Datainnsamling med XForms i Dynamic Presentation Generator

Morten Høiland

Institutt for informatikk
Universitetet i Bergen
Norge

Lang Masteroppgave
2011

Forord

Denne masteroppgaven er resultatet av kandidatens mastergrad i informatikk ved Universitetet i Bergen.

Oppgaven undersøker mulighetene for kunne samle inn data i Dynamic Presentation Generator 2.1 ved hjelp av XForms.

Kandidaten vil gjerne takke Tobias Rusås Olsen, Ole Henning Vårdal, Kelly Alexander Whiteley, Aleksander Waage og Aleksander Vines som alle er tilknyttet JAFU. En spesiell takk til Øystein Lund Rolland, Jostein Bjørge og Peder Lång Skeidsvoll for å ha korrekturlest oppgaven. Alle de nevnte har gjort tiden som masterstudent til en svært trivelig og minnerik opplevelse. Kandidaten vil gi en stor takk til veilederene for oppgaven, Khalid Azim Mughal og Torill Hamre. De har gjort en god innsats for å gjøre denne oppgaven mulig. En stor takk går også til Haakon Nilsen som har holdt infrastrukturen i orden og bidratt med viktige innspill.

Til slutt vil kandidaten takke familien for støtten gjennom hele studietiden, og spesielt Elisabeth Høiland som har korrekturlest oppgaven.

Morten Høiland
Bergen, 1. juni 2011

Innhold

Figurer

Tabeller

Eksempler

Forkortelser

AJAX:	Asynchronous JavaScript and XML [26]
API:	Application Programming Interface
CMS:	Content Management System [15]
DOM:	Document Object Model [79]
DPG:	Dynamic Presentation Generator [68]
HTML:	Hypertext Markup Language [81]
JAFU:	JAva for FjernUndervisning [32]
JCR:	Java Content Repository [6]
PCE:	Presentation Content Editor [68]
PM:	Presentation Manager [68]
PV:	Presentation Viewer [68]
UiB:	Universitetet i Bergen
W3C:	World Wide Web Consortium [82]
XHTML:	Extensible Hypertext Markup Language [83]
XSLT:	Extensible Stylesheet Language Transformations [88]
XPL:	Extensible Stylesheet Language Transformations [19]
XSLT:	XML Pipeline Language [88]
XML:	Extensible Markup Language [80]
RPC:	Remote Procedure Call [46]
PPDev:	Presentation Pattern Development [10]
JDOM:	Java Document Object Model [30]

1 Introduksjon

1.1 Innledning

JAva FjernUndervisning (JAFU) er et prosjekt som tilbyr nettbasert undervisning ved Institutt for informatikk, Universitetet i Bergen (UiB). JAFU har siden 1999 drevet med fjernundervisning, og tilbyr to fag. Det ene er INF-100F - *Grunnkurs i programmering*. Det andre er INF-101F - *Videregående programmering*.

Fjernundervisningen blir utført ved at foreleseren publiserer kursinformasjon, pensum, oppgaver, nyheter og notater på kurssidene. Etterhvert ble det vanskelig å bare bruke statiske nettsider til å presentere kurssidene. Derfor ble det utviklet en dynamisk løsning.

I 2004 laget masterstudenten Kevin Cruickshanks den første applikasjonen som brukte *presentasjonsmønster*. Applikasjon ble kalt *Java Presentation Generator* (JPG) og var knyttet til fjernundervisningen. Presentasjonsmønster er beskrevet av Khalid A. Mughal i artikkelen *Presentation Patterns: Composing Web-based Presentations* [52]. Presentasjonsmønster er et konsept som skiller innholdet fra strukturen på innholdet. I 2005 utviklet masterstudenten Yngve Espelid den første *Presentation Dynamic Generator* (DPG 1.0). DPG 1.0 bygger på samme konsept som JPG. DPG 1.0 ble brukt i i fjernundervisningen fra 2006 til 2009.

DPG 1.0 hadde en del mangler og svakheter. Derfor bestemte masterstudentene Ka-

rianne Berg [8], Bjørn C. Sebak [68] og Bjørn O. Ingvaldsen [31] seg for å utvikle en ny utgave av DPG, kalt DPG 2.0. DPG 2.0 ble bygget på de samme konseptene som de forrige applikasjonene, men ble utviklet med flere moduler, har ny presentasjonsmønsterspesifikasjon, god testdekning og kjente teknologier. Applikasjonen er blitt brukt i fjernundervisningen i 2010. Den viste seg å være enkel å bruke.

I 2010 ble DPG 2.0 videreutviklet av Peder Skeidsvoll [70] og Tobias Olsen [54]. De fjernet innebygde typer fra applikasjonen, som for eksempel `String` og `date`. I stedet for innebygde typer, brukte den nye versjonen plugins. De utviklet også plugin-arkitekturen til å kunne bruke AJAX [26] og støtte for interaksjon med brukeren. Den nye versjonen ble kalt DPG 2.1. DPG 2.1 har blitt brukt i fjernundervisningen i 2011.

Denne masteroppgaven bygger på DPG 2.1. Når kandidaten referer til DPG videre i oppgaven, vil det vise til DPG 2.1.

1.2 Mål for oppgaven

1.2.1 Overordnet mål

Det overordnede målet for masteroppgaven er å tilby støtte for datainnsamling med XForms i DPG. Det vil si å kunne presentere XForms-skjema og samle inn data, for så å lagre dataene.

1.2.2 Delmål

For å kunne nå det overornede målet må følgende delmål være oppnådd:

1. Vurdere de forskjellige mulighetene for å integrere XForms i DPG.
2. Utvikle eller tilpasse biblioteker som støtter XForm-spesifikasjonen, som kan integreres i DPG.
3. Vurdere muligheter for å motta og lagre XForms-inndata i DPG på en god og sikker måte.
4. Gjøre nødvendige endringer i DPG basert på disse vurderingene for å oppnå støtte for XForms-skjema.

1.2.3 Motivasjon

Motivasjonen er å få utvidet funksjonaliteten til DPG slik at man kan støtte datainnsamling med XForms. DPG vil da kunne brukes i større grad til å samle inn informasjon. Man kan også bruke XForms-skjema fra andre applikasjoner som bruker XForms. Data samlet fra XForms-skjema i DPG skal kunne lagres på en måte som gjør det enkelt å dele dataene med andre systemer.

1.3 Organisering av oppgaven

Oppgaven er delt opp i 8 kapitler, som tar for seg hvert sitt hovedtema. Det finnes to tillegg (appendikser) bakerst i oppgaven. Tilleggene inneholder JavaScript-ene som blir lagt til i hovedmalen for et presentasjonsmønster og et XForms-skjema eksempel.

Kapittel 2: Bakgrunn og problembeskrivelse

Dette kapittelet presenterer viktige konsepter som har med hvordan applikasjonen er bygd opp. Viktige konsepter i kapittelet er presentasjonsmønster, presentasjonsmønsterspesifikasjonen, DPG og innholdshåndteringssystem. Det blir gitt en innføring i teknologiene bak DPG. Alt dette er viktig for at leseren lettere skal forstå resten av oppgaven.

Kapittel 3: Integreringsmuligheter i DPG

Plugin-arkitekturen blir i dette kapittelet gjennomgått og vurdert. Vurderingen går ut på å undersøke om kandidaten kan integrere XForms ved hjelp av plugin-arkitekturen.

Kapittel 4: Xforms designer og rendrer

Dette kapittelet presenterer eksempel på XForms-skjema og fordeler med å bruke det i forhold til HTML-skjema. Det blir også gjennomgått hvordan XForms-rendreren og XForms-designeren skal brukes sammen. I kapittel 6 blir det forklart hvordan XForms-rendreren fungerer i DPG. XForms-designeren i DPG vil bli forklart i masteroppgaven til Øystein L. Rolland [67].

Kapittel 5: Støtte for XForms i DPG

I dette kapittelet vil flere XForms-implementasjoner bli gjennomgått og vurderte. Grunnen til at man trenger en egen implementasjon for å bruke XForms-skjema, er fordi alle de vanligste nettleserene ikke støtter XForms.

Kapittel 6: Implementasjon av XFormsPlugin

Dette kapittelet forklarer selve integreringen av støtte for XForms i DPG. XForms-implementasjonen valgt fra kapittel 5 blir integrert i DPG ved bruk av plugin-arkitekturen. Hele denne prossesen blir gjennomgått i kapittelet.

Kapittel 7: Vurdering av plugin-ressurser

Kapittelet presenterer svakheter med den eksisterende ressurshåndtering for plugins i DPG. Det vil så bli foreslått løsninger for svakhetene og begrunnet valg av nye løsninger.

Kapittel 8: Evaluering, konklusjon og videre arbeid

I siste kapittelet presenteres en konklusjon og vurderinger av målene for oppgaven. Det vil bli foreslått ideer til videre arbeid og en vurdering av teknologier som kandidaten mener kan være nyttige i videreutviklingen av DPG.

2

Bakgrunn og problembeskrivelse

2.1 Innholdshåndteringssystem

Et *Innholdshåndteringssystem* (eng. *Content Managment System*) er en programvare som organiserer forskjellige typer innhold. De forskjellige typene innhold kan være tekst, bilder, lyd, video og andre typer som kan lagres digitalt.

Et innholdshåndteringsystem som organiserer innhold og presenterer det på en nettside kalles et *internettinnholdssystem*. Eksempler på populære internettinnholdssystem er Wordpress [91] og Movable Type [76]. Målet med innholdshåndteringssystemer er at det ikke kreves at brukerene kjenner til teknologier som for eksempel HTML [81] og CSS. De er ikke nødvendig med kunnskaper om teknologien bak systemet, fordi systemet tilbyr et brukergrensesnitt som gjør det enkelt å organisere innhold. Organisering av innhold vil si å lage, slette, endre og publisere de forskjellige typer av innhold. Det som er fordelen med et internettinnholdssystem, er at man enkelt kan publisere forskjellige typer innhold på Internett.

DPG er et internettinnholdssystem. En viktig del av innholdshåndteringssystemer er hvordan innhold blir strukturert. Det er vesentlig at innhold har en logisk struktur som gir stabilitet og fleksibilitet. I dette kapittelet vil det bli gjennomgått konseptene som DPG er bygd på.

2.2 Presentasjonsmønster

I 2003 introduserte Khalid A. Mughal [52] konseptet *presentasjonsmønster*. Presentasjonsmønstre beskriver følgende:

- Innhold og struktur skal være separerte.
- Man skal kunne endre strukturen, uten å forandre innholdet.
- Man skal kunne endre innholdet, uten å forandre strukturen.

Et eksempel som kunne hatt nytte av denne tankegangen er politiet. Politiet brukte i 2009 25,8 millioner kroner på nye internettsider [77]. En av unnskyldningene var at de måtte utvikle sider for hvert politidistrikt. Hvis de hadde brukt presentasjonsmønster, kunne de brukt samme struktur på presentasjonen for alle sidene til distriktene. Da kunne man i hver enkelt distrikt bare skiftet ut design og innhold. Det er også mulig å gjenbruke innhold i andre deler av strukturen.

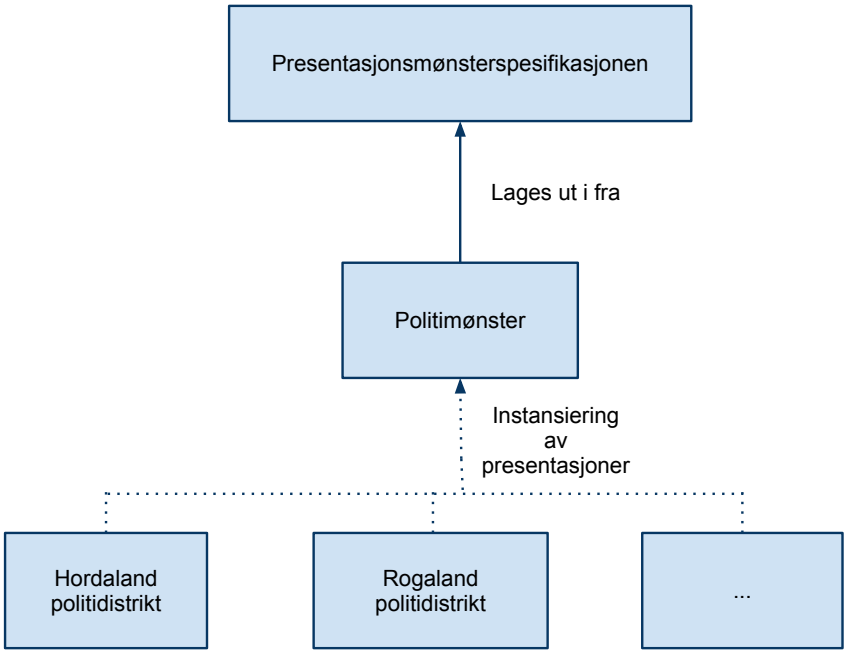
2.3 Presentasjonsmønsterspesifikasjonen

Presentasjonsmønsterspesifikasjonen setter begrensninger på presentasjonsmønsteret. I figur 2.1 blir det vist hvordan `Politimønster` blir laget ut i fra presentasjonsmønsterspesifikasjonen. I figur 2.1 er `Politimønster` et presentasjonsmønster. Presentasjonsmønsteret følger regelsettet til presentasjonsmønsterspesifikasjonen om hvordan syntaksen skal være og hvilke elementer som kan benyttes. Presentasjonsmønsteret kan instansierest til flere *presentasjoner*. Det blir vist i figur 2.1 med `Rogaland politidistrikt` og `Hordaland politidistrikt`, som er instanser av `Politimønster`. Man kan definere hvor mange instanser av presentasjonsmønsteret man ønsker.

2.3.1 Oppbygningen i et presentasjonsmønster

Presentasjonsmønsterspesifikasjonen har fire hovedbestanddeler som presentasjonsmønster blir bygget på. De er som følger:

- *entitet* (eng.*entity*)
- *entitetsinstans* (eng. *entity-instance*)
- *utsnitt* (eng. *view*)



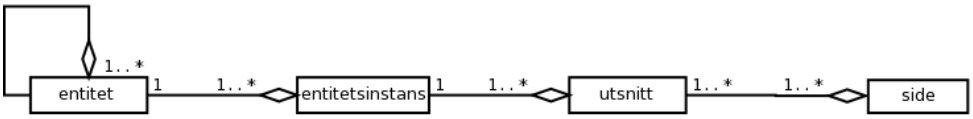
Figur 2.1: Instansiering av presentasjoner

- *side* (eng. *page*)

Figur 2.2 viser begrensningene mellom de fire elementene i presentasjonsmønsteret. En entitet kan inneholde flere entiteter i en liste. En entitetsinstans kan bare ha en entitet. Entitetsinstansen kan kan ha mange utsnitt. Ett utsnitt kan bare ha en entitetinstans. Ett utsnitt kan presenteres på flere sider. En side kan ha flere utsnitt.

En entitet definerer strukturen til data. En entitet kan bestå av ett eller flere *felt* (eng. *field*). Felt er i DPG 2.1 av typen plugins. Attributtet `type` i felt, definerer hvilken plugin som skal håndtere feltet. Lister av entiteter blir også håndtert av plugins.

Entitetsinstanser er instanser av entiteter. Man kan bare ha en entitet, men mange instanser av en entitet. Innholdet i entitetinstansene har samme struktur, men selve innholdet kan være forskjellig.



Figur 2.2: Forholdet mellom elementer i presetansjonsmønster

Utsnitt har en referanse til en entitetsinstans. Utsnittet har også en referanse til en transformasjon. Det er mulig å bruke forskjellige transformasjoner. Det er utsnittets oppgave å bestemme hvordan en entitetsinstans skal presenteres. Man kan bruke forskjellige transformasjoner til samme entitetsinstans. Dette gjør det mulig å presentere innhold ulikt, selv om innholdet er det samme. Et eksempel kan være at man har et utsnitt som viser politiet sine tjenester sortert etter hvilke som er mest brukt. Eksempler på tjenester kan være anmeldelse av tyveri eller å tipse politiet. Man kan så ha et utsnitt som sorterer tjenestene alfabetisk.

Side er det øverste elementet i hierarkiet. Det definerer hvordan man setter sammen transformerte utsnitt. Dette blir utført etter en *mal* (eng. *page-template*) som definerer hvor hvert utsnitt skal være plassert i en nettside.

2.4 Dynamic Presetation Generator 2.1

Dynamic Presentation Generator 2.1 (DPG) er et internettinnholdssystem utviklet av flere studenter. Studentene har vært tilknyttet JAFU-prosjektet ved Institutt for Informatikk ved UiB. DPG bygger på konseptene til presentasjonsmønster.

Figur 2.3 viser hvordan DPG bruker `pattern.xml` til å implementere presentasjonsmønster. I figuren er `pattern.xml`, filen til høyre, og den definerer siden som er til venstre. Det blir brukt Extensible Markup Language (XML) for å håndtere begrensningene presentasjonsmønsterspesifikasjonen setter. I figur 2.3 er det vist følgende:

- `address` og `name` er begge felter, som blir satt til å være av plugin-typen `String`. De blir definert i `pattern.xml` på linje 11 og 12.
- `url` er en entitet som har har to felter, `address` og `name`. `url` skal representere en lenke. Den har et navn på lenken og en lenkeadresse.
- `urlEntityInstance` er instanseentiteten til `url`. Den blir definert på linje 16-20. På linje 18 ser vi hvor referansen til `url` blir definert. Ut i fra

`urlEntityInstance` i siden, ser vi en pil som går til `urlEntityInstance.xml`. Det er denne filen `address` og `name` sitt innhold blir lagret i.

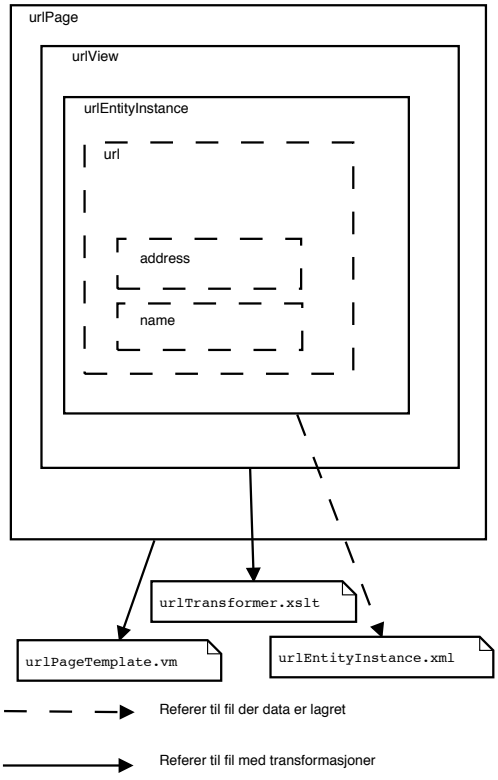
- `urlView` er utsnittet som blir definert på linje 23-29. På linje 25-26 er referansen til hvilken entitetinstans den skal transformere til HTML. Linje 27-28 viser referansen til filen som håndterer transformasjonen. Det blir og vist i siden til venstre med en pil til `urlTransformer.xslt`.
- `urlPage` er siden som blir definert på linje 33-41. På linje 36-37 er referansen til `urlView`. Ser man på figuren over siden, peker det en pil ned på `urlPageTemplate.vm`. Dette er Velocity-malen som definerer hvor på siden `url` skal vises. Malen blir referert til i `pattern.xml` på linje 39-40.

Figur 2.3 viser en forenklet utgave av et presentasjonsmønster. I figuren er det bare definert en enitet med to felt, en instanseentitet, ett utsnitt og en side. Hadde et presentasjonsmønster som var i bruk blitt vist, ville det vært mange flere elementer med i presentasjonen. Det er valgt å bruke de engelske navnene, siden det er konvensjonen når man lager et presentasjonsmønster.

2.4.1 Viktige teknologier i DPG

I DPG blir det brukt mange forskjellige teknologier. De teknologiene som er vesentlige blir forklart kort her:

- XML [80] blir brukt for å definere presentasjonsmønster og til å håndtere begrensningene presentasjonsmønsterspesifikasjonen setter. Det blir også brukt til å lagre innholdte i entitetinstansene.
- Jackrabbit [6] er implementasjonen av Java Content Repository (JCR-170). Det brukes for lagring og henting av innhold. Bruker XML for lagring og XPath [84] som spørrespråk. DPG bruker filsystemet for lagring under utvikling, fordi det er enklere å endre på filer. De to løsningene fungerer på en lignende måte. De bruker begge stier til ressurser.
- Extensible Stylesheet language transformation (XSLT) [88] blir brukt til å transformere utsnitt fra XML til HTML.
- Velocity-maler [22] komponerer utsnitt sammen til en side.
- Spring [73] blir brukt i DPG for sikkerhet,for å oppnå Model-View-Controller (MVC)-arkitektur og Dependecy Injection [24] for modularitet.

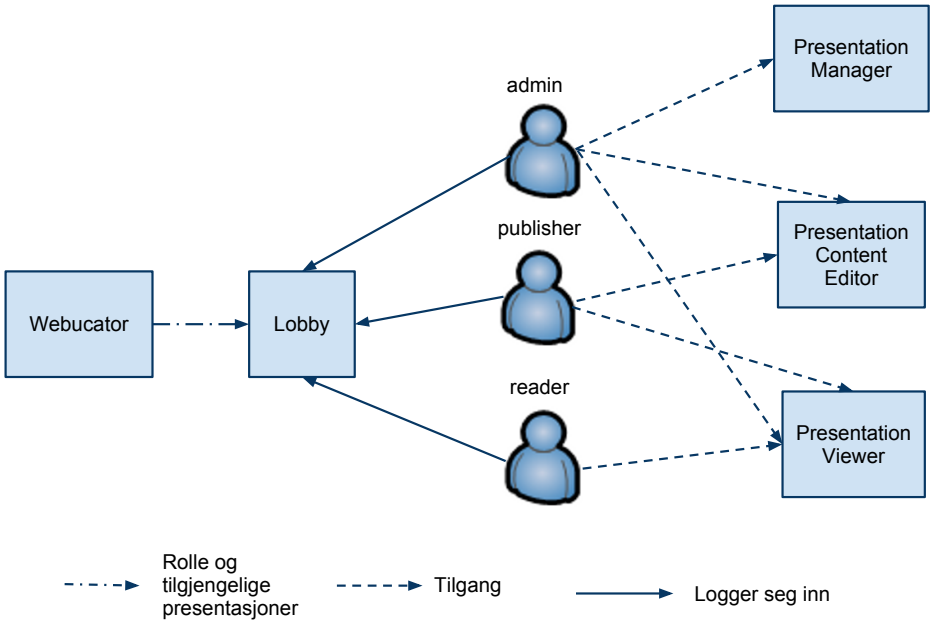



```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <specification>
4   <title>URL pattern</title>
5   <description>Viser hvordan
6    pattern.xml fungerer
7    </description>
8
9  <entities>
10    <entity id="url">
11      <field type="string">address</field>
12      <field type="string">name</field>
13    </entity>
14  </entities>
15
16  <entity-instances>
17    <entity-instance id="urlEntityInstance">
18       <entity-ref>url</entity-ref>
19    </entity-instance>
20  </entity-instances>
21
22  <views>
23    <view id="urlView">
24        <description>Viser url</description>
25        <entity-instance-ref>urlEntityInstance
26        </entity-instance-ref>
27        <transformation>urlTransformer
28        </transformation>
29     </view>
30   </views>
31
32  <pages>
33     <page id="urlPage" default="true">
34        <description>URL</description>
35        <composition>
36           <view-ref defaultView="true">urlView
37           </view-ref>
36        </composition>
39        <page-template>urlPageTemplate
40        </page-template>
41      </page>
42   </pages>
43
44</specification>
```

Figur 2.3: Hvordan DPG bruker presentasjonsmønster

2.4.2 Delsystem i DPG

DPG er delt opp i fire delsystemer. De kalles *Presentation Content Editor*, *Presentation Viewer*, *Lobby* og *Presentation Manager*. DPG har tre brukerroller, *admin*, *publisher* og *reader*. Brukerrollene er tett koblet opp imot delsystemene. Det vises i figur 2.4. *Webucator* i figuren er et eksternt brukerhåndteringssystem, som forteller hvilke roller brukerene har og hvilke presentasjoner de har tilgang på.



Figur 2.4: Oversikt over brukere og systemer i DPG

Lobby

Dette delsystemet tar seg av innlogging av brukeren. I figur 2.4 viser man at Lobby logger inn brukerne med *Webucator*, som får vite hvilken rolle de har og hvilke presentasjoner de har tilgang til. Lobby viser presentasjoner man har tilgang til etter innlogging. Hvis man har rettigheter til å endre presentasjoner, får man opp lenker til å gjøre det. Har brukeren for eksempel rollen *publisher*, vil det være lenker til å redigere innhold.

Presentation Manager

Det er bare *admin* som har tilgang til dette delsystemet. Det brukes til å håndtere forskjellige presentasjoner. Man kan opprette nye presentasjoner ut i fra eksisterende presentasjonsmønster, redigere presentasjoner og slette presentasjoner.

Presentation Content Editor

Figur 2.4 viser at brukerrollene *admin* og *publisher* har tilgang til Presentation Content Editor (PCE). Dette delsystemet brukes til å endre innholdet i presentasjonen. Man kan for eksempel legge ut filer eller publisere bilder.

Presentation Viewer

Dette delsystemet tar seg av transformeringen av XML til HTML med bruk av XSLT. Presentation Viewer (PV) tar også hånd om å sette sammen utsnittene ved hjelp av Velocity-malene. Det er dette delsystemet som rendrer innholdet for brukeren. I figur 2.4 ser man at alle brukerrollene har tilgang til dette systemet. DPG legger opp til at *admin* administrer presentasjonene. Brukerrollen *publisher* legger ut innhold. Brukere med *reader*-rollen leser og laster ned innhold.

2.4.3 Plugins

DPG har en plugin-arkitektur som gjør at man kan legge til ny funksjonalitet uten å måtte forandre hele systemet. *Plugins* blir forklart i kapittel 3. Der blir også plugin-arkitekturen nøye gjennomgått.

2.5 Problembeskrivelse

2.5.1 Datainnsamling i DPG med XForms

DPG 2.1 har muligheten til å interaktere med brukerene. Det er laget eksempler på denne funksjonaliteten gjennom en plugin som lar brukerene skrive inn kommentarer på sider og en plugin som lar brukerene stemme i en spørreundersøkelse [54]. Skal man samle inn mer komplekse data blir det tungvindt å bruke den eksisterende løsningen. Et eksempel på komplekse skjemaer kan være å ha en fagevaluering når semesteret er over. DPG har blitt brukt til fjernundervisning i fagene INF-100F og INF-101F. Foreleser vil ha tilbakemelding fra studentene om hva de syntes var bra og hva som kunne blitt gjort bedre. Dette eksemplet viser at det er interessant å kunne tilby innsamling av mer komplekse inndata, enn DPG har muligheten til med HTML-skjemaer [81].

XForms kan være en god kandidat til å gi DPG denne funksjonaliteten. XForms-skjema [86] blir definert og leverer inndata som XML. Dette passer bra med DPG sitt

paradigme med gjenbruk av data. Det er også viktig å undersøke andre teknologier som tilbyr innsamling av data. For å vurdere hvor god XForms er eller om det bør brukes enn annen teknologi.

OpenXdata [55] er en web-applikasjon som bruker XForms-skjemaer. Applikasjonen blir brukt til å samle inn data fra forskjellige felter, blant annet helse og utdanning. Jafu har en visjon om å kunne samarbeide med OpenXdata, slik at DPG kan bruke XForms-skjema fra OpenXdata og samle inn data for OpenXdata. DPG må ha mulighetene å dele dataene som blir samlet inn, med andre applikasjoner. Derfor er det viktig å lagre data på en god måte, slik at de lett kan deles.

2.5.2 Plugin-arkitekturen

Plugin-arkitekturen er laget for å kunne utvide funksjonaliteten til DPG, uten å gjøre kjernen større. Man skal kunne integrere nye teknologier sømløst. Det gjør det enkelt å fjerne funksjonalitet man ikke trenger og legge til ny funksjonalitet.

Kandidaten ønsker å finne ut om plugin-arkitekturen i DPG kan integrere bruk av XForms-skjema. Hvis den ikke kan det, vil det bli funnet andre løsninger eller det vil bli gjort endringer av plugin-arkitekturen som kan gjøre bruk av XForms-skjema mulig.

2.5.3 Håndtering av plugins-ressurser

Plugin-arkitekturen er god i DPG, men håndteringen av ressurser har mangler. Det er for eksempel ikke satt noen føringer på hvordan man skal lagre ressurser, og plugins har tilgang til hverandre sine ressurser. Under utviklingen av DPG 2.1, var det mest fokus på at plugins skulle ha muligheten til å håndtere sine egne ressurser, ikke hvordan det skulle håndteres. Det er derfor en del svakheter som må ordnes. Man kan lese mer om det i masteroppgavene til Tobias Olsen [54] og Peder Skeidsvoll [70].

En viktig del av innsamling av data er at man kan lagre data på en sikker måte. Det skal og være lett å bruke disse dataene, både for DPG og for andre applikasjoner. Det vil bli gjort en vurdering av den eksisterende løsningen for håndtering av plugin-ressurser. Er det svakheter som skaper problemer for lagringen eller innsamlingen av data, vil kandidaten finne mulige løsningner for disse. Dette innebærer å vurdere løsningene og begrunne valget for den som er mest egnet, for så å utvikle løsningen.

3

Integreringsmuligheter i DPG

DPG har gode integreringsmuligheter med sin plugin-arkitektur. Den ble først utviklet i DPG 2.0 av Bjørn Ove Ingvaldsen i hans masteroppgave [31]. Den ble så videreutviklet i DPG 2.1 av Tobias Olsen [54] og Peder Skeidsvoll [70] i deres masteroppgaver. Plugin-arkitekturen baserer seg på plugin-mønsteret til Martin Fowler [25]. Mønsteret bruker *kontrakter* (eng. *interfaces*) som plugins må implementere. I DPG 2.1 er det én slik kontrakt, `FieldPlugin`.

XForms er konstruert med XML (se avsnitt 4.3 for utdypning). Kjernen til DPG bygger også på XML. Det er derfor en mulig løsning å kunne bruke *felter* (eng. *fields*) som har funksjonaliteten til forskjellige XForms-funksjoner. For eksempel er det mulig å lage et felt som gir datoelementet slik XForms-spesifikasjonen definerer det. Dette hadde gjort det mulig å bygge et XForms-skjema ved hjelp av DPG sin struktur. Denne løsningen kobler de forskjellige teknologiene tett sammen. Det blir vanskelig å fjerne XForms-delen om man ikke ønsker å bruke den lenger eller å skifte den ut. Løsningen vil bli veldig komplisert med alle pluginsene som skal tilsvare all funksjonaliteten til XForms-spesifikasjonen. Denne løsningen er derfor ugunstig. Man ønsker heller ikke spesialtilfeller i DPG-arkitekturen. Derfor er løsninger der man bruker forskjellige typer utsnitt og sider heller ikke aktuelle. Løsninger med forskjellige typer sider og utsnitt kan for eksempel være at XForms-skjemaet er en side eller et utsnitt, men man må spesialhåndtere disse utsnittene og sidene for å rendre de i DPG.

Valget blir å undersøke hva plugins-arkitekturen kan gjøre og hva den ikke kan gjøre. Deretter vil kandidaten kunne ta et valg om å benytte plugin-arkitekturen som den er, eller forandre den. Forandringer vil bli gjort om det er nødvendig for å støtte XForms-skjema i DPG.

3.1 Plugin-arkitekturen i DPG

Figur 3.1 viser plugin-arkitekturen i DPG. DPG laster inn plugins med følgende steg:

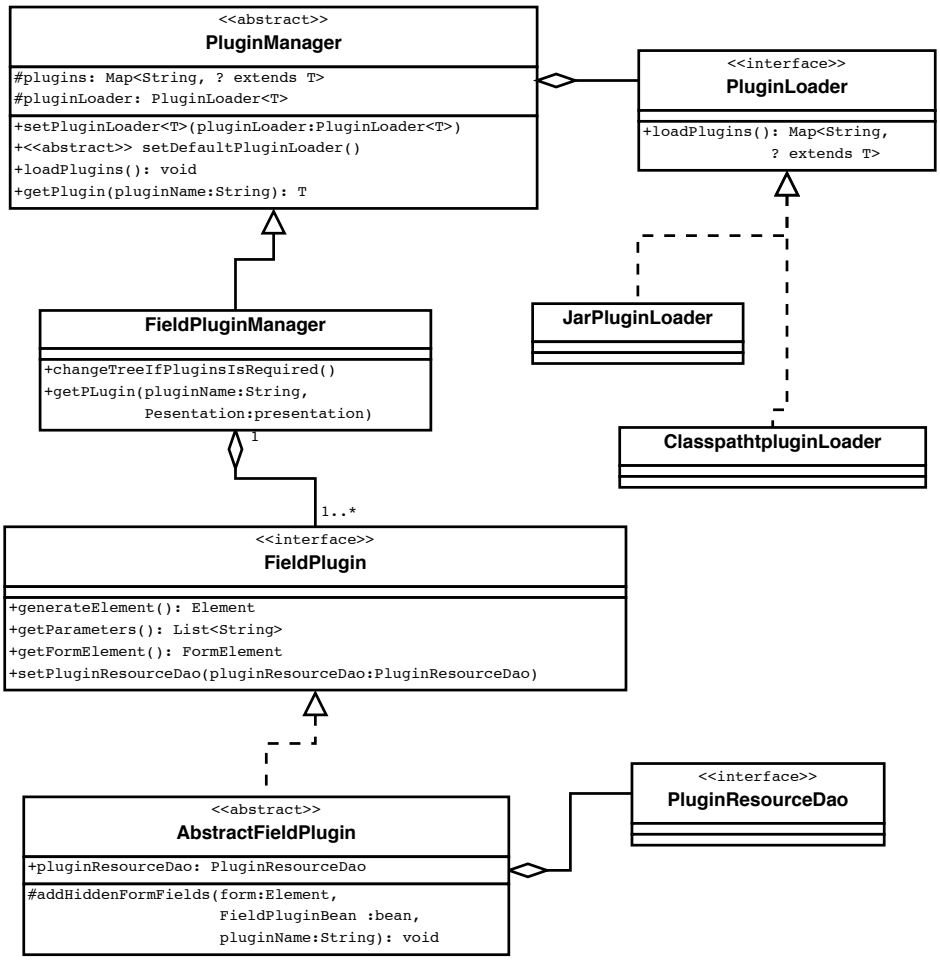
1. DPG starter opp `PluginManager`.
2. `PluginManager` laster inn plugins med `PluginLoader`.
3. Alle plugins som implementerer `FieldPlugin` og er i `classpath`-en blir lastet inn. Dette er fordi DPG bruker klassen `ClasspathPluginLoader`.

Innlastingen av plugins er generisk. Det betyr at man kan lage andre typer plugins enn den ene kontrakten DPG tilbyr. Vil man lage en plugin til DPG, må man implementere `FieldPlugin`-kontrakten eller arve fra `AbstractFieldPlugin`-klassen. `AbstractFieldPlugin`-klassen har mange metoder som kan gjøre utvikling av plugins enklere. Eksempel på det er `addHiddenFormFields( ... )`, som legger til skjulte felter (eng. *hidden fields*) til HTML-skjemaer [81]. Disse skjulte feltene inneholder informasjon om hvilken side og utsnitt inndataene er hentet fra. Denne informasjonen trenger DPG for å vite hvilken side og utsnitt som skal rendrest, når skjemaet er sendt inn.

Plugins kan bli lastet inn fra en egen mappe som ligger i `lib/plugins`. Da må man bruke `JarPluginLoader`, som implementerer `PluginLoader`-kontrakten. Dette vises i figur 3.1. Man må også pakke pluginen i en JAR-fil og legge den i `lib/plugins`.

`FieldPlugin` har følgende metoder:

- `generateElement(FieldPluginBean bean, Map<String, Object> input)` er metoden DPG kaller for å få JDOM-elementet, som blir rendret i et utsnitt.
- `getParameters()` returnerer en liste med navn til parameterene som pluginen tar inn. Detter er for plugin-utvikleren skal vite hvilke parametere som er tilgjengelige.
- `getFormElement(String value, Field field)` returnerer et passende `FormElement` for denne pluginen. Eksempel på det er `FileField`, om pluginen skal ta imot en fil.



Figur 3.1: Plugin-arkitekturen i DPG

- `getXmlContent(FormElement formElement, PluginContext context)` returnerer en XML-presentasjon av `FormElement`. For eksempel en plugin som tar imot filer, returnerer navnet til filen i XML.
- `setPluginResourceDao(PluginResourceDao pluginResourceDao)` er en metode for å instansiere `PluginResourceDao` i plugins. Kontrakten `PluginResourceDao` lar plugins lagre, hente og sjekke ressurser. Plugins har en egen mappe for ressurser.

Metodene `getFormElement` og `getXmlContent` er ment for bruk av PCE-delen av DPG. De blir brukt når man skal forandre på innholdet pluginen skal presentere. Den mest interessante metoden til `FieldPlugin` er `generateElement`. Metoden gjør det mulig å returnere et *Java Document Object Model* (JDOM)-element [30].

JDOM-elementet representerer en XML-*markering* (eng. *tag*) med tekst, som kan tilknyttes underelementer og attributer. JDOM-elementet blir så plassert i utsnittet pluginen er definert i. DPG rendrer så utsnittet i en side og JDOM-elementet blir vist til brukeren. Hadde alle de mest brukte nettleserene støttet XForms, kunne man lagt til hele XForms-skjemaet i JDOM-elementet. Da ville man kunne rendret XForms-skjemaet på lik linje med HTML-skjema i nettleseren.

En ny type plugin er enkelt å lage, fordi innlastingen er generisk. For å lage ny type plugin må man gjøre følgende:

- Lage en ny kontrakt som plugins kan implementere.
- Lage en ny klasse som arver fra `PluginManager`. Den må laste inn plugins av den nye typen. Klassen må ha en metode som kjører selve pluginen.
- Tilpasse DPG slik at den nye `PluginManager`-klassen kan bli brukt.

Det siste steget er tidkrevende, fordi man må tilpasse hvordan DPG håndterer den nye typen plugin. Det kan bli komplisert, for plugin-arkitekturen er en viktig del av hvordan DPG fungerer. Hvor vanskelig det blir, er avhengig av hva den nye plugin-typen skal gjøre. Det mest naturlige er at den lager et utsnitt, som blir returnert til DPG. Da kan man også legge til andre utsnitt fra DPG, fordi en side kan ha flere utsnitt. Denne løsningen bryter med DPG sitt paradigme som bygger på gjenbruk. Det er fordi man kan ikke tvinge plugins til å bruke en fast struktur på dataene den vil rendre. Plugins vil som oftest bare hardkode inn hvordan dataene skal strukturerest i presentasjonen. Derfor vil det bli vanskelig å bruke dataene om igjen.

Det er i tillegg mulig å lage en løsning som ligner denne med den eksisterende plugin-arkitekturen. Har man et utsnitt som bare har en *entitet* (eng.*entity*), med et *felt* (eng.*field*), som er en plugin. Da vil dette være en tilsvarende løsning, som å lage en plugin som returnerer et utsnitt. Man må også definere en ny type utsnitt for utsnitt fra plugins. Det blir et spesialtilfelle, fordi utsnitt fra plugins må håndteres annerledes enn hvordan utsnitt blir håndtert i DPG. Det er bare felt som blir definert som plugins i DPG, og ikke andre deler som utsnitt og sider. For at det ikke skal bli et spesialtilfelle, må man gjøre vesentlige forandringer på presetentasjonsmønsterspesifikasjonen. Dette er noe som er tidkrevende og svært intrikat, fordi reglene og syntaksen for presentasjonsmønster må være logisk og lett å forstå.

3.1.1 Konfigurasjon av plugins

Plugins må bli definert i filene `pattern.xml`, `entitetsIntansen.xml` og `entitetsTransformasjon.xslt`. Skal pluginen ha muligheten til å ta imot parametere må den defineres i `pluginConfig.xml`.

3.1.2 Definere plugins i pluginConfig

Er ikke pluginen definert i `pluginConfig.xml`, vil man kunne bruke pluginen sitt navn der den skal defineres. Navnet på pluginen blir satt som en *annotasjon* (eng. *annotation*) [53], for eksempel `@PluginName("string")`. Hvis man ikke bruker annotasjonen blir plugin-navnet satt til klassenavnet. I listing 3.1 blir pluginen, `XFormsPlugin`, definert som `xform`. På linje 2 i listing 3.1, blir parameteren `filePath` definert til å ha filstien `/Users/morten/Desktop/index.xhtml` som verdi. Man kan velge hvor mange parametere man vil ha. Navnene på disse parameterene som skal returnerest i metoden `getParameters()` i `FieldPlugin`-kontrakten.

Listing 3.1: Definering av plugin i pluginConfig.xml

```
1 <plugin-config id="xform" plugin-name="XFormsPlugin">
2     <param name="filePath">/Users/morten/Desktop/index.xhtml</param>
3 </plugin-config>
```

3.1.3 Definere plugins i pattern.xml

Nå som pluginen er definert med ID-en `xform`, må det brukes videre i `pattern.xml`. Pluginen defineres som et felt i en entitet. På linje 8 i listing 3.2 blir pluginen definert. Den vil da høre til entiteten `contactEntity`.

Listing 3.2: Definere plugins i pattern.xml

```
1 <entity id="contactEntity">
2     <field type="string" required="true">name</field>
3     <field type="string">role</field>
4     <field type="string">email</field>
5     <field type="string">phoneNumber</field>
6     <field type="string">homePage</field>
7     <field type="bildePlugin">image</field>
8     <field type="xform">xform</field>
9 </entity>
```

3.1.4 Definere plugins i contactEntityInstance.xml

Plugins må defineres i entitetsinstansen til entiteten. Entiteten `contactEntity`, har instansen som heter `contactEntityInstance`. Instansen vil ha andre navn

om man vil definere plugins i andre entiteter. Hver entitet har en entitetinstanse. Listing 3.3 viser definisjonen av pluginen på linje 9. Elementet `xform` refererer til `XFormsPlugin`, som blir vist i listing 3.1. Man må til å skrive feltet inn for hånd i filen `contactEntityInstance.xml`. Dette er den eneste filen man må definere pluginen for hånd, som ikke hører til selve mønsteret, men presentasjonen. At man må gjøre dette er unødvendig, for instansene av entitet vil alltid ha alle feltene til entiteten.

I listing 3.3 ser man på linje 9, at `xform`-elementet har "`none`" som innhold. Dette er fordi den skal rendre et JDOM-element og ikke tekst eller et bilde, slik som elementene `email` og `image` gjør. Feltet `xform` kan heller ikke bli forandret i PCE-en, derfor har den teksten "`none`".

Listing 3.3: Definere plugins i contactEntityInstance.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <contactEntity>
3    <name required="true" type="string">mortenporten</name>
4    <role type="string">gruppeleder</role>
5    <email type="string">mortenporten</email>
6    <phoneNumber type="string">44554455</phoneNumber>
7    <homePage type="string">saiboten.com</homePage>
8    <image type="bildePlugin">tobsama.jpg</image>
9    <xform type="xform">none</xform>
10 </contactEntity>
```

3.1.5 Definere plugins i contactTransformer.xslt

Hver entitet har en egen XSLT-transformasjon, som blir definert i `pattern.xml`. Dette ble forklart i kapittel 2. Entiteten `contactEntity` har transformasjonen `contactTransformer.xslt`. Listing 3.4 viser hvordan `xform` blir definert som et `div`-element med klassenavnet `xform`. Linje 2 er hvor man legger til alle elementene som er under feltet `xform`. Det som ligger under `xform` er JDOM-elementet, som `XFormsPlugin`-klassen returnerer med metoden `generateElement()`.

Listing 3.4: Definere plugins i contactTransformer.xslt

```
1  <div class="xform">
2    <xsl:copy-of select="xform/*" />
3  </div>
```

3.1.6 Svakheter med plugin-arkitekturen

`PluginManager` er en abstrakt klasse. Hvilke metoder den har blir vist i figur 3.1. Den mangler den viktigste metoden `changeTreeIfPluginIsRequired`, og er metoden DPG kaller for å legge til JDOM-elementet fra plugins. Denne metoden blir implementert i `FieldPluginManager`, som arver fra `PluginManager`. Det ville vært bedre om `PluginManager`-klassen var kontrakt som man implementerte. Den kontrakten innholdt den metoden som DPG kalte. Da kunne man enkelt ha skiftet ut implementasjonen av `PluginManager`-kontrakten. Det vil også gjøre det mulig for `PluginManager` å arve fra en annen klasse. Eksempel på det kan være om man ønsker å bruke en *manager*-implementasjon, som gjør det enklere å håndtere plugins. Det går ikke når man har en abstrakt klasse, siden Java ikke tillater multippel arv.

Man burde slippe å definere plugins i entitetsinstansen. Entiteter har alltid de samme feltene som entitetsinstansen. Det burde være mulig å la DPG legge til felter som mangler i entitetsinstansene ved å sjekke feltene i entitetene.

Plugins i DPG har ingen *livssyklus* (eng. *life-cycle*). Livssyklus er vanlig funskjonalitet i en plugin-arkitekturer. Eksempel på et livssyklusrammeverk for plugins er OSGi [3]. Bruk av livssykluser vil gjøre det mulig å starte, stoppe og avinstallere plugins når vertapplikasjonen kjører. Dette ville vært nyttig i DPG. Eksempel på det er om DPG avslutter når en plugin kjører, så blir ikke pluginen skikkelig avsluttet. Bruker en plugin en filstrøm så vil ikke den bli lukket om DPG blir avsluttet.

Dette er observasjoner som ble oppdaget når kandidaten undersøkte mulighetene til plugins-arkitekturen. Dette er ikke noe kandidaten vil rette opp, fordi det påvirker ikke målet med oppgaven. Svakhetene og mulige løsninger vil imidlertid bli lagt frem under forslag til videre arbeid i kap 8.

3.2 Konklusjon

Det er mulig å lage en ny type plugin, men det vil skape ekstra arbeid uten god grunn. Det vil også bli et spesialtilfelle å lage plugins som returnerer et utsnitt. Løsningen med plugins som returnerer et JDOM-element er ganske fleksibel. Man kan da definere hva man vil legge til i presentasjonen med XML. Dette er en løsning som vil fungere med XForms-implementasjoner. Kandidaten velger derfor å bruk den eksisterende arkitekturen.

4

Xforms designer og rendrer

Dette kapittelet er skrevet av Morten Høiland og Øystein Lund Rolland i felleskap. Kapittelet er å finne i begges masteroppgave. Dette er blitt gjort fordi begge parter er interessert i å få XForms-støtte i DPG. Morten Høiland jobber med å rendre XForms og samle inn data. Øystein Lund Rolland [67] jobber med å integrere en designer i DPG, som gjør det enklere å lage XForms-skjemaer. Dette kapittelet vil ta for seg det som er felles for begge, nemlig begrunnelsen for valg av XForms som webskjema, og hvordan rendreren og designeren skal fungere sammen.

4.1 HTML-skjemaer

I 1993 ble skjemaer inkludert i HTML-*forslaget* (eng. *recommandation*) med navnet HTML+ [44]. Siden da har HTML-skjemaer ofte blitt brukt i webapplikasjoner for netthandel fra midten av 90-årene og senere til bruk i Web 2.0. Den sosiale bølgen som omdefinerte bruken av internett i begynnelsen av millenniumet med nettsteder som Facebook, Twitter, Last.fm og MySpace. Web 2.0 bygger på at brukeren står for mye av innholdet, og at arkitektene bak nettstedet tilrettelegger for at brukeren kan dele dette med andre. Interaksjon med brukeren er blitt veldig viktig for de fleste websider uavhengig av bruksområde. For å oppnå interaksjon trenger man å ta inndata fra brukeren, slik webskjemaer tilbyr.

HTML-skjemaer er en enkel måte å samle inn data fra brukeren uten at det avhenger av hvilken nettleser og operativsystem man bruker, men HTML-skjemaer har også sine begrensninger:

- HTML-skjema baserer seg på navn-verdi par som gir en flat datastruktur som ikke alltid er tilstrekkelig for komplekse skjemaer.
- Man må bruke JavaScript om man ønsker å validere, gjøre kalkulasjoner og vise feilmeldinger på klientsiden. Dette kan være vanskelig siden de ulike nettleserene tolker JavaScript på sin egen måte.
- Å instansiere ferdig utfylte HTML-skjemaer er noe man ofte ønsker for å tilby brukeren å endre tidligere innsendt data. Da må man ofte hente ut data fra forskjellige databaser og sy disse sammen i et skjema. For å fylle et skjema må hver bit finnes og settes sammen. Dette er en krevende prosess for serveren, og kan forårsake ytelsesproblemer.
- I tilfeller der dataene i HTML-skjemaer må igjennom mange instanser før prosessen er ferdig, er vanskelig å oppnå uten å måtte skrive tilpasset kode som behandler og opprettholder dataen mellom fasene. Eksempel på dette tilfellet kan være om man søker dagpenger på NAV så må skjemaet gjennom en saksbehandler for godkjenning osv. Dataene blir da tolket ulikt for hvert steg i prossesen.

De nevnte begrensningen fører til undøvendig arbeid for web-utviklere. Det vil bli programmert mye duplikat kode i forskjellige programmeringsspråk om man ønsker klientsidevalidering og serversidevalidering, og dokumentene som inneholder disse skriptene kan bli store og vanskelige å håndtere. Mangel av støtte for komplekse strukturer kan føre til komplisert spesialtilpasset kode for å rette opp begrensningene. Initialisering av data i skjemaer er krevende for serveren for den må sette sammen informasjon som er spredt i systemer i dokumentet. For å rette opp i disse begrensningene og tilby mer avansert funksjonalitet, startet HTML-arbeidsgruppen en undergruppe som skulle lage en ny spesifikasjon for webskjemaer. Spesifikasjonen fikk navnet XForms, og er ikke bakoverkompitabel med HTML-skjema.

4.2 HTML5

Begrensningene som har blitt gjennomgått til nå, har tilhørt HTML 4.01 [17] og XHTML 1.1 [69], som er de gjeldene anbefalingene til W3C [82]. Det finnes forøvrig en ny spesifikasjon som fortsatt er under utvikling, der mye funksjonalitet er tilgjengelig for bruk i dag. Den får navnet HTML5, og er planlagt til å bli en WC3-anbefaling i

2014. Mange av det mest brukte nettleserene støtter en rekke deler av HTML5 i de nyeste versjonene [62].

4.2.1 HTML5-skjemaer

HTML5-skjemaer får 13 nye inndata-felttyper [87]. De lar deg definere typer som e-post, URL, tall, dato, rekkevidde og farge. Figur 4.1 viser eksempel på inndatafelt rendret av Opera 11.

Data for en rekke typer, som tall, e-post og URL blir validert. På typen tall kan man sette maksimums- og minimumsverdier og at typen må være tall, e-post feltet må ha en gyldig adresse med "@" (alfakrøll) og "." (punktum) og URL må bestå av en gyldig webadresse etter standarden. Skriver man inn verdier som ikke stemmer overens med det skjemaet forventer, skal ikke skjemaet bli sendt inn og det skal dukke opp en feilmelding. Hvordan man håndterer de forskjellige typene er opp til implementasjonen av spesifikasjonen i de forskjellige nettleserene. Per dags dato er det bare Firefox 4 som har implementert alle inndata-typene. Fordelen med spesifikasjonene er at man slipper en del JavaScript med disse enkle valideringene. Det er også en fordel for håndholdte apparater med berørelseskjerm som har lite tastaturplass, for da kan man tilpasse hvilke taster som skal vises som "@" og ".com" om feltet er av typen e-post. Man kan også markere felter som er påkrevd, slik at de må være fylt ut før man kan sende inn skjemaet.

4.3 XForms oversikt

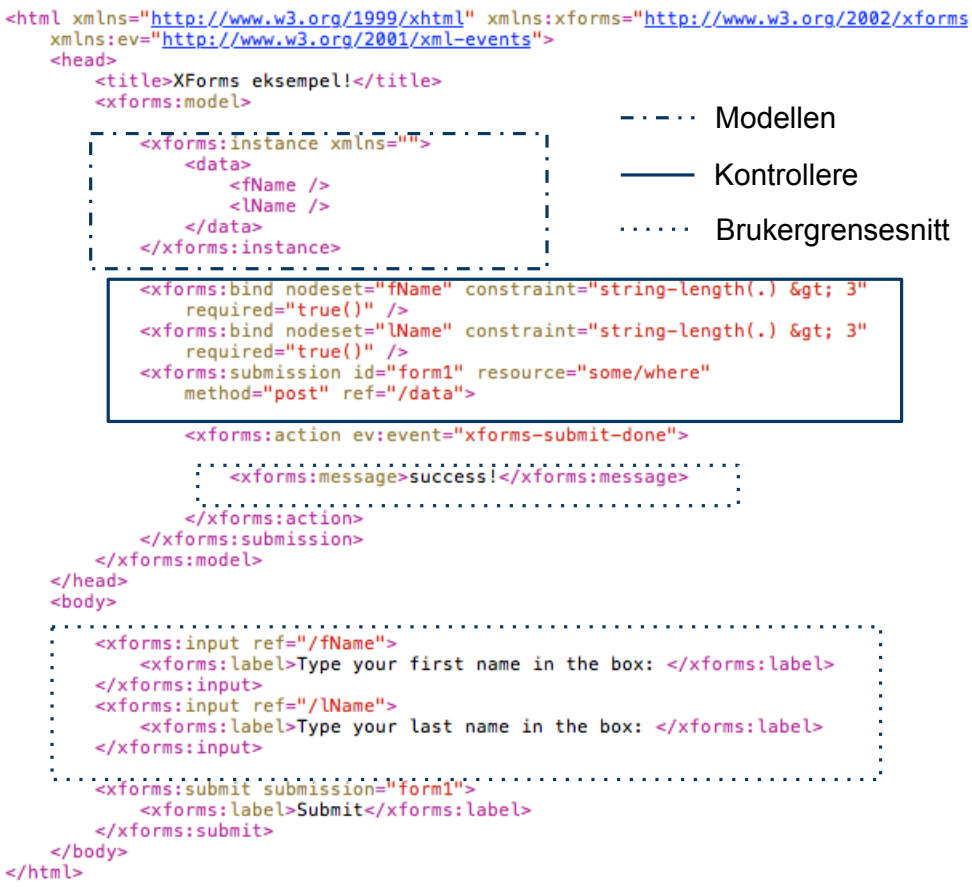
XForms er neste generasjons webskjemaer, og har vært en W3C-anbefaling siden 14. oktober, 2003. Den nyeste versjonen XForms 1.1 ble en anbefaling 20. oktober, 2009. XForms baserer seg på XML og er ikke i seg selv frittstående men er ment for å bli integrert med andre markeringsspråk som XHTML og SVG. XForms bygger på en MVC-arkitektur, slik at man får separert innholdet, modellen og presentasjonen. Et XForms-basert skjema samler XML-data og behandler dem. Det som ligger i bunnen av et XForms-skjema er instanser definert i XML, vist i den svarte boksen i figur 4.2.

XForms er bygd med tanke på mangler i HTML-skjema standarden. Her er noen av de viktigste kravene:

- XForms skal bruke XML for de opprinnelige dataene og for dataene som blir sendt fra klient til server.

Figur 4.1: HTML5-skjema eksempler

- Forskjellen mellom et XForms-skjema som er utfylt og et som ikke er det skal være minimal.
- Det skal være en klar oppdeling av modellen, kontrollere og brukergrensesnittet.
- Språket skal støtte kalkulasjoner på inndata, validering og feilmeldinger av data. Det vil kraftig redusere nødvendigheten av skripting.
- Xforms skal støtte de datatypene som allerede finnes i HTML-skjemaer.
- Det skal være enkelt la flere brukere benytte skjemaet i en arbeidsprosses.
- Språket skal være uavhengig av plattformer.
- Xforms skal kunne integreres sømløst med andre XML-teknologier.




```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:xforms="http://www.w3.org/2002/xforms
    xmlns:ev="http://www.w3.org/2001/xml-events">
    <head>
        <title>XForms eksempel!</title>
        <xforms:model>
            <xforms:instance xmlns="">
                <data>
                    <fName />
                    <lName />
                </data>
            </xforms:instance>
            <xforms:bind nodeset="fName" constraint="string-length(.) &gt; 3"
                required="true()" />
            <xforms:bind nodeset="lName" constraint="string-length(.) &gt; 3"
                required="true()" />
            <xforms:submission id="form1" resource="some/where"
                method="post" ref="/data">
                <xforms:action ev:event="xforms-submit-done">
                    <xforms:message>success!</xforms:message>
                </xforms:action>
            </xforms:submission>
        </xforms:model>
    </head>
    <body>
        <xforms:input ref="/fName">
            <xforms:label>Type your first name in the box: </xforms:label>
        </xforms:input>
        <xforms:input ref="/lName">
            <xforms:label>Type your last name in the box: </xforms:label>
        </xforms:input>
        <xforms:submit submission="form1">
            <xforms:label>Submit</xforms:label>
        </xforms:submit>
    </body>
</html>
```

Figur 4.2: Eksempel på XForms-skjema

4.4 XForms-arkitekturen

4.4.1 Modellen

Modellen er en XML-representasjon av skjemaet og definerer hvilke data som kreves fra brukeren. Den viktigste delen er vist i figur 4.2 i boksen med striplet linje med prikk, det er selve instansdataene som blir fylt inn av brukeren og sendt inn til serveren. Instansdataene kan inneholde forhåndsdefinerte data, noe som er nyttig hvis intensjonen er å redigere et allerede utfylt skjema. Man har mulighet til å separere modellen fra instansdata ved hjelp av `bind`-elementet, illustrert i boksen med hele linjer i figur 4.2. Da får man et naturlig skille mellom data og modell. I `bind`-elementene kan man også spesifisere datatype og andre begrensninger for hvert inndata-felt ved hjelp av XPath [84] som bidrar til validering av inndata på

klientsiden. Det er også mulig å utføre kalkulasjoner med XPath. Gir brukeren inndata som ikke tilfredstiller kravene stilt i `model`-elementet, vil klienten gi en direkte feilmelding før skjemaet sendes inn. Dermed unngår man at validatoren på serversiden finner feilen, noe som både sparer brukeren for tid og serveren for unødvendig datatrafikk.

I `model`-elementet spesifiseres også hvor utfylt skjema skal sendes, hvilket navn det skal ha og hvordan det skal sendes, med hjelp av `submission`-elementet. Man har også mulighet til å velge ut spesifikke data man ønsker å sende, noe som gjør det mulig å ha flere innsendingsknapper på samme skjema.

4.4.2 Brukergrensesnittet

Brukergrenssnittet har egne markeringer som består av abstrakte kontrollere som kan koble sammen data fra instans dataene og kan med det lage rike brukergrensesnitt. Det er også mulig å lage egendefinerte hendelser ved hjelp av `action`-elementet. Dette fungerer som *utløser* (eng. *trigger*), og utfører ønsket handling som for eksempel en tilbakemelding, ved spesifisert tilstand. Eksempel på dette kan man se i boksen med striplet linje i figur 4.2. Hovedtanken bak brukergrensesnittet er å definere hvordan data skal mottas, ikke hvordan det vil se ut på forskjellige plattformer og apparater.

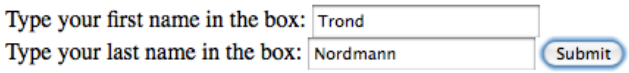
4.4.3 Kontrollerene

Kontrollerene er det som kobler sammen brukergrensesnittet mot instansdataene og tar hånd om manipuleringen av dataene. De tar for seg sending av data, utregninger og validering. I figur 4.2 er noen av kontrollerene vist i boksen med prikkete linje.

4.4.4 Eksempel

Figur 4.2 er et enkelt eksempel på hvordan XForms er brukt. Det er brukt i et XHTML-dokument, hvor `model`-elementet er definert i `head`. Elementet `Model` har også `submission`-elementet som setter hvordan informasjonen skal bli sendt og hvilke deler av informasjonen som blir sendt. I `body`-elementet er brukergrensesnittet deklarert og koblinger mellom inndata elementene mot instansdataene.

Figur 4.3 viser rendret skjema fra koden i figur 4.2 i en nettleser. Det er mulig å forandre designet på XForms-skjemaer med CSS. Trykker man på `Submit` sender man inn dataene spesifisert i `submission` elementet. Det som blir sendt da er



Figur 4.3: XForms rendret

instansdataen med inndataen fra brukeren i XML.

Listing 4.1: Eksempel på instans data sendt fra XForms-skjema

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <data xmlns="" xmlns:xforms="http://www.w3.org/2002/xforms" xmlns:ev="↩
      http://www.w3.org/2001/xml-events">
3     <fName>Trond</fName>
4     <lName>Nordmann</lName>
5 </data>
```

Hvis man sammenligner forskjellen mellom hvordan dataene er definert i figur 4.2 og listing 4.1, er det bare inndataene fra brukeren som skiller. Det er derfor lett å kunne laste inn allerede initialiserte data fra XML-dokumenter inn i XForms-skjemaet, så når brukeren skal redigere eller fylle inn data, er dataene på plass uten at serveren må gjøre noe prossesering for å sette sammen data.

4.5 HTML5 vs. XForms

HTML5 [62] har kommet med en god del forbedringer siden HTML 4.01 [17] og XHTML 1.1 [69], som valideringer og muligheten for påkrevd inndata. Mange av forbedringene vil gjøre det enklere å bruke HTML5-skjemaer på mobilen. Men selve datamodellen er den samme og det er ikke mulig å validere på klientsiden annet enn hva som er spesifisert, slik at bruken av JavaScript vil fremdeles være nødvendig hvis man skal gi brukeren funksjonaliteten som blir forventet. For eksempel validering av data mens man skriver inn dataene. Noe HTML5 har, men som ikke støttes av XForms-spesifikasjonen [86], er muligheten for å velge farge ut i fra et fargekart.

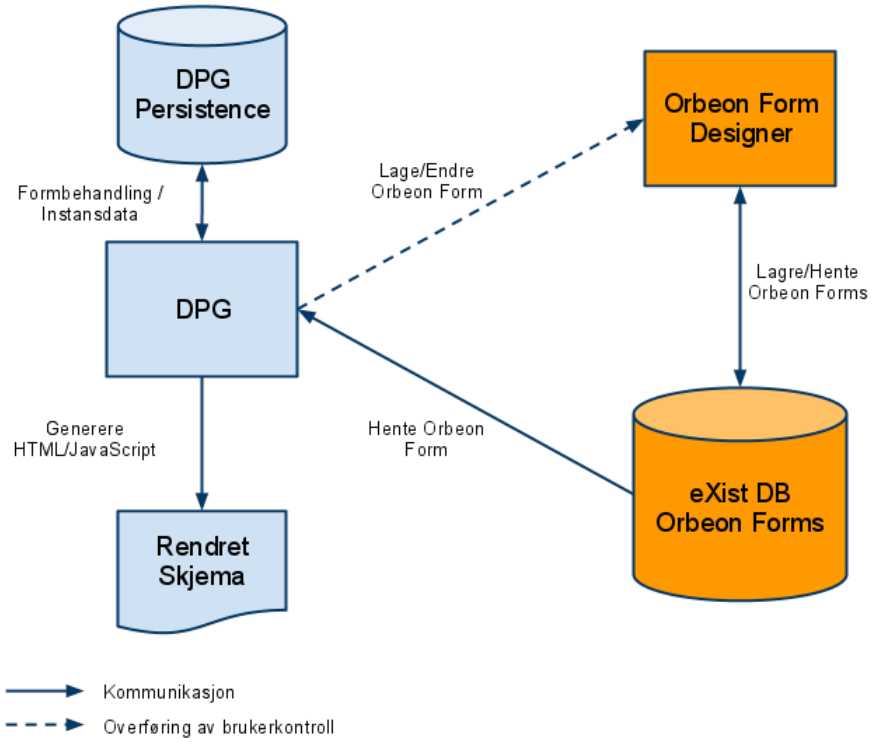
XForms er overlegen når det kommer til muligheten for å validere data med bruk av XPath-uttrykk, men det er naturligvis vanskeligere enn å bare bruke markeringer. Datamodellen til XForms er også bedre siden HTML5 bruker samme modell som HTML 4.01. Man står også friere til å gi gode feilmeldinger med XForms, med HTML5 er nettleseren som står for dette. HTML5 er bakoverkompatibelt, slik at hvis man bruker noen av de nye typene så vil de automatisk falle tilbake til vanlige tekstfelter. Men det vil ta lang tid før alle oppdaterer nettleserene sine til de nyeste

versjonene, og nettleserene har ikke implementert hele HTML5 skjemaspesifikasjonen enda. Så man vil i flere år fremover være avhengig av metoder i JavaScript som man kan falle tilbake på om man ønsker klientside-validering.

Det er fornuftig at HTML5 kommer med nye funksjoner for å minske bruken av JavaScript, men det er ikke de store muligheten til å lage egne regler enda. Det er derfor klart mye bedre med XForms som webskjema for mer kompliserte oppgaver enn å ta inn enkle inndata som navn og e-post. HTML5 har arvet det meste av forgjengerene som er kritikkverdig fra HTML 4.01, bortsett fra validering for noen typer.

På bakgrunn av foregående informasjon, har kandidatene konkludert med at XForms pr dags dato er mer fleksibelt og enklere å bruke enn HTML5, og vil derfor være valgt teknologi for utviklingen av inndatasamling i DPG.

4.6 Overrordnet struktur



Figur 4.4: Overrordnet struktur - Hvordan skjema blir laget og presentert

Figur 4.4 illustrerer overrordnet struktur og viser hvordan designer, skjema-lager og rendrer jobber sammen for å danne et XForms-system for design, presentasjon og innsamling av data. I denne delen av kapittelet forklares komponentene og hvordan de kommuniserer, hvordan skjema går fra utvikling til presentasjon.

4.6.1 Orbeon Form Designer

Orbeon [60] er et system for design, strukturering og presentasjon av XForms, men i DPG er det bare designerdelen som benyttes. Systemet er løst integrert, hvilket betyr at Orbeon kjøres som en separat webapplikasjon. En med *plublisher*-rollen som ønsker å opprette eller redigere skjema vil bli sendt til designervektøyet via en hypertekstlenke. Hvis et spesifikt skjema skal endres på, vil brukeren få mulighet til dette ved å bli sendt til Orbeon sin formdesigner. Dette er illustrert med striplet pil, som betyr at brukeren bytter system. I motsetning til rette streker som indikerer kommunikasjonen mellom komponentene. Mer om dette systemet er beskrevet i Øystein Lund Rolland sin masteroppgave [67].

4.6.2 eXist Database

XForms-skjemaer som Orbeon genererer, blir lagret i en eXist-database [48]. I Orbeon blir XForms-skjema kalt *Orbeon forms*, og grunnen til dette er at de ikke bare inneholder XForms-definisjonen, men også en del XML som brukes i Orbeon sin rendrer for å strukturere det visuelle uttrykket skjemaene skal ha. De benytter også sin egen ressurshåndtering. Skjemaene blir derfor referert til som Orbeon Forms før de blir hentet i DPG. Dokumentene aksesseres gjennom en felles API som baserer seg på XMLRCP-spørringer [90].

4.6.3 DPG

I DPG blir Orbeon Forms hentet på forespørsel fra XForms-pluginen. Skjemaet går gjennom en filtreringsprosess for å fjerne alle ugjenkjennelige XML elementer og andre modifikasjoner slik at XForms-pluginen kan lese innholdet og generere JavaScript og HTML som nettleseren kan forstå. Skjema som pluginen genereres velges ut ifra en ID og en egen skjemabehandler som referere til riktig skjema spesifisert av en publisher. XFormspluginen har ansvaret for å samle inn instansdata og lagre disse i DPG sitt persistenslag. Mer om hvordan DPG presenterer skjema og samler inn data kan leses i kapittel 6. Informasjon om hvordan XForms-skjemaer filtreres og behandles i DPG kan leses i Øystein Lund Rolland sin masteroppgave [67].

5

Støtte for XForms i DPG

5.1 Problemer relatert til bruk av XForms

Ingen av de mest brukte nettleserene i dag har integrert XForms-støtte. Det vil si at de fleste brukerene ikke vil kunne bruke applikasjoner som tar i bruk XForms, men det finnes en rekke løsninger på dette problemet. De er delt opp i følgende kategorier [42]:

1. Nettlesere som støtter XForms.
2. Utvidelser til nettlesere.
3. Klientsideløsninger.
4. Serversideløsninger

5.2 Forskjellige typer av XForms-implementasjoner

5.2.1 Nettlesere som støtter XForms

Det finnes nettlesere som har XForms-støtte integrert. Eksempeler på slike nettlesere er PicoForms [61], Mozquito DENG [13] og X-Smiles [39]. Med denne løsningen vil

man kunne bruke XForms i sin originale form rett i nettleseren. Brukeren må da laste ned en ekstra nettleser, som de sannsynglivis bare vil bruke til applikasjoner som bruker XForms. Det er fordi 90% av alle nettleserbrukere bruker Internet Explorer, Firefox, Safari eller Chrome [89]. Disse nettleserene er mer modne og har mer funksjonalitet enn hva en nettleser som er utviklet med et formål om å støtte XML-teknologier. Dette er målet til alle disse nettleserene som støtter XForms. Hvordan disse nettleserene støtter XForms blir vist i figur 5.1 og forklart i neste avsnitt.

5.2.2 Utvidelser til nettlesere

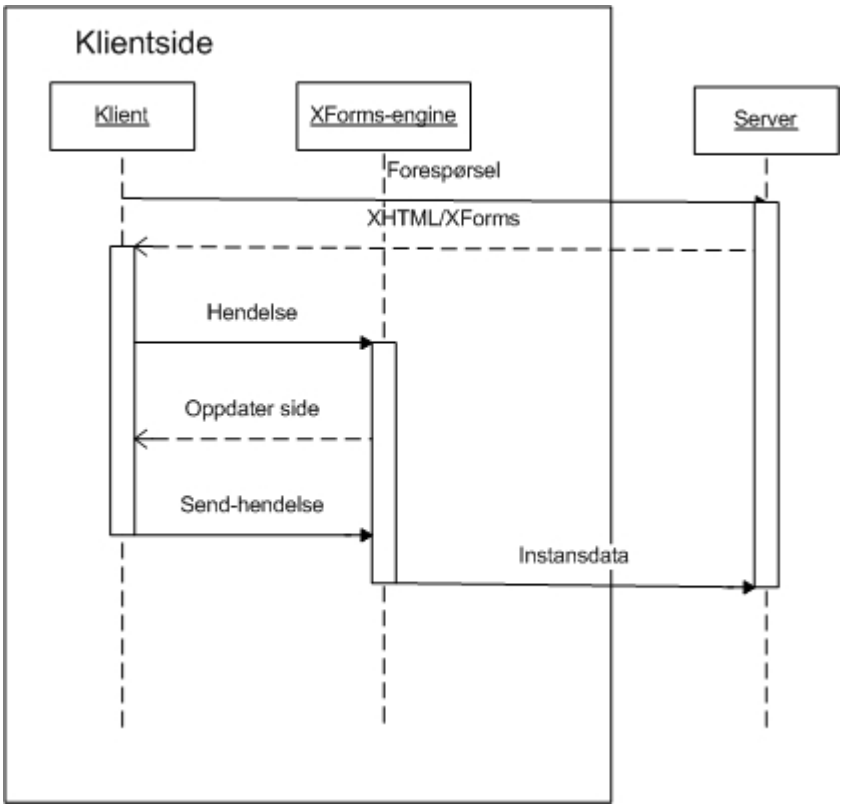
Det finnes utvidelser til forskjellige nettlesere som gjør det mulig å gi støtte for XForms i Internet Explorer og i Mozilla Firefox. For Internet Explorer kan man bruke MozzIE [51] og Formsplayer [11]. Firefox har utvidelsen Mozilla XForms Project [50].

Både utvidelser for nettlesere og nettlesere som støtter XForms har lignende modell på hvordan XForms fungerer. I figur 5.1 ser man hvordan klienten, som er selve nettleseren, bruker en `XForms-engine` til å håndtere forskjellige *hendelser* (eng.*events*). Brukergrensesnittet blir så oppdatert etter som tilstanden forandrer seg i selve XForms-skjemaet. Når brukeren har fylt inn dataene i skjemaet og de er validerte, kan brukeren sende det inn. Skjemaet sender så inn sine instansdata som XML til serveren.

5.2.3 Klientsideløsninger

Alle de mest brukte nettleserene har støtte for JavaScript [21], derfor er det mulig å lage en fullstendig XForms-implementasjon ved hjelp av bare JavaScript. Det gjør det mulig å lage en klientsideimplementasjon, som ikke er nettleseravhengig eksempeler på det er FormFaces [40] og Ubiquity XForms [45]. XSLTForms [16] bruker en annen løsning hvor man gjør XForms-skjemaet om til JavaScript ved hjelp XSLT-transformasjoner. Det gjør at alle manipulasjonene på XForms-instansen blir gjort av JavaScript. Transformasjonen gjøres av nettleseren.

Figur 5.2 viser hvordan `AJAX-engine` i nettleseren som håndterer XForms-skjemaet og at alle hendelser blir gjort i JavaScript. JavaScriptet blir lastet ned ilag med XHTLM og XForms-dokumentet, som gjør at første nedlasting er stor og vil ta opp båndbredde. Brukeren fyller inn data og dataene blir validert av JavaScript. Når brukeren har fylt inn dataene og de er validerte, sendes dataene inn som XML til serveren.

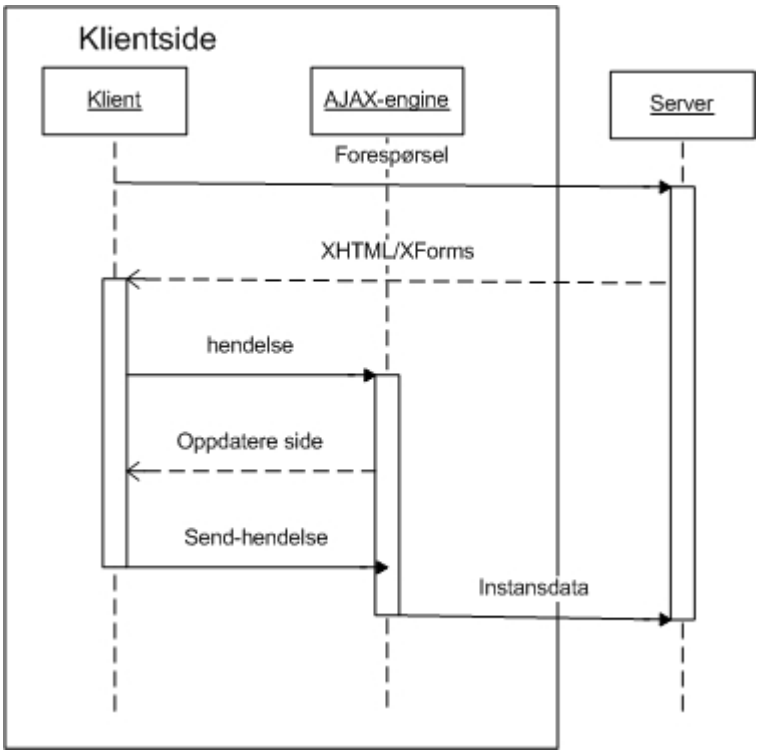


Figur 5.1: Hvordan utvidelser til nettlesere og nettlesere som støtter XForms fungerer

5.2.4 Serverside løsninger

Det finnes mange serversideløsninger som tilbyr støtte for XForms uten å måtte installere ekstra programvare. Den vanligste løsningen er å transformere XForms-skjemaet om til HTML, eller en blanding av HTML og JavaScript, også kalt *Dynamic HTML*, for så å levere den til brukeren. XForms-skjemaet ligger på serveren og blir oppdatert av JavaScript. Eksempler på slike løsninger er BetterForm [36], Orbeon [60], OpenXdata [55] og AJAXForms [1]. BetterForm og Orbeon bruker XSLT til å transformere XForms-skjemaet til HTML og JavaScript. OpenXdata og AJAXForms lager ren JavaScript som blir sendt til nettleseren, OpenXdata benytter seg av GWT [27].

Figur 5.3 viser hvordan serversidetransformasjoner skiller seg ut fra de andre løsningene ved at man får inn HTML og JavaScript i stedet for HTML og XForms. Håndteringen av hendelser må helt tilbake til serveren for å oppdatere XForms-skjemaet, fordi all validering blir gjort på serveren. Brukeren vil ikke legge merke til



Figur 5.2: Hvordan klientsideløsninger fungerer

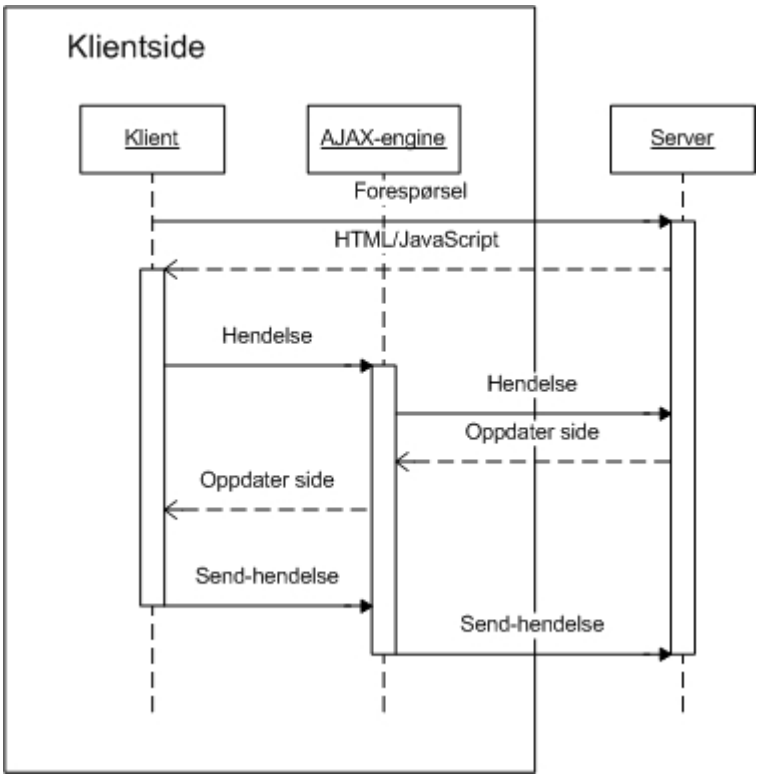
valideringsprossesen, fordi det blir brukt JavaScript til å oppdatere brukergrensesnittet dynamisk. Når brukeren trykker på send-knappen og alle dataene er validerte, vil serveren få beskjed om å sende instansdataene til destinasjonen som er definert i XForms-skjemaet.

5.2.5 Sammenligning av implementasjonstyper

Implementasjonsløsningene har ulike fordeler og ulemper. Neste seksjon vil gjøre rede for disse, og legge grunnlag for å gjøre et kvalifisert valg blant løsningene i henhold til funksjonaliteten som er ønskelig i DPG.

5.2.6 Vurdering av implementasjonstyper

Tabellen 5.1 er laget ut i fra Lain [42] og viser hva de forskjellige implementasjonene er velegnet for.



Figur 5.3: Hvordan serversidetransformasjoner fungerer

Det er andre kriterier som kunne blitt vurdert, men som kandidaten ikke valgte å ta med fordi det ikke er relevant for DPG sitt bruksområdet. Det er blant annet å kunne bruke applikasjonen uten nettverksforbindelse, bruk av båndbredde med XForms og mengde data ved første nedlasting. Disse kriteriene vil være mer aktuelle om man skulle bruke løsningen over mobilnettverket eller nettverk med lite båndbredde.

I tabell 5.1 ser man at svakheten med å bruke en spesiallaget nettleser og utvidelser til nettlesere er at brukeren må laste ned disse for å kunne ta i bruk applikasjonen. Dette er negativt fordi brukere er blitt vant til at alt fungerer med det oppsettet de allerede har. Det er her styrken til serversidetransformasjoner og AJAX-implementasjoner viser seg. Løsningen fungerer med nesten hvilke som helst nettleser og krever ikke at brukeren trenger å gjøre noe. Velger man å bruke nettlesere og utvidelser som støtter XForms, velger man å stole på flere komponenter, og ikke bare sin egen applikasjon.

Serversidetransformasjoner og AJAX-implementasjoner sin svakhet er at det kan oppstå forsinkelser på brukergrensesnittet, fordi oppdateringene av XForms-skjemaet utføres av JavaScript. Med serversidetransformasjoner må man helt tilbake til serveren for å validere og oppdatere brukergrensesnittet. Prossesen må gjøres for hver

Tabell 5.1: Oversikt over typer av XForms-implementasjoner.
"-" betyr at kriterien påvirker negativt, "-/+" betyr det kan være forskjellige for implementasjoner og "+" betyr at det ikke påvirker noe.

Implementasjoner/ Kriterier	Nettleser støtte	Nettleser utvidelse	AJAX-implementasjon	Serverside-transformasjon
Klientside installering	-	-	+	+
Nettleser uavhengig	-	-	+	+
JavaScript uavhengig	+	+	-/+	-/+
Serverside teknologiavhengig	+	+	+	-
Forsinkelse av UI	+	+	-/+	-
Usikre komponenter	-	-	+	+

endring. Med serversideløsningen må man også forholde seg til en teknologi som tilbyr støtte for XForms på serversiden. Dette kan skape problemer om man ikke støtter den teknologien. AJAX-implementasjoner bygger på ett spesifikt JavaScript-bibliotek, men Ubiquity XForms bygger på et markeringsspråk som gjør det mulig å bruke det AJAX-biblioteket man ønsker. Mange implementasjoner av serverside-transformasjoner bruker et JavaScript-bibliotek for å støtte XForms-skjema, for eksempel BetterForm som bruker DOJO [18] og OpenXdata som tar i bruk GWT [27].

5.2.7 Valg av implementasjonstype

Kriteriet som kandidaten legger mest vekt på er at brukeren av applikasjonen ikke må gjøre noe for å få funksjonaliteten man ønsker. Da er det bedre at arbeidet blir gjort på serversiden. Derfor er nettlesere og utvidelser til nettlesere uaktuelle. Kandidaten vil derfor undersøke nærmere bibliotekene som er i kategoriene AJAX-implementasjoner og serversidetransformasjoner.

5.3 Vurdering av XForms-biblioteker

AJAX-implementasjoner som kandidaten har funnet aktuelle er XSLTForms og Ubiquity XForms. Serversidetransformasjonerimplementasjonene som blir vurderte er BetterForm, Orbeon og OpenXdata. Disse er blitt valgt ut på grunnlag av følgende kriterier:

- Det pågår aktiv utvikling av biblioteket.
- Biblioteket er åpen kildekode, som er lisensiert under *GNU Lesser General Public License* (LGPL) [64] eller *Apache License, Version 2.0* [23].
- Biblioteket blir brukt i flere andre applikasjoner.
- Det skal være mulig å bruke biblioteket med plugin-arkitekturen til DPG.

5.3.1 Ubiquity XForms

Ubiquity XForms [45] er laget for at man skal kunne ha støtte for XForms og samtidig velge fritt blant AJAX-biblioteker man vil bruke. Biblioteket består av en kjerne som er et markeringsspråk kalt Mixins [9], som bygger på AJAX-biblioteker. Man vil da kunne skrive XForms-skjemaer som blir omgjort til Mixins som kan brukes av det forskjellige AJAX-bibliotekene. Ubiquity XForms er laget i AJAX-biblioteker som Yahoo! User Interface (YUI) Library [92] og DOJO [18]. Man vil kunne velge flere AJAX-biblioteker senere når de er ferdig utviklet.

Fordeler:

- Det er Lett å intergrere i DPG. Man trenger bare legge ved en lenke til et JavaScript, som vil rendre XForms-skjemaet.
- Kan bruke forskjellige AJAX-biblioteker.
- Tilbyr validering og prosesshåndtering i klienten.

Ulemper:

- Har ikke støtte for hele XForms-spesifikasjonen.
- Biblioteket er enda under utvikling til å bli en stabil versjon.

5.3.2 BetterForm

BetterForm [36] er utviklet for å kunne rendre XForms-skjema og samle inn data i nettleseren. Biblioteket kan brukes med alle de mest brukte nettleserene.

BetterForm er utviklet i Java og bruker XSLT-transformasjoner til å gjøre om XForms-skjemaer til HTML og JavaScript. DOJO [18] er AJAX-biblioteket som blir brukt til å håndtere JavaScript-delen, som har med å presentere XForms-skjemaet til de forskjellige nettleserene. DOJO kommuniserer så med selve XForms-skjemaet som ligger på serveren, og kjører i Java.

Fordeler:

- Det er utviklet i Java.
- Det brukes et kjent AJAX-bibliotek til å håndtere alle de forskjellige nettleserene.
- Det har veldig god støtte for XForms-spesifikasjonen.

Ulemper:

- Biblioteket er tidkrevende å integrere. Det er mange teknologier som må integreres i DPG.
- Det bruker mye båndbredde, med JavaScript-kall som går mellom serveren og klienten, for å oppdatere XForms-skjemaet.
- Biblioteket bruker DOJO, men det hadde vært bedre om JQuery [38] brukt siden dette er allerede integrert i DPG [70].

5.3.3 Orbeon

Orbeon [60] har både en XForms-designer og en XForms-rendrer, begge må integreres for å kunne bruks. Det finnes også en profesjonell versjon som gir deg noen flere funksjoner og er bedre optimalisert, men gratis-versjonen er god nok til kandidatens mål.

Orbeon er skrevet i Java og transformerer XHTML/Xforms-dokumentet om til JavaScript og HTML ved hjelp av XML PipeLine Language(XPL) [19], som blir sendt til klienten. JavaScript-et oppdaterer XForms-skjemaet på serveren.

Fordeler:

- Det er tviklet i Java.
- Biblioteket tilbyr en XForms-designer.
- Det fungerer i de mest brukte nettleserene.

Ulemper:

- Biblioteket er tidkrevende å integrere. Mange teknologier som må integrerest.
- Orbeon tar opp mye båndbredde, med JavaScript-kall som går mellom server og klienten.
- Orbeon bruker selvlaget JavaScript, som ikke lett kan skiftes ut.
- Biblioteket bruker XPL som er noe Orbeon har utviklet selv. Det kan være vanskelig å bruke i DPG siden det er lite støtte fra andre biblioteker.

5.3.4 XSLTForms

XSLTForms [16] transformerer XHTML/XForms til HTML og JavaScript, ved hjelp av XSLT i nettleseren. Man trenger ikke bruke en server, noe som betyr at klienten og server er uavhengige av hverandre. Den eneste kommunikasjonen mellom server og klienten som blir gjort er av instansdata i XML, som JavaScript tar hånd om.

Fordeler:

- Uavhengig av server-teknologier og nettlesere.
- Bilblioteket tilbyr validering og prosesshåndtering i klienten.
- Det benytter kjente teknologier som XSLT og JavaScript.
- XSLTForms er lett å integrere i DPG. Man legger bare med transformasjonen med i dokumentet som blir sendt til nettleseren.

Ulemper:

- XSLTForms støtter ikke hele XForms-spesifikasjonen.
- Det er enda under utvikling til å bli en stabil versjon.

5.3.5 OpenXdata

OpenXdata har også en XForms-designer i tillegg til en rendrer. OpenXdata bruker GWT [27] til å lage brukergrensesnittet, som kommuniserer med serveren. GWT bruker en Java API og widgets som lar utviklere skrive koden i Java og krysskompilere den til optimalisert JavaScript som vil fungere i de mest brukte nettleserene.

Fordeler:

- OpenXdata har en XForms-designer.
- Biblioteket er utviklet i Java.

Ulemper:

- OpenXdata er tidkrevende å integrere i DPG. Mange teknologier må integreres, som for eksempel GWT.
- Demoen for XForms-rendreren er svært lite stabil.

5.4 Valg av XForms-implementasjon

Ut i fra tabell 5.1 ser vi at serversidetransformasjoner er overlegne på en rekke områder. Ubiquity og XSLTForms har vist seg å være ganske uferdige, og har enda langt igjen før de er like modne som serversidetransformasjonimplementasjonene. De har også en del problemer med å støtter forskjellige nettlesere, som man kan se i figur 5.3. Der er serversideløsningene overlegne. Fordelen med å bruke AJAX-implementasjonene er at det er enkelt å integrere i DPG, fordi man legger bare ved de dokumentene som inneholder JavaScript-kode og håndtere transformasjonen på klienten.

Av serversidetransformasjonene er det Orbeon og BetterForm som kommer ut best. OpenXdata er spennende fordi det bygger på GWT, som virker lovende for JavaScript-applikasjoner. XForms-designeren til OpenXdata er bra, men selve XForms-rendreren er ikke god nok enda, det har hovedsaklig å gjøre med at de fokuserer på å bruke mobile klienter med Java Platform, Micro Edition (Java ME) [57]. De har ikke brukt mye ressurser på utvikle en applikasjon som rendrer XForms i nettlesere. OpenXdata har heller ikke kjørt applikasjonen sin igjennom spesifikasjonstesten [86] til XForms 1.1 slik at man ikke vet hvor mye av spesifikasjonen som er implementert.

Orbeon er en sterk kandidat til å bli valgt, den har også en XForms-designer som gjør den attraktiv. Biblioteket har noen mangler som ikke kan overses, som for eksempel dårlig støtte for spesifikasjonen og bruk av egenutviklede teknologier. De har god dokumentasjon på hvordan man bruker selve applikasjonen, men har ikke dokumentert hvordan man kan integrere Orbeon inn i andre applikasjoner. Den løsning di har dokumenter er å kjøre Orbeon parallelt med den applikasjonen man utvikler. Dette er noe som er ugunstig for DPG, siden man allerede har en rekke applikasjoner som kjører parallelt, som Webucator [43] og Solr [54]. Det er også veldig vanskelig å finne ut av eksakt hvor mye av XForms 1.1 spesifikasjonen den støtter i dag, fordi Orbeon har bare publisert resultatetene fra 2009.

Implementasjoner/ Kriterier	BetterForm	Orbeon	Ubiquity	XSLTForms	OpenXdata
Bra dokumentert	☑	☐	☐	☐	☐
Utviklet i Java	☑	☑	☐	☐	☑
Enkelt å integrere i DPG	☐	☐	☑	☑	☐
Høy utviklingsaktivitet	☑	☑	☐	☑	☑
Stabil versjon	☑	☑	☐	☐	☐
God demo	☑	☑	☐	☑	☐
Spesifikasjonstest	98%	63%	58%	48%	0%
Designer	☐	☑	☐	☐	☑

Tabell 5.2: Oversikt over XForms-implementasjoner og kriterier

Implementasjoner/ Nettlesere	BetterForm	Orbeon	Ubiquity	XSLTForms	OpenXdata
FireFox	☑	☑	☑	☑	☑
Opera	☑	☑	☐	☑	☑
Chrome	☑	☑	☐	☐	☑
IE	☑	☑	☑	☑	☑
Safari	☑	☑	☐	☐	☑

Tabell 5.3: Støtte for forskjellige nettlesere

BetterForm er den som kommer best ut av kriteriene i tabell 5.1. Det er mye bra med BetterForm, men det kandidaten verdsetter mest er en stabil versjon som kan integreres med andre Java applikasjoner og at den er godt dokumentert. Det som er negativt er at det ikke er noen XForms-designer, det hadde vært nyttig for de som vil lage XForms-skjemaer i DPG på en enkel måte. Det er negativt at man må bruke DOJO og ikke jQuery som AJAX-bibliotek, fordi jQuery er allerede integrert i DPG.

5.5 Konklusjon

Det hadde vært enklere å integrere en AJAX-implementasjon, men det har ikke kommet en stabil versjon som støtter nok av XForms spesifikasjonen til at kandidaten vurderer det som ett valg. Av serversidetransformasjonene er det BetterForm som er det beste valget, bortsett fra at den ikke har en designer slik som OpenXdata og Orbeon har. Det viser seg også at BetterForm støtter bedre XForms-spesifikasjonen slik den skal være. OpenXdata og Orbeon støtter de XForms-skjemaene som er laget

av designeren for applikasjonene, og lagt til noen konvensjoner som ikke er i Xforms-spesifikasjonen. Det gjør de mindre kompatible mot andre systemer som ønsker å utveksle XForms-skjemaer. Spesifikasjonstesten [86] fra W3C, som forteller hvor mye av selve spesifikasjonen som er implementert, bekrefter dette i tabell 5.1. På det området er BetterForm bedre enn de andre.

Valget blir å bruke BetterForm, for det støtter mest av XForms-spesifikasjonen og er bedre dokumentert på områdene om integrering i andre systemer. Det er også et viktig poeng at de to hovedmennene bak BetterForm, Windauer og Turner, har en lang bakgrunn med XForms og har bygget en annen implementasjon av XForms, Chiba [35]. De har brukt erfaringene fra Chiba til å utvikle BetterForm.

6

Implementasjon av XFormsPlugin

6.1 Integreringsmuligheter i DPG

I kapittel 3 ble det gjennomgått mulighetene for å integrere applikasjoner i DPG på en sømløs måte. Den metoden som ble valgt er å lage en plugin som implementerer `fieldplugin`-kontrakten, som gir muligheten til å legge til et JDOM-element til et utsnitt. Dette gjør det enkelt å legge til JavaScript og HTML slik at skjemaet kan bli rendret av DPG.

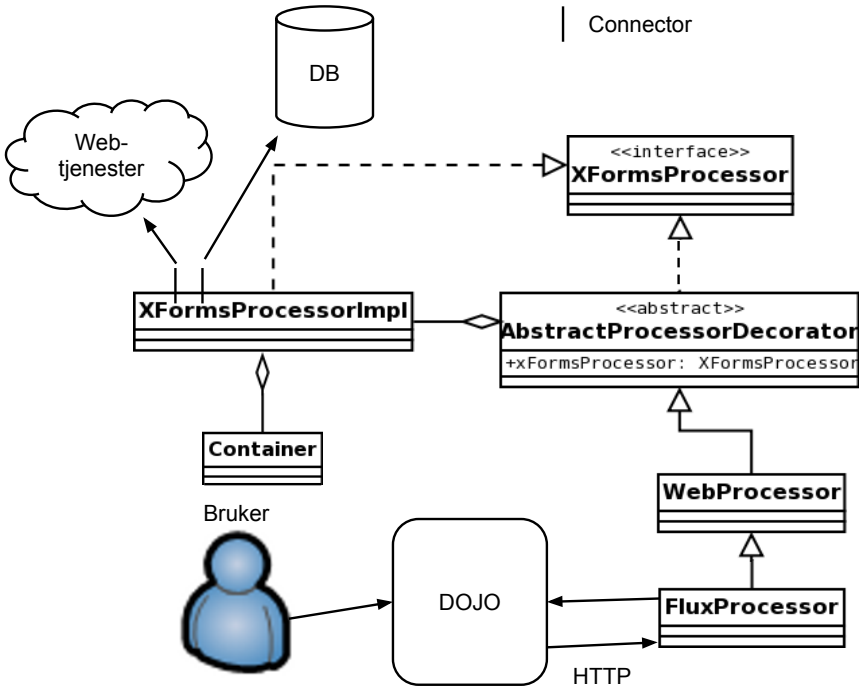
6.1.1 BetterForm

BetterForm er bygd opp av *prosessorere* som sammen utgjør en webapplikasjon, som kan rendre XForms-skjema og samle inn data. Arkitekturen er vist i figur 6.1 og har følgende komponenter:

- `Container` inneholder en DOM-presentasjon av selve XForms-skjemaet. Hvert element i skjemaet har hver sin unike ID.
- `XFormsProcessorImpl` er selve implementasjonen av XForms 1.1 [86]. Den tilbyr alle metodene for å håndtere et XForms-skjema. Prosessoren har en referanse til en `Container`, og oppdaterer XForms-skjemaet ved hjelp av DOM-

hendelser (eng. *events*).

- `WebProcessor` styrer kommunikasjonen med servlets og tar hånd om konfigurasjonen i en *web-container* [20].
- `FluxProcessor` er en implementasjon av *Direct Web Remoting*(DWR) [65]. DWR er et Remote Procedure Call (RPC)-bibliotek [46], som gjør det mulig å kalle JavaScript-metoder fra Java, og Java-metoder fra JavaScript.
- `Connector` gir muligheten til å koble til andre teknologier som ikke har med XForms å gjøre. Man kan utvide koblingene til å kommunisere med for eksempel en database eller forskjellige web-tjenester. BetterForm tilbyr allerede noen koblinger brukt til lagring og henting, blant annet med XMLRPC-protokollen [90] og HTTPS [2]. Kontrakten `Connector` blir brukt gjennom `XFormsProcessorImpl`, og kan implementeres for å gi mer funksjonalitet.
- `DOJO` er et AJAX-bibliotek. Det blir brukt til å rendre det transformerte XForm-skjemaet til brukeren og oppdatere brukergrensesnittet etter som endringer skjer i `Container`.



Figur 6.1: BetterForm arkitektur

BetterForm benytter seg av `decorator`-designmønsteret(eng.*Decorator pattern*) [47] som fungerer slik for BetterForm:

- Det er implementert en abstrakt klasse som utvider fra `XFormsProcessor` kalt `AbstractProcessorDecorator`. Dette er selve *decorator*-klassen.
- Klassen `AbstractProcessorDecorator` har en referanse til `XFormsProcessor`.
- I `AbstractProcessorDecorator`-klassen blir alle `XFormsProcessor`-metodene kalt på referansen til `XFormsProcessorImpl`.
- I den konkrete *decorator*-klassen, som er `WebProcessor`, blir metodene som trenger å forandres overkjørte.

Dette gjør det mulig å utvide funksjonalitet når applikasjonen kjører uavhengig av andre instanser. Hadde man gjort dette med subklasser ville det blitt gjort når koden ble kompilert og det ville påvirket alle instansene av den originale klassen. Man kan da forandre funksjonalitet med å lage en ny klasse som arver fra `AbstractProcessorDecorator`, også overkjøre de metodene som man vil forandre. `FluxProcessor` er ikke med i mønsteret, det er en servlet [59] som håndterer JavaScript-kall.

6.1.2 Integrering av BetterForm i DPG

Det er to mulige måter å integere BetterForm i DPG. Man kan bruke den eksisterende løsningen med hele web-applikasjonen eller utvide `XFormsProcessor`-klassen og lage sin egen web-applikasjon. Fordeler med å bruke den eksisterende løsningen:

- Den er raskere å integrere.
- Den har blitt testet og er i bruk.

Ulemper med å bruke den eksisterende løsningen:

- Den benytter seg av en løsning med servlets som ikke vil fungere med plugin-arkitekturen i DPG.
- Man blir nødt til å utvide funksjonaliteten for å få den eksisterende løsningen og plugin-arkitekturen til å samarbeide.

Fordeler med å lage en egen løsning:

- Den vil være godt tilpasset DPG.
- Det vil være enkel å integerere.

Ulemper med å lage en egen løsning:

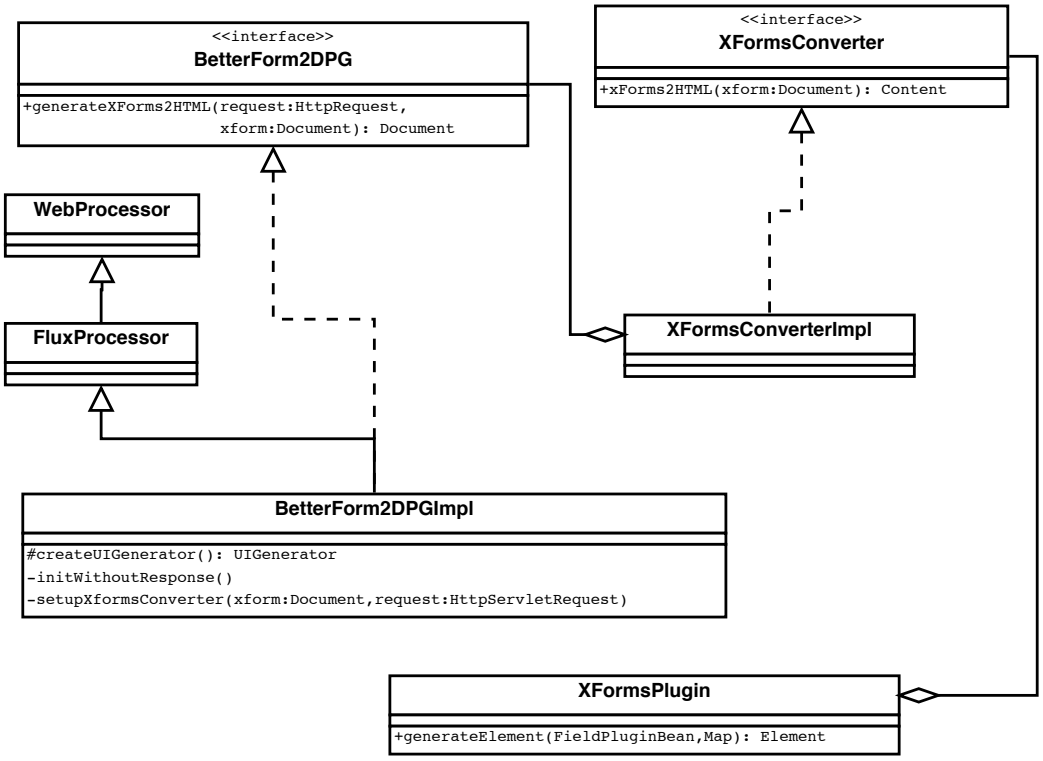
- Det vil ta lang tid å utvikle et ny web-applikasjon.
- Den må testes omfattende for å være sikker på at all funksjonalitet er intakt.
- Når BetterForm oppdateres vil det være større sjanse for at biblioteket ikke er kompatibelt med den applikasjonen man har laget selv.

Den eksisterende løsningen kan utvides slik at man kan bruke den med plugin-arkitekturen. Man kan utvide løsningen med `Adapter`-designmønsteret [47]. Man lager en kontrakt som gjør det mulig å integerere de eksisterende løsningen, uten å måtte forandre på koden i selve BetterForm. Det kunne blitt en bedre løsning med å lage en ny versjon tilpasset DPG, men det er tidkrevende og kan ha mangler som gjør at man mister funksjonalitet. Valget blir å bruke den eksisterende løsningen og tilpasse den til plugin-arkitekturen.

6.2 Utvidelse av BetterForm

`WebProcessor`-klassen har alle dataene man trenger for å kunne rendre XForms-skjemaet med plugin-arkitekturen til DPG. Prosessoren har ikke noen metoder for å hente ut XForms-skjemaet som er transformert til JavaScript og HTML. Dette er fordi prossesoren benytter seg av servlets som sender rundt informasjonen som parametere i `request`-objektet. Det transformerte XForms-skjemaet blir sendt til brukeren i `response`-objektet. Plugins-arkitekturen kan ikke benytte seg av denne flyten fordi plugins har ikke tilgang på `response`-objektet. Plugins leverer fra seg et JDOM-element som blir rendret i et utsnitt av DPG. Se kapittel 3 for mer informasjon. Skulle man bruk løsningen til BetterForm måtte man ha omstrukturert fullstendig om hvordan DPG bruker MVC-modulen til Spring [73]. Dette ville også ødelagt MVC-delen, fordi plugins skal legge til sitt innhold i et utsnitt og ikke med bruk av `response`-objektet.

Løsningen blir å bruke `adapter`-designmønsteret for å la pluginen kunne kalle metoder på prosessoren uten at man bruker servlets som et mellomledd. Det er derimot et problem fordi man kan ikke bare opprette en instans av `WebProcessor`. `WebProcessor` må opprettes igjennom en `WebFactory` for at skjemaet skal bli registrert og kjøres i containeren på serveren.



Figur 6.2: UML-diagram av BetterForm integrering

Figur 6.2 viser designet på løsningen. Grunnen til at det ble så mange klasser er fordi man trenger å instansiere `FluxProcessor`-klassen fra `WebFactory`-klassen for å registrere XForms-skjemaet. Man må også implementere ny funksjonalitet i `FluxProcessor`-klassen. Den ny funskjonaliteten er å kunne hente ut det transformerte XForms-skjemaet og sette opp prossesoren uten `response`-objektet. Derfor er det ett ekstra lag. Hadde det ikke blitt brukt en `Factory`-metode [47] ville det bare vært nødvendig med én kontrakt. `WebFactory`-klassen lager et objekt som må være en instans av `WebProcessor`, derfor vil man ikke ha den avhengigheten av BetterForm i pluginen. `BetterForm2DPGImpl` arver fra `FluxProcessor` fordi det er den klassen som blir brukt av web-applikasjonen. `FluxProcessor` arver så fra `WebProcessor`. `BetterForm2DPGImpl` inneholder de nye metodene som gjør det mulig for plugins til å transformere XForms-skjemaet til JavaScript og HTML.

Kontrakten `BetterForm2DPG` har en metode `generateXForms2HTML` som returnerer HTML og JavaScript, som er laget ut fra XForms-skjemaet. Klassen `BetterForm2DPGImpl` har metoden `setUpXFormsConverter`, som tar inn XForm-skjemaet og ett `request`-objekt. BetterForm trenger `request`-objektet for den må lagre sesjons-ID og legge til XForms-skjemaet i objektet. Objektet `request` blir der-

etter plukket opp av XFormsFilter-servleten, som henter ut skjemaet og kjører det på serveren. Metodene initWithoutResponse, setUpXFormsConverter og createUIGenerator i BetterForm2DPGImpl blir brukt for å intialisere FluxProcessor-delen av klassen. Alle disse metoden blir kalt i generateXForms2HTML-metoden som er implementert i BetterForm2DPGImpl-klassen. Dette vises i figur 6.3. Grunnen til at intialiseringen blir gjort av en metode er for å gjøre BetterForm2DPG-kontrakten enkel. Kandidaten valgte å lage kontrakten BetterForm2DPG for å gjøre designet mer kompatibel mot nyere versjoner av BetterForm.

I figur 6.2 ser man at XFormsConverter-kontrakten bare har metoden xForms2HTML(Document xform). Dette gjør det enkelt å skifte ut den gjeldene implementasjonen. XFormsConverterImpl instansierer et objekt av kontrakten BetterForm2DPG med klassen BetterForm2DPGImpl gjennom WebFactory. Dette blir vist i figur 6.2. XFormsConverterImpl kaller metoden generateXForms2HTML og returnerer XForms-skjemaet som nå er transformert til HTML og JavaScript. JavaScript-delen blir fjernet før innholdet blir returnert til XFormsPlugin, fordi JavaScript-koden er allerede lagt til i sidemalen. Dette blir forklart i avsnitt 6.3. Klassen WebFactory instansierer et objekt som blir definert i betterform-config.xml. Det gjør det mulig å skifte hvilken klasse som skal bli instansiert når applikasjonen kjører.

Listing 6.1: Valg av klasse for å instansiere XFormsConverter

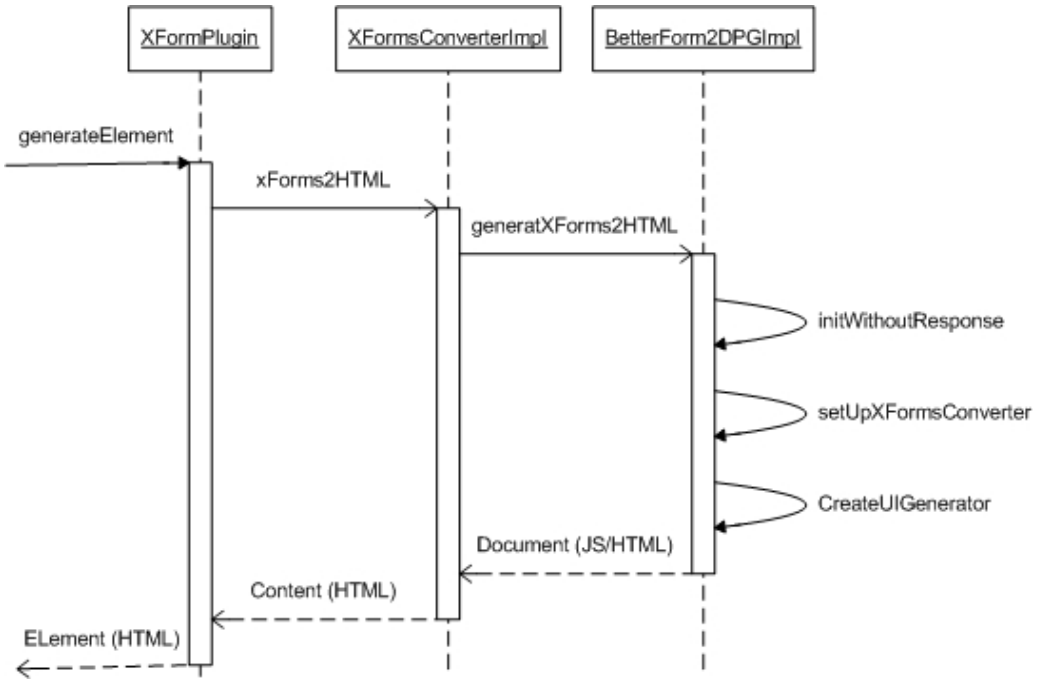
```
1
2 <bean id="xFormsConverter"  class="no.uib.ii.dpg2.plugins.xform.↩
    XFormsConverterImpl" />
```

Figur 6.2 viser XFormsPlugin, som er plugin-implementasjonen, som gir JDOM-elementet med HTML-snutten til DPG som rendrer den. Kapittel 3 forklarer prossesen men rendring i DPG. Pluginen har en referanse til kontrakten XFormsConverter, og man definerer i filen applicationContext-service.xml hvilken klasse den skal bli instansiert til. Listing 6.1 viser hvordan man definerer klassen som skal instansiere XFormsConverter. Man kan bytte ut instansen når applikasjonen kjører. Referansen XFormsConverter blir med dette oppsettet satt til å bli instansiert ac XFormsConverterImpl-klassen.

Figur 6.3 viser hvordan DPG kaller generateElement() på pluginen. Pluginen kaller så metoden xForms2HTML(Document xform) på XFormsConverterImpl. XFormsConverterImpl lager en instanse av BetterForm2DPG med WebFactory. Denne instansen er satt til å være BetterForm2DPGImpl. XFormsConverterImpl kaller metoden generateXForms2HTML på BetterForm2DPGImpl-klassen. Klas-

sen `BetterForm2DPGImpl` kaller så metodene for å initialisere seg selv. Dette blir vist i figur 6.3.

Metoden `generateXForms2HTML` returnerer det transformerte XForms-skjemaet, som nå består av HTML og JavaScript-kode, til `XFormsConverterImpl`-klassen. I `XFormsConverterImpl` blir JavaScript-koden fjernet. HTML-snutten blir returnert til `XFormsPlugin`-klassen som leverer den til DPG som rendrer skjemaet. HTML-snutten inneholder ID-er på inndatafelter, som referer til elementer i `Container`-instansen, som kjører i `XFormsProcessorImpl`-klassen.



Figur 6.3: Sekvensdiagram for BetterForm i DPG

6.3 JavaScript i Velocity-malene

Når BetterForm transformerer XForms-skjemaet om til HTML og JavaScript, er alltid JavaScript-delen lik. Det er fordi JavaScript-et bruker ID-ene satt i HTML-koden og bruker de samme metodene hver gang. Defor er det en bedre løsning å legge til JavaScript-delen i selve `Velocity`-malen [22] til hovedsiden. Da vil JavaScript-ene til BetterForm bli importert i `head`-elementet i hovedmal, som innolder menyer og lenker som hver side har. Dette er samme stedet hvor DPG sine JavaScript-filer blir importerte. Da vet man hvor all importeringen av JavaScript-ene skjer, og de

kan lett fjernes eller endres.

Det som er negativt med denne løsningen er at man laster ned BetterForm sine JavaScript på sider man ikke trenger de på. Dette skjer bare første gangen, fordi skriptene blir lagret i nettleseren. Tillegg viser JavaScript-ene som blir lagt til i hovedmalen. Alle JavaScript-filene BetterForm trenger, er blitt lagt i `JavaScript`-mappen som ligger under presentasjonsmønsteret man ønsker å bruke.

6.4 Konfigurasjon av BetterForm i DPG

Det er nødvendig å legge til forskjellige konfigurasjonsfiler i DPG for at BetterForm skal fungere. Filene legges i stien `src/main/webapp/WEB-INF/`. Filene er som følger:

1. `betterForm-config.xml` inneholder informasjon om hvor man finner ressurser, kobling mellom klasser og ID-er og feilmeldinger.
2. `dwr.xml` inneholder instillingene til DWR-implementasjonen [65].
3. `dwr20.dtd` er valideringen av filen `dwr.xml`.
4. `log4j.xml` inneholder instilligene til loggeren som BetterForm bruker.

Man kunne integrert disse i filene i konfigurasjonsfilene DPG allerede har, men det er bedre å klart avskille de to applikasjonene. I `betterForm-config.xml` trenger man å definere hvilken klasse som skal bli laget av `WebFactory`. Det blir vist i eksempel 6.2, hvor prosessoren er satt til å peke til `BetterForm2DPGimpl`. Det er også nødvendig å legge til XSLT-transformasjonene i DPG. De plasseres i stien `src/main/webapp/WEB-INF/classes/META-INF/resources`.

Listing 6.2: Nødvendige endringer i betterForm-config.xml

```
1
2 <useragent name="dojo"
3   processor="no.uib.ii.dpg2.plugins.xform.BetterForm2DPGimpl"
4     description="AJAX-enabled useragent with packed resources for ↩
          production environments">
5 </useragent>
```

6.4.1 Endring av DPG sine konfigurasjons-filer

Det er nødvendig å legge til informasjon om DWR-implementasjonen [65] og `XFormsFilter`-klassen i filen `web.xml`. Endringene i `web.xml` blir vist i eksempel 6.3. Klassen `XFormsFilter` er en servlet som kjører i web-containeren og tar imot XForms-skjemaer som blir registrert av `WebFactory`. `XFormsFilter`-klassen blir definert på linje 6-13 som et filter, og det er satt hvilken implementasjon den skal bruke og hvilken `useragent` den skal bruke. Parameteret `useragent` har en verdi som forteller hvilken XSLT-transformasjon som skal brukes.

DWR-implementasjonen er også en servlet, den kommuniserer med JavaScriptet som kjører i nettleseren. Hvordan DWR-implementasjonen fungerer er vist i figur 6.4. På linje 20-27 er DWR-implementasjonen definert som en servlet, med navnet `Flux`. Den blir definert med ett parameter, som definerer om det skal bli skrevet ut feilmeldinger. På linje 29-32 blir stien til DWR-implementasjonen satt til `/Flux/*`. Det vil si at DWR-implementasjonen vil kjøre bak denne stien. I listing 6.3 blir Det er også nødvendig å spesifisere hvor `betterForm-config.xml` befinner seg, det blir gjort på linje 1-4.

Listing 6.3: Endringer i web.xml

```
1   <context-param>
2     <param-name>betterform.configfile</param-name>
3     <param-value>WEB-INF/betterform-config.xml</param-value>
4   </context-param>
5
6   <filter>
7     <filter-name>XFormsFilter</filter-name>
8     <filter-class>de.betterform.agent.web.filter.XFormsFilter</filter-↩
          class>
9         <init-param>
10            <param-name>useragent</param-name>
11            <param-value>dojo</param-value>
12        </init-param>
13    </filter>
14
15    <filter-mapping>
16    <filter-name>XFormsFilter</filter-name>
17    <url-pattern>*.xhtml</url-pattern>
18  </filter-mapping>
19
20  <servlet>
21        <servlet-name>Flux</servlet-name>
22        <servlet-class>org.directwebremoting.servlet.DwrServlet</↩
              servlet-class>
23        <init-param>
```

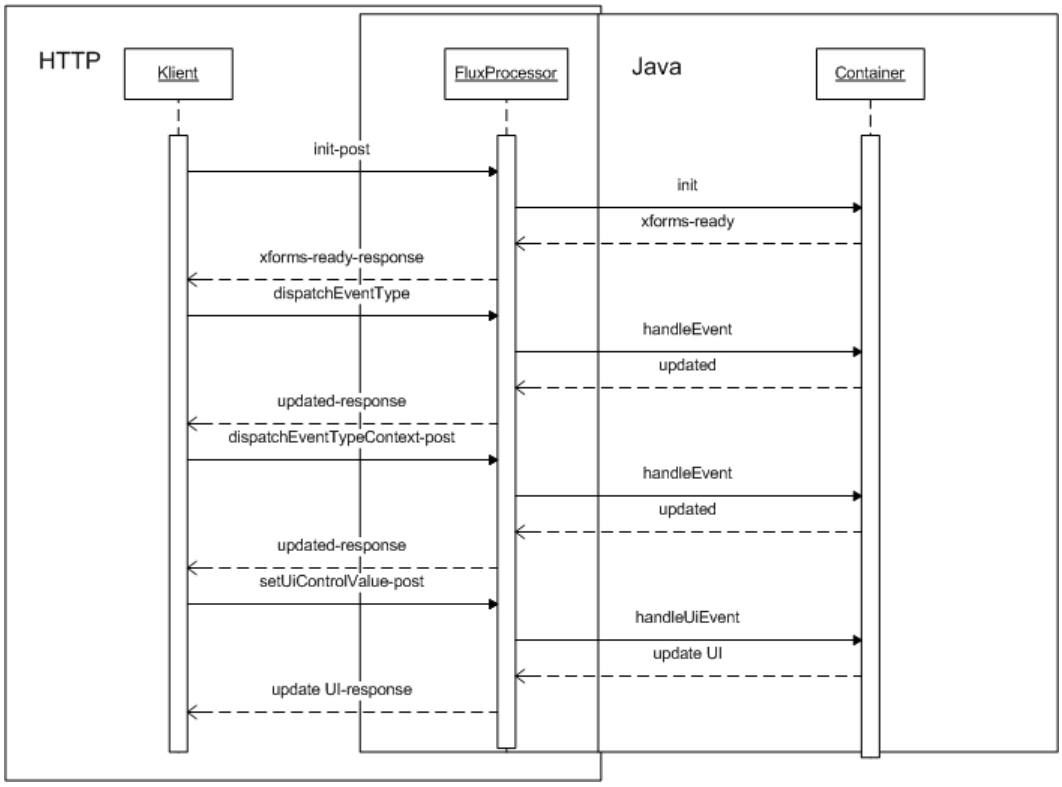
```
24              <param-name>debug</param-name>
25              <param-value>false</param-value>
26          </init-param>
27      </servlet>
28
29    <servlet-mapping>
30          <servlet-name>Flux</servlet-name>
31          <url-pattern>/Flux/*</url-pattern>
32      </servlet-mapping>
```

`FluxProcessoren` er selve DWR-implementasjonen. Hvordan prosessoren fungerer blir vist i figur 6.4. `FluxProcessor`-klassen kommuniserer med `Klient`, som er nettleseren. Dette blir gjort gjennom HTTP-protokollen [41]. JavaScript-koden på klienten sender `post-request` til prossesoren som igjen gjør forandringer på instansen av XForms-skjemaet. XForms-skjemaet blir så validert og oppdatert i `Container`-klassen. `Container`-klassen sender forandringer eller feilmelding tilbake til `FluxProcessor`-klassen. `FluxProcessor`-klassen sender så en `response`-melding til klienten. Meldingen har den nødvendige informasjonen for å synkronisere brukersiden med XForms-instansen. Figur 6.4 viser hvordan hele syklusen er med meldingene som blir sendt mellom komponenetene. Først blir intialiseringsmeldingen sendt og prossesoren svarer at den er klar. Det blir så sendt meldinger som har forskjellige hendelser å gjøre for eksempel at brukeren trykker på en XForms-komponent eller forandrer data i inndatafeltet.

6.5 Eksempel på XForms-skjema i DPG

Figur 6.5 viser et eksempel på et XForms-skjema rendret i DPG ved hjelp av `XFormsPlugin`-klassen. Skjemaet lar brukere melde seg inn i et forum. Det er satt begrensinger på noen felt, for eksempel må *Brukernavn* være lengre enn 3 bokstaver og *Passord* må være minst 6 bokstaver. Man må bekrefte passordet en gang til, og det må være likt som første gang man skrev det.

Det er et spørsmål om man eier en PC eller ikke. Svarer man ja på dette spørsmålet vil det dukke et nytt spørsmål om hvilket operativsystem (OS) man bruker. Dette er et eksempel på *skip-logic*, som er å bestemme om spørsmål er relevante ut ifra andre spørsmål. Det også et eksempel på *rekkevidde* (eng. *range*), hvor det blir spurt hvor ofte man vil bruke forumet om dagen. Man kan velge fra antall fra 1 til 10. Til slutt er det et felt hvor man kan skrive inn datoen man er født, og når man trykker på den dukker det opp en datovelger. Ser man ved siden av boksene som tar inn data, er det gitt tilbakemelding på at alt er i orden eller hva som er galt. XForms-skjemaet er lagt ved i oppgaven i tillegg B.

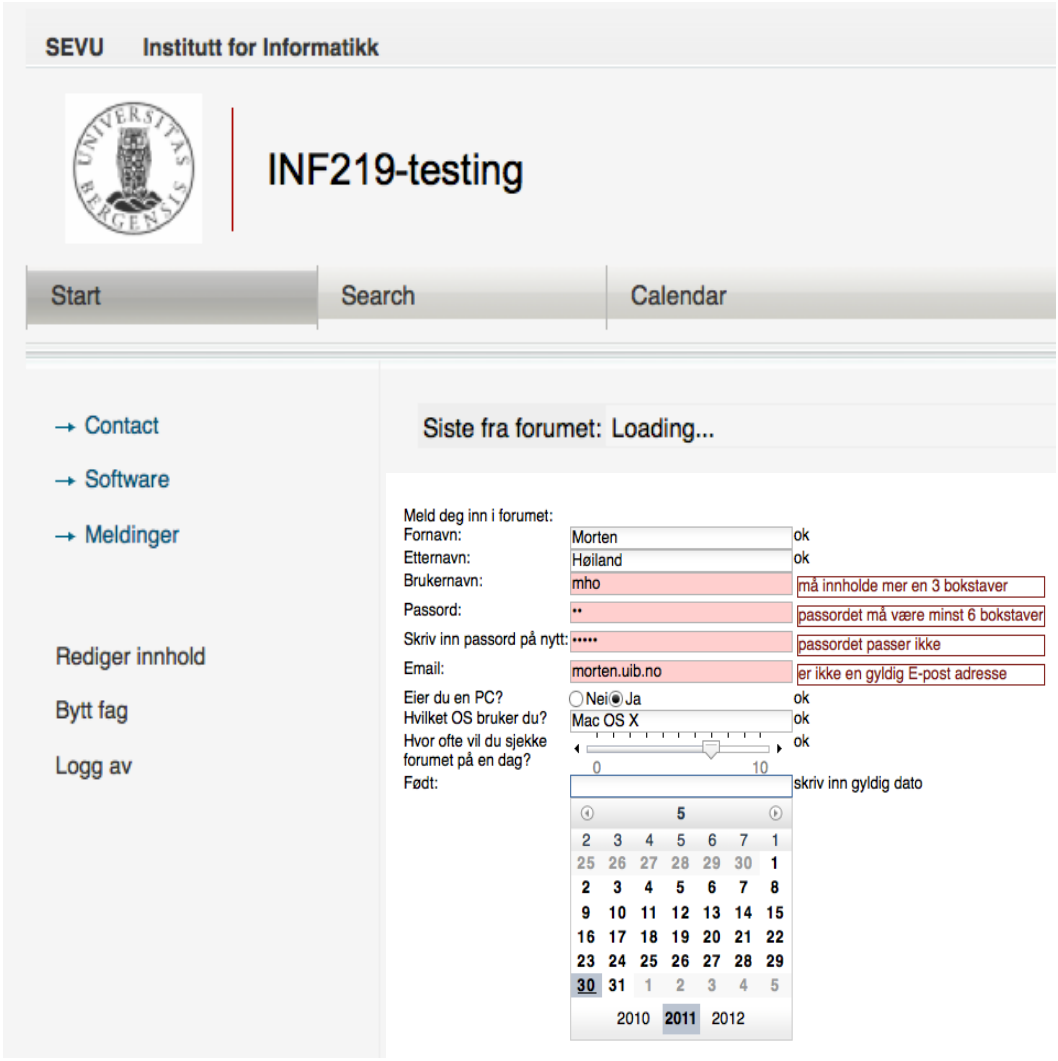


Figur 6.4: Hvordan FluxProcessor fungerer

6.6 Vurdering av integrasjonen av BetterForm

BetterForm sin dokumentasjon forklarer godt hvordan applikasjonen er bygget opp. Dokumentasjonen forklarer derimot ikke på en god måte hvordan konfigurasjonen fungerer, og hva som trengs for å få viktige komponenter til å fungere. Det har vært en lang prosess med prøving og feiling for å få til integreringen på en fornuftig måte. De viktigste punktene med integreringen er:

- Plugin-arkitekturen er nå kompatibelt med BetterForm, fordi man bruker *Adapter*-designmønsteret. Bruken av mønsteret gjør det mulig å skifte til nyere versjoner av BetterForm, bare ved å skifte ut biblioteket.
- Alle komponentene til BetterForm er løst koblet til DPG, bortsett fra JavaScript i sidemalen og endringene i `web.xml`.
- Det er blitt utviklet en plugin som heter `XFormsPlugin`, som gjør det mulig å rendre XForms-skjemaer i DPG.



Figur 6.5: XForms-skjema rendret i DPG

7

Vurdering av plugin-ressurser

7.1 Håndtering av plugin-ressurser i DPG 2.1

I Tobias Rusås Olsen [54] og Peder Skeidsvoll [70] sine masteroppgaver ble det gjort følgende forandringer for ressurshåndteringen til plugins:

- Plugins kan motta inndata fra brukerrollen *reader*.
- Plugins kan lagre, slette og hente data fra persistenslaget til DPG.

Dette er viktig for kandidaten sine mål, fordi det gir muligheten til å motta instansdataene fra XForms-skjemaene og lagre disse i ressursene til plugins.

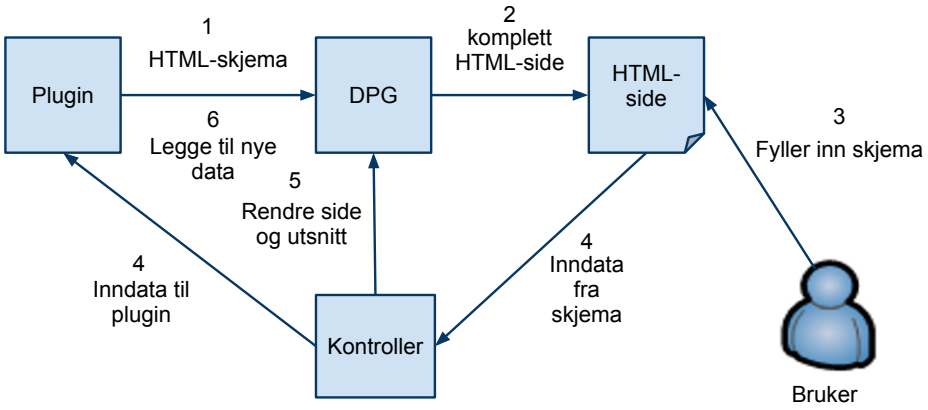
7.2 Svakheter med plugins-ressurser i DPG 2.1

I dette kapittelet vil ressurshåndteringen til plugins bli gjennomgått, for å finne svakheter og deler som ikke vil fungere med XForms-skjemaer. Deretter vil en rekke løsninger bli vurderte og en løsning vil bli valgt og implementert.

7.2.1 Inndata til plugins

Figur 7.1 viser hvordan inndata til plugins blir håndtert. Saksgangen er gitt ved følgende steg:

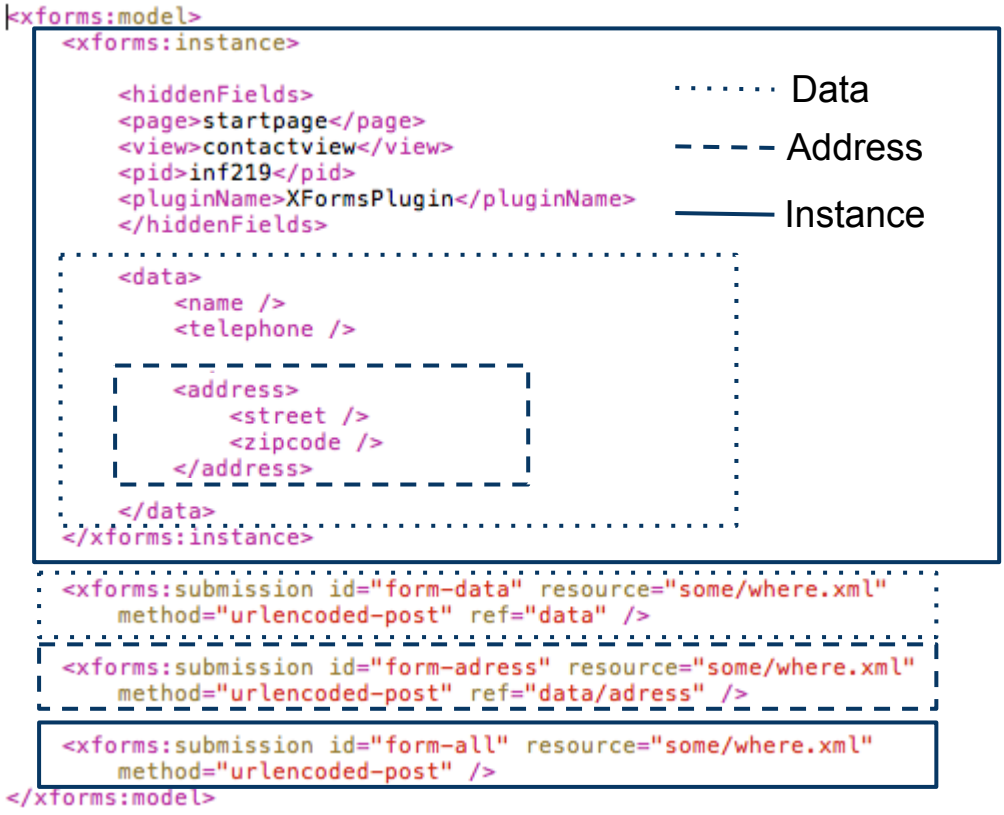
1. Pluginen lager skjemaet, med skjulte felter (eng. *hidden fields*) som innholder informasjon om i hvilken presentasjon, side, utsnitt og plugin-navn dataene blir hentet fra.
2. DPG rendrer så skjemaet for brukeren.
3. Brukeren fyller inn data i skjemaet og sender det inn.
4. Dataene blir sendt til kontrollerklassen som henter ut dataene og sender de videre til pluginen.
5. Kontrolleren sender så brukeren tilbake til siden hvor skjemaet ble sendt inn.
6. Pluginen har da muligheten til å legge til nye data til siden som blir rendret om den ønsker det.



Figur 7.1: Inndata til plugins

Punkt 5 i saksgangen om hvordan inndata til plugins blir håndtert, er hvor de skjulte feltene blir brukt. De skjulte feltene inneholder informasjon om hvilke side og utsnitt man skal sende brukeren til etter skjemaet er blitt sendt inn. Løsningen i DPG er at man blir sendt tilbake på den siden hvor man sendte inn skjemaet. Konseptene *side* og *utsnitt* blir forklart i kapittel 2.

Bruken av skjulte felter skaper problemer med bruk av XForms-skjemaer. Det er enkelt å legge til skjulte felter i XForms-skjemaer. Skjulte felter i XForms er instansdatafelter som er fylt inn fra før og er ikke koblet mot et inputfelt, dette blir illustrert i figur 7.2. Elementene i `hiddenFields`-elementet er skjulte felter. Problemet er at det er mulig å sende inn forskjellige deler av instansdataene ved hjelp av `ref`-attributtet i `submission`-elementet. Det blir derfor vanskelig å legge til de skjulte feltene automatisk, for de må legges til i alle elementene som kan bli sendt inn.



Figur 7.2: Skjulte felter i XForms-skjema

I figur 7.2 ser man et eksempel på hvordan instansdata blir koblet til `submission`-elementet og hvilke deler som blir sendt inn. Hvis man velger å koble opp `submission`-elementet med `id`-attributtet `form-all` til en send-knapp, vil alle feltene inni boksen med hele linjer bli sendt inn. Kobler man send-knappen opp mot elementet med `id`-attributtet `form-adress`, vil bare instansdataene inni den striplete boksen bli sendt inn, og man får ikke med de skjulte feltene. Da vil ikke kontrollerklassen sende brukeren til rett side og utsnitt.

Man blir også tvunget til å bruke `urlencoded-post`, som er metoden HTML-skjema sender inn data på. Bruker man `urlencoded-post` med XForms vil man miste en viktig del av det som gjør XForms nyttig. Data vil bli sendt inn som nøkkel-verdi par, og ikke som XML. Man vil derfor miste den komplekse datastrukturen Xforms-skjema kan ha på sine instansdata. For å sende inndata som XML med XForms-skjemaer må attributtet `method` i elementet `submission` være satt til `post`. Når data blir sendt inn fra XForms som XML, må de bli hentet ut fra `content` til `request`-objektet i stedet for å bli hentet ut som parametere i `request`-objektet.

7.2.2 Ressurstilgangen til plugins

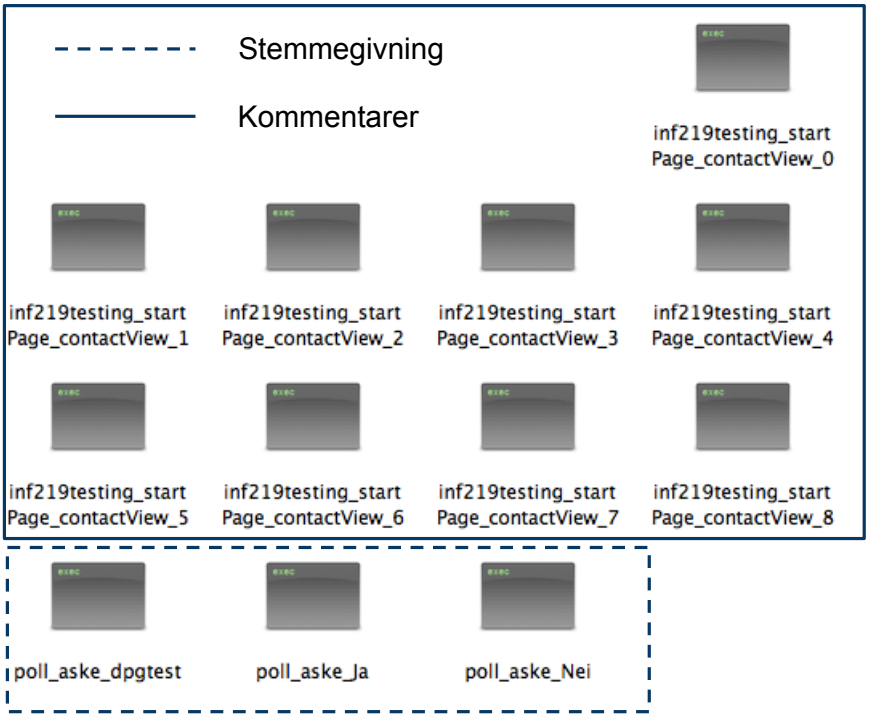
I Tobias Olsen [54] og Peder Skeidsvoll [70] sine masteroppgaver kommer det frem at plugins kan overskrive hverandre sine ressurser. Det er uheldig om man for eksempel har en spørreundersøkelse og en annen plugin overskriver eller sletter dataene. Det er veldig frustrerende for brukeren om dataene må fylles inn på nytt. Kandidaten mener det er essensielt når man samler inn større mengder med data, at de skal bevares sikkert.

7.2.3 Uoversiktlig lagring av ressurser

Når det er mange plugins som deler samme mappe for lagring av ressurser blir det fort uoversiktlig. Skal man da gjøre noe manuelt som flytte ressursene til en annen presentasjon blir det en tidkrevende oppgave. Figur 7.3 viser et eksempel på hvor mange filer det blir. Eksempelet har bare ressurser fra to plugins, `PollPlugin` og `SaveComment`, hvor det er en stemmegivning og 9 kommentarer. Ressursene i eksempelet har forskjellige navnekonvensjon ut i fra hvilke plugin de tilhører. Filer fra stemmegivningen-pluginen er i den boksen med striplete linjer og filer fra kommentar-pluginen er i den boksen med hele linjer.

`PollPlugin` bruker i figur 7.3 `poll_aske` som grunnstamme, og `poll_aske_ja` er en fil hvor det står hvor mange som har svart ja. Den siste filen `poll_aske_dpgtest` er hva brukeren har stemt, brukeren her har navnet `dpgtest`. `SaveComment` bruker konvensjonen `presentasjonsID_side_utsnitt_X`, hvor `side` og `utsnitt` referer til siden og utsnittet kommentaren hører til i den presentasjonen med `presentasjonsID`-en. `X` er et heltall som blir inkrementert når det blir lagt til nye kommentaterer.

Det er ikke lagt noen føringer på hvordan navnekonvensjonen og formatet på filer skal være. Det er opp til hver enkelt plugin-utvikler å bestemme dette. Dette gjør det vanskelig å finne filer og dele dataene i filene med andre plugins eller applikasjoner.



Figur 7.3: Mappe med ressurser fra plugins

7.3 Forbedringer av ressurshåndtering

Det har blitt avdekket følgenede svakheter med ressurshåndteringen til plugins:

- Kontrolleren for inndata som DPG benytter seg av er vanskelig å bruke med XForms-skjemaer.
- Alle plugins deler samme ressursmappe. Det er uoversiktlig.
- Plugins har tilgang til andre plugins sine ressurser.
- Det er opp til pluginutvikleren å bestemme i hvilket format ressursene skal bli lagret i og hvilke ID-er ressursene skal ha.

Kandidaten mener at alle disse problemene er relevante for problemstillingen til masteroppgaven. Skal man bruke XForms må man sørge for å lagre dataene på en forsvarlig måte og sørge for at det blir enklest mulig å bruke XForms uten at det skal påvirke resten av DPG negativt. Det vil være fokus på å lage løsninger som egner seg for XForms-skjemaer og løsningene skal kunne brukes av andre teknologier.

7.3.1 Løsning for inndata fra XForms

En mulig løsning er å legge til de skjulte feltene i alle de forskjellige elementene som blir sendt inn. For eksempel i figur 7.2 må man legge til `hiddenFields` under både `data`-elementet og `address`-elementet. Dette er ingen god løsning for det strider imot beste praksis med bruk av XForms-skjema. De skjulte feltene har heller ikke noe å gjøre med dataene som skjemaet vil samle inn. Fordelen er at man kan bruke den eksisterende løsningen for innsending av data, men man må bruke nøkkel-verdi datamodellen som gjør at man mister XML-strukturen XForms-skjemaene har på sine inndata.

En bedre løsning er å lage en kontroller som tar imot inndata ved hjelp en REST-metode [66]. Da kan man definere alle de nødvendige dataene kontrolleren trenger i URL-en til der man sender inn skjemaet. Dette vil fungere for både HTML-skjemaer og XFoms-skjemaer på en fornuftig måte. Slik listing 7.1 og 7.2 viser, bruker begge nesten samme metode for å sende inn skjema til en spesifikk URL. Da slipper man å bruke skjulte felter. REST gjør at man kan hente ut informasjonen man trenger fra URL-en. Kontrolleren sjekker hvilke MIME-type innholdet har i `request`-objektet og velger den rette metoden for å hente ut dataene. MIME er en tekststreng som forteller mottakeren hvilke type innholdet er.

Listing 7.1: Eksempel på hvordan XForms-skjema definerer hvor inndata skal bli sendt til.

```
1
2 <xforms:submission id="form1" resource="/presentationId/page/view/↩
      pluginName"
3 method="post"/>
```

Listing 7.2: Eksempel på hvordan HTML-skjema definerer hvor inndata skal bli sendt til.

```
1
2 <form name="input" action="/presentationId/page/view/pluginName" method↩
      ="post">
```

Listing 7.3 viser hvordan REST-metoden er laget. Det er enkelt å lage REST-metoder ved hjelp av Spring-rammeverket [73], man trenger bare benytte `@RequestMapping`-annotasjonen.

Listing 7.3: REST-metoden i kontrolleren

```
1
2
3   @RequestMapping(value="/pv/{pid}/{page}/{view}/{pluginName}/
4   {userName}", method=RequestMethod.POST)
5   public ModelAndView getParameters(@PathVariable("pid") String pid,
6   ...){
7   ...
8   }
```

7.3.2 Uoversiktlig lagring av plugin-ressurser

DPG benytter seg nå av filbasert lagring, men det kan lett gjøres om til å bruke Jackrabbit [8]. Ser man på navnekonvensjonene brukt i figur 7.3, så er det basert på `presentasjon`, `side`, `utsnitt`, egne ID-er og `brukernavn`. En mulig løsning er å bruke DPG sin struktur på presentasjoner i en mappestruktur i pluginressursmappen. Man vil da få en mappestruktur som dette:

`/plugin-resources/side/utsnitt/pluginNavn/`

`plugin-resources` er mappen der plugins kan lagre ressurser. `side` og `utsnitt` er hvor pluginen er definert i presentasjonen. `presentasjon` er ikke tatt med for det er allerede gitt hvilken presentasjon det gjelder for `plugin-resources`-mappen ligger i presentasjonen sin mappe. `pluginNavn` er hvor de endelige ressursene vil ligge. Da kan man enkelt se at pluginen som er i den siden i det utsnittet og med det gitte navnet har sine ressurser der.

Man kan da enkelt flytte, kopiere og slette ressurser som hører til den pluginen man ønsker. Dette er også en løsning som gjør det enklere å sette restriksjoner på hvilke plugins som har adgang til de forskjellige mappene. Strukturen kan enkelt overføres til Jackrabbit implementasjonen for den bruker en lignende struktur som filsystem bruker [8].

7.3.3 Begrensning av ressurstilgang for plugins

I DPG har plugins fri tilgang til andre plugins sine ressurser. Har man en ondsinnet plugin i systemet kan den slette og forandre ressurser som hører til andre plugins. Det kan også oppstå komplikasjoner som at data blir overskrevet eller slettet. Tobias Olsen [54] og Peder Skeidsvoll [70] laget en ny kontrakt `PluginResourceDAO`, som gir de metodene plugins trenger for å lagre, slette, hente og sjekke om ressurser finnes. Derfor er det naturlige å sette begrensninger på metodene i den kontrakten.

Kandidaten har vurdert Spring Security [72], JAAS [58] og Apache Shiro [56] for autorisasjon på metodenivå. JAAS og JSecurity trenger konfigurasjonsfiler for å kunne brukes. Spring Security har annotasjoner som sjekker om forskjellige vilkår er oppfylt før og etter metoden blir kjørt. Er ikke vilkårene oppfylt vil ikke metoden kunne kjøres eller returnere noe. DPG bruker allerede Spring Security til autorisasjon for de forskjellige delsystemene. Kandidaten velger å bruke annotasjonsløsningen til Spring Security, fordi den krever ingen konfigurasjonsfiler og DPG bruker dette biblioteket allerede.

Hvordan skal plugins kunne lagre hvilke plugins som har tilgang til deres ressurser? Dette minner mye om problemstillingen, Tobias Olsen [54] og Peder Skeidsvoll [70], hadde når de skulle tilby muligheten å gi bedre navn til plugins. De foreslo mulig løsninger om å lage en abstrakt klasse som mellomledd mellom pluginet og kontrakten. De mente også det kunne vært enkelt å lage en metode i kontrakten for å hente ut de pluginene som kunne aksessere ressursene til den gitte pluginen. Det anbefales å implementere en kontrakt i stedet for å utvide en klasse. Det er heller ikke optimalt å ha for mange unødvendige metoder i kontrakten.De kom frem til at ingen av de løsningene var bedre enn å bruke annotasjoner. Annotasjoner gir muligheten til å legge til pluginsen som skal ha tilgang uten påvirke selve kontrakten og kan lett fjernes og endres.

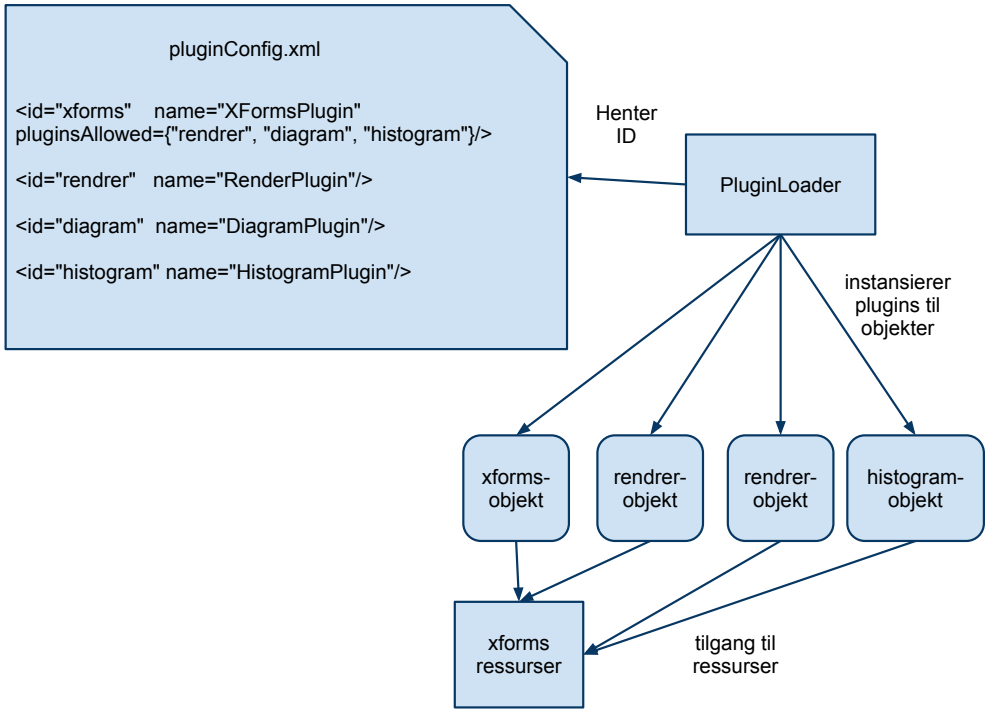
Valget blir da å lage en annotasjon som tar inn en liste over de pluginsene som har tilgang til den pluginen som har annotasjonen. Det er et problem å velge hvilke ID-er vi skal bruke i annotasjonen. Kandidaten har kommet frem til følgende løsninger:

- Gjøre plugins om til å bli instansiert ut i fra ID-en satt i `pluginConfig.xml`.
- Begrense tilgang ut fra navnet på plugins.
- Begrense tilgang ut i fra plugin-navn, side og utsnitt.

7.3.4 Instansiering av plugins ut i fra pluginConfig.xml

Plugins er *singletons*, som betyr at man ikke kan lagre data i instansvariablene, fordi det lages bare ett objekt per plugin [54]. Ser man på figur 7.4 viser den hvordan man kunne koblet sammen `PluginConfig.xml` og `PuginLoader`-klassen, slik at hver plugin blir instansiert med ID-en satt i `PluginConfig.xml`. Da kunne man definert i `PluginConfig.xml` hvilke plugins som har tilgang til en gitt plugin. I figur 7.4 er `xforms` ID-en til en instans av `XFormsPlugin`-klassen, og alle plugins definert i `pluginConfig.xml` skal ha tilgang til `xforms`-instansen sine ressurser. Dette blir oppnåd med å bruke `allowePlugins`-attributtet som sier hvilke plugins som har tilgang til den pluginen sine ressurser.

Denne løsningen vil gi kontroll på hvilke plugins som har har tilgang til andre plugins ressurser. Problemet med denne løsningen er at det blir veldig intrikat for presentasjonsmønsterutvikleren å holde orden på hvilke plugins som har tilgang til andre plugins. Det er fordi utvikleren holder styr på ID-ene satt i `PluginConfig.xml`, og må derfor fikse dette manuelt for alle plugins. Dette er en tidkrevende jobb. Figur 7.4 viser en plugin med tre andre plugins som har tilgang.

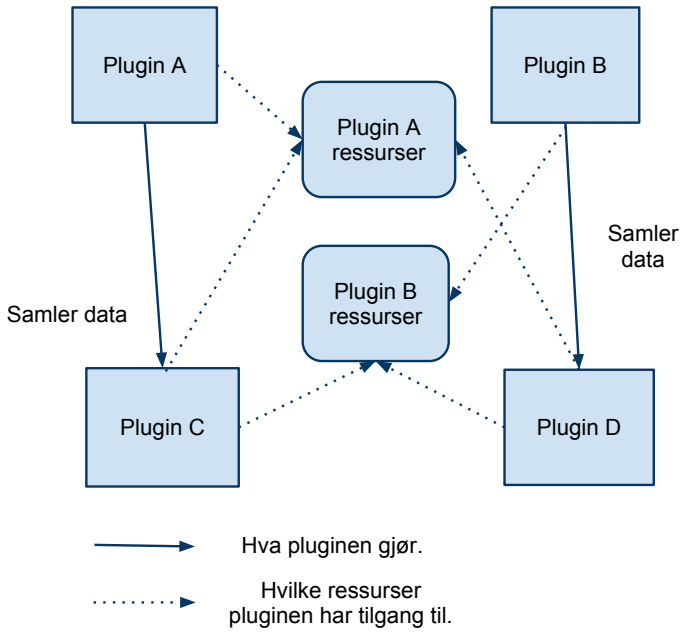


Figur 7.4: Instansiering av plugins med ID fra `pluginConfig.xml`

7.3.5 Begrense ressurstilgang med plugin-navn

Begrense ressurstilgang ut i fra navnet til pluginen er annen løsning. Problemet er presentert i figur 7.5. Plugin A og B er samme plugin, men i forskjellige sider og utsnitt. Plugin A samler inn data for plugin C, som kan for eksempel lage et kakediagram av dataene. Plugin B samler inn data for plugin D, som lager for eksempel et histogram av dataene. Det som er negativt er at både plugin C og D har tilgang til ressursene til både plugin A og B. Det er for plugin A og B har samme navn definert i annotasjonen selv om de opptrer på ulike plasser i presentasjonen. De deler ikke samme ressursmappe på grunn av den nye mappestrukturen. Med denne løsningen er det enkelt for programmereren av plugins å definere hvilke andre plugins som har

tilgang til ressursene. En bivirkning av den nye mappestrukturen blir at de plugins som har tilgang til andre sine ressurser må definere stien til mappen i kildekoden eller som parametere i `pluginConfig.xml`. For eksempel i fra figur 7.5 så må plugin C definere i hvilken side og utnsitt plugin A har ressursene sine lagret for å få den rette stien.



Figur 7.5: Begrense ressurstilgang basert på plugin-navn

7.3.6 Begrense ressurstilgang med side, utsnitt og plugin-navn

En annen mulig løsning er å bruke side, utsnitt og plugin-navn som ID på hvilke plugins som får tilgang til den gitte pluginen sine ressurser. Denne løsningen er i likhet med, instansiering av objekter for hver plugin, bedre når det gjelder å avgrense, men like ille når det gjelder å utvikle og vedlikeholde presentasjonsmønstre. Med denne løsningen vil bare en plugin få tilgang til ressursene til den andre pluginen, for det kan bare være en plugin av samme klasse i samme side og utsnitt. Men man må definere stien for hver plugin man ønsker skal ha tilgang til den gitte pluginen sine ressurser. Det betyr mer å holde orden på for utvikleren av presentasjonsmønster. Det blir også veldig vanskelig å forandre et presentasjonsmønster, om man ønsker

å flytte plugins til andre sider og utsnitter. Det er for man må skifte alle stiene til den pluginen som skal ha tilgang. Eksempel 7.4 viser hvordan denne løsningen hadde blitt.

Listing 7.4: Begrensning av plugin-ressurser

```
1
2 @PluginsAllowed(allowedPlugins = {"startPage/contactView/XFormsRendrer"
3 , "calenderPage/syllabusView/XFormsPieChart"})
4 public class XFormsPlugin implements FieldPlugin {
```

7.3.7 Valg av begrensning av plugin-ressurser

Den beste løsning når man legger vekt på begrensing, hadde vært å instansiert plugins ut i fra `pluginConfig.xml`. Denne løsningen gir full kontroll over hvilke plugin som har tilgang til andre plugin sine ressurser. Man slipper og å definere lange stier som man må med løsningen som bruker side, utsnitt og plugin-navn. Begge disse løsningene har en stor svakhet, fordi det blir vanskeligere å vedlikeholde og utvikle presentasjonsmønstere. Utvikleren av presentasjoner skal ikke bruke tid på å få plugins til å fungere. Det er allerede komplisert nok å utvikle presentasjonsmønster. Valget blir å bruke plugin-navn som ID-er i annotasjonen, som sier hvilke plugins som har tilgang til den gitte plugin sine ressurser. Selv med svakheten at andre plugins av samme klasse også får tilgang til ressurser som ikke er ment for de. Kandidaten mener det er en bedre løsning fordi den krever mindre av utviklere av presentasjonsmønster og det minsker tilgangen til andre plugins sine ressurser i tilstrekkelig grad.

7.3.8 Formatet på plugins-ressurser

Det er utvikleren av plugins som bestemmer i hvilke format plugin-ressursene skal bli lagret i. Det gir utvikleren stor frihet til å lagre hva man måtte ønske, som bilder, lydspor, XML-filer og filmer. Denne friheten har sin ulemper, fordi utvikleren velger sin egen navnekonvensjon på hva filene skal hete og hvilket format de skal lagres i. Når det kommer til media-filer er det ikke mye å gjøre med, men når det er tekst-filer bør man bruke XML-formatet. Hele konseptet med DPG er gjenbruk av data. Det kravet bør og settes for ressursene til plugins. Problemstillingen har ikke blitt vurdert enda, fordi det er ikke laget plugins som samler inn store mengde dataer som kan brukes av andre deler av DPG og andre applikasjoner. Nå når XForms er integrert i DPG vil denne problemstillingen være aktuell.

Data ment for gjenbruk bør lagres i XML. Det mener kandidaten bør gjelde for andre

plugins som samler inn data med HTMl-skjemaer. De dataene som blir samlet inn vil for eksempel bli brukt av andre plugins eller deler av DPG. Det er vanskelig å forholde seg til mange forskjellige data-formater på tekstfiler. Eksempel 7.5 viser hvordan kommentar-pluginen lagrer dataene sine. De er formatert i HTML, og det fungerer bra når man vil vise dataene slik kommentar-pluginen gjør. Skal andre plugins for eksempel vise dataene på en annen måte, vil det skape unødvendig arbeid med å formatere dataene annerledes. Den pluginen som vil vise dataene annerledes, vil for eksempel ha data om når kommentaren ble skrevet og hvem som skrev den. Dette er ikke enkelt med HTML-formatet.

Listing 7.5: Kommentar-plugin sine ressurser

```
1
2 <p><strong>dpgtest</strong> skrev den 02.03.11 klokken <strong> 19:35 <↩
      /strong>
3 kommentaren:<br>Dette er en kommentar, som viser formatet til
4 kommentar-pluginen</p>
```

Blir dataene lagret i XML-formatet er det enklere for utviklere å bruke dataene om igjen. Det vil også gjøre utviklingen raskere for utviklere kan gjenbruke metoder man har for å hente ut XML-data. Eksempel 7.6 viser hvordan data kunne blitt lagret som XML.

Listing 7.6: Kommentar-plugin sine ressurser som XML

```
1
2 <pluginInput>
3   <userName>dpgtest</userName>
4   <comment>Dette er en kommentar, som viser formatet til
5     kommentar-pluginen</comment>
6   <time>19:35</time>
7   <date>02.03.11</date>
8 </pluginInput>
```

Problemet er hvordan man kan tvinge utviklere å lagre dataene de får i XML-formatet. Det er først og fremst inndata fra brukeren man ønsker å lagre som XML. Det er derfor en god løsning at kontrolleren som tar imot data fra skjemaet brukeren har fylt inn, lagrer dataene som XML. Problemet med dette igjen er hvilke navnekonvensjon man skal bruke for at plugin skal kunne hente filene. Plugin kan bare hente inn filer basert på navnet til filen. Kandidaten mener det er fornuftig å bruke *timestamps* som filnavn og lage en ny metode som plugins kan bruke for å hente ut alle XML-filene. Filene vil bli levert til pluginen som en liste, som er sortert fallende med den nyeste filen først.

Dette er ingen optimal løsning, men det persistenslaget DPG har i dag er det ikke enkelt å finne noen fornutige ID-er for å hente ut filene man ønsker. Det vil også ta lang tid når man skal hente inn alle filene og finne den filen man ønsker. Dette må gjøres hver gang pluginen kjøres på nytt, fordi den er en singleton.

7.4 Konklusjon

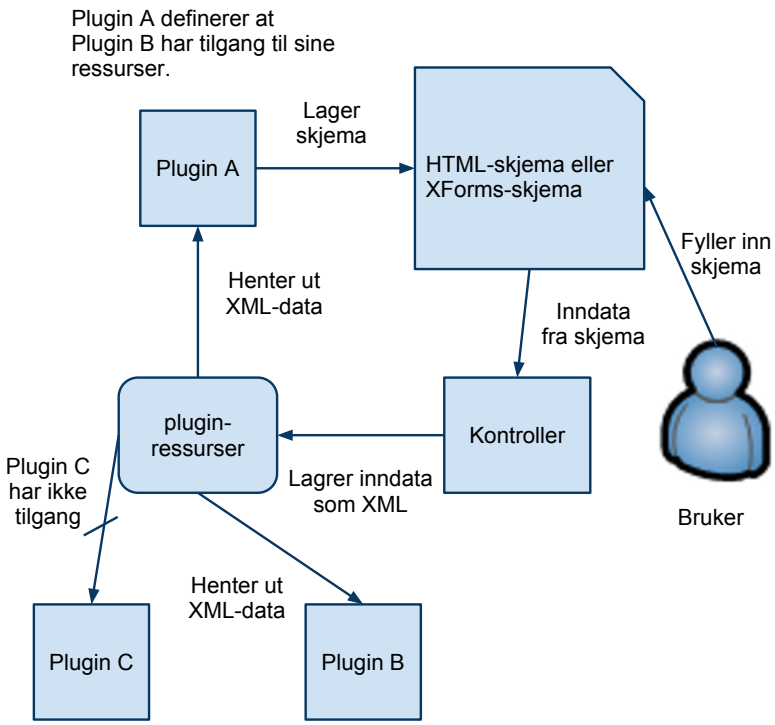
Det er utviklet en rekke forbedringer for ressurshåndteringen til plugins:

- Det er blitt laget en mappestruktur som gjør ressursene enklere å håndtere. Det gjør også begrensingen av ressurstilgang for andre plugins enklere.
- Det er satt begrensninger for ressurstilgangen for plugins. Plugins kan definere hvilke andre plugins som har tilgang til sine ressurser ved å liste opp navnet deres i en annotasjon.
- Lage en ny kontroller som bruker en REST-metode for å få tak data som er nødvendig for å sende inndata til rett plugin. Dette ble gjort for det var problematisk for XForms-skjemaer å bruke skjulte felter. Denne løsningen har ikke negativ effekt for bruk av HTML-skjemaer
- Kontrolleren lagrer inndata fra brukeren i XML-formatet i pluginen sin ressursmappe. Det gjør det enklere å gjenbruke data.

I figur 7.6 kan man se et forenklet diagram over hvordan dataflyten er mellom kontrolleren og pluginen. Pluginen lager skjemaet som blir presentert til brukeren. Brukeren fyller inn skjemaet og dataene blir sendt videre til kontrolleren. Kontrolleren vet i hvilke mappe den skal plassere ressursen, og lagrer inndataene der som XML-filer. Plugin A og B har tilgang til disse filene for Plugin A definerer ved hjelp av en annotasjon at plugin B har tilgang, men det er ikke gjort for plugin C, som ikke har tilgang. Plugin A har implisitt tilgang til sine egne ressurser.

Ulemper med disse løsningene er:

- Med den nye mappestrukturen, må andre plugins som ønsker tilgang til andre plugins sine ressurser, hardkode inn stien til ressursen eller få stien inn som parameter fra `pluginConfig.xml`.
- Plugins av samme klasse har tilgang til andre plugins sine ressurser, som ikke nødvendigvis er ment for de, se avsnitt 7.3.3.
- Å bruke *timestamps* som filnavn for de XML-filene kontrolleren lagrer er ikke optimalt. Det er vanskelig å finne noen unike ID-er for å hente ut filer til plugins.



Figur 7.6: Overordnet diagram over dataflyt mellom plugins og inndata-kontroller

Alle disse ulempene er noe som kan bli forbedret i fremtidige masteroppgaver. Kandidaten er fornøyd med å oppnå bedre begrensning for tilgang til ressursene til plugins og at det er mer oversiktlig lagring. Det er og viktig at formålet til DPG, med gjenbruk av data går igjen i hele systemet. Lagring av inndata som XML er et viktig for gjenbruk av dataene av andre ressurer, deler av DPG og eksterne systemer.

8

Evaluering, konklusjon og videre arbeid

8.1 Evaluering av mål

Det overordnede målet med oppgaven var å få støtte for datainnsamling med XForms-skjema i DPG. For å oppnå dette, ble det satt opp delmål som må oppfylles. Disse var som følger:

1. Vurdere de forskjellige mulighetene for å integrere XForms i DPG.
2. Utvikle eller tilpasse biblioteker som støtter XForm-spesifikasjonen, som kan integreres i DPG.
3. Vurdere muligheter for å motta og lagre XForms-inndata i DPG på en god og sikker måte.
4. Gjøre nødvendige endringer i DPG basert på disse vurderingene for å oppnå støtte for XForms-skjema.

Det første delmålet var å undersøke hvordan man kan integere XForms [86] i DPG. I kapittel 3 blir vurderingen gjennomgått. kandidaten vurderte muligheter for å integrere XForms i selve strukturen til DPG, lage en ny type plugin eller bruke den eksisterende plugin-arkitekturen. Det endelige valget falt på å utvikle en plugin med den eksisterende plugin-arkitekturen i DPG. Med den løsningen er det mulig å legge

til et JDOM-element, som gir muligheten å legge til XML-*markeringer* (eng. *tags*) på en side. Dette var en god og sømløs måte å integrere XForms i DPG. Datainnsamling bør lett kunne skilles fra annen funksjonalitet. Delmål 1 anses derfor som oppnådd.

Det andre delmålet var å vurdere og velge et XForms-implementasjon som kunne integreres i DPG. XForms er laget for å brukes rett i nettleseren, men det er ingen av de mest brukte nettleserene som har støtte for XForms. Det er lagt vekt på at brukeren skulle kunne bruke DPG med XForms med den nettleseren de bruker til vanlig. Det var også viktig at implementasjonen som ble valgt hadde de fleste funksjonene i XForms-spesifikasjonen implementert og testet. Noen av bibliotekene hadde funksjonaliteten, men den var utviklet med en løsning som ikke andre XForms-implementasjoner kunne bruke. Hele vurderingen er gjennomgått i kapittel 5 Det endelig valget falt på BetterForm [36]. Selve integereringen av BetterForm, med bruk av plugin-arkitekturen, blir forklart i kapittel 6. Delmål 2 anses som oppnådd.

Det tredje delmålet er å vurdere mulighetene til å motta og lagre inndata fra XForms-skjemaer med DPG. Det er en rekke svakheter med ressurshåndtering til plugins. Svakhetene og vurderte løsninger blir gjennomgått i kapittel 7. Kort fortalt er disse:

1. Kontrolleren for inndata fra HTML-skjema vil ikke kunne brukes med inndata fra XForms-skjema. Det ble laget en REST-metode som har tilsvarende funksjonalitet som den eksisterende løsningen for HTML-skjema, men den vil også kunne brukes med XForms-skjema.
2. Plugins deler samme ressursmappe. Det ble laget en ny mappestruktur som baserer seg på en sti som er slik: *side/utsnitt/pluginNavn/*. Hver plugin har derfor sin egen ressursmappe.
3. Plugins har tilgang til andre plugins sine ressurser. Det blir satt restriksjoner på metodene som henter ut ressursene. Plugins kan da bare hente ut ressurser fra plugins med samme navn og definere hvilken andre plugins som har tilgang sine ressurser. Man definerer hvilke andre plugins som har tilgang til ressursene med en annotasjon som tar inn en liste med plugin-navn.
4. Pluginutvikleren bestemmer i hvilke format data skal lagres i. Etter å vurdert de forskjellige løsningene bestemte kandidiaten at inndata fra skjema skal lagres som XML. Inndata bør lagres som XML, fordi de gjerne skal gjenbrukes av andre plugins eller applikasjoner.

Kandidaten er fornøyd med hvordan inndata nå blir håndtert i DPG. REST-metoden gjør det enkelt for XForms-skjema og HTML-skjema å bruke samme metode for å sende inn data. Løsningen med mappestrukturen til plugins-ressurser er god løsning fordi det blir enklere å håndtere ressursene og begrense tilgang til ressursene. Lagring

av inndata som XML er en viktig forbedring, som vil gjøre det mer attraktivt å utvikle plugins som bruker dataene.

Den løsningen som kandidaten ikke er helt fornøyd med er restriksjonen på tilgangen til plugins sine ressurser. Plugins med samme navn, men på forskjellig side og utsnitt, har tilgang til ressurs som de ikke skal ha tilgang til. Det var vanskelig å finne en virkelig god løsning på dette problemet. Plugins som har tilgang til andre plugins sine ressurser, burde bare hatt lesetilgang. Det er fordi plugins som bruker andre sine ressurser vil lese av disse dataene, og om de vil forandre dataene kan de lagres på nytt i denne pluginen sin mappe. Dette ble ikke utviklet for problemstillingen dukket opp for sent i oppgaven. Selv om ikke alle punktene i dette delmålet er oppnåd på en optimal måte, så er de oppnådd på en tilstrekkelig måte. Delmål 3 anses som derfor som oppnådd.

Det siste delmålet er å gjøre de nødvendige endringene som må til for å samle inn data i DPG. Dette er blitt gjort ved å bruke *Adapter*-designmønsteret [47] til å kunne integrere BetterForm i DPG. Dette ble en løsning som kandidaten er fornøyd med, fordi man integrerer BetterForm slik at man enkelt kan bruke nye versjoner av BetterForm eller skifte ut hele biblioteket. Filen `web.xml` og JavaScript-koden i hovedmalen til et presentasjonsmønster, er de eneste stedene man blander BetterForm og DPG sine instillinger. Endringene som var nødvendige for å integrere XForms-skjema i DPG er forklart i kapittel 6. Delmål fire anses for oppnådd.

Alle delmålene er oppnådd. DPG har blitt det et internettinnholdsystem, som kan tilby datainnsamling med XForms. Man kan i DPG presentere XForms-skjema, samle inn data og lagre disse. Hovedmålet anses derfor som oppnådd.

8.2 Vurdering av teknologier

I denne delen blir det gjort rede for teknologier som kandidaten føler er spennede og kan være nyttige for videreutviklingen av DPG. Det vil bli gitt en forklaring på teknologien og en subjektiv mening om den.

8.2.1 XForms

XForms blir forklart i kapittel 4. Kandidaten har ikke laget veldig avanserte XForms-skjema, men har testet ut en rekke funksjoner i DPG. Det som er bra med XForms-skjemaer er at man enkelt kan lage avanserte skjemaer og bare fokusere på dataene. Man slipper å bruke AJAX [26] for å gi feilmeldinger på klienten. Det er et klart skille mellom presentasjonen og selve innholdet. Det er også stor fordel at man kan motta

inndata i XML-formatet. XML har en fast struktur og gjør det mulig å håndtere komplekse dataer enklere.

Det som er ulempen med XForms er mangelen på støtte i de mest brukte nettleserene. Hadde man kunnet bruke XForms direkte i nettleseren ville nok mange flere brukt teknologien. Når XForms ble gjort en W3C-*anbefaling* [82] (eng. *recommendation*) ble det spådd at den ville ta over for HTML-skjemaer, men det har ikke skjedd. Når kandidaten skulle finne Xforms-implementasjoner, var det mange implementasjoner som ikke ble utviklet lenger. Det virket som det var høyest aktivitet på utvikling av XForms mellom 2003-2006. Etter det ser det ut som flere og flere har sluttet. Det var også veldig få nyetablerte implementasjoner.

Kandidaten mener XForms har mange fordeler og er en god teknologi. XForms er en åpen standard, som gjør at den bør stille sterkere med tanke på kompetabilitet med andre systemer. Det blir spennende å se om teknologien vil bli utbredt eller om den vil dø ut.

8.2.2 Trac

Trac [71] er et *problemsporingssystem* (eng. *Issue Tracking System*), og det har blitt brukt til fremdriftsplan på både masteroppgaven og utviklingen. Dette systemet gjer at man kan sette opp mål man skal nå og oppgaver for å nå det målet. Da kan man enkelt se hva som trenger å bli gjort og hva som er blitt gjort. Dette er en fordel for kandidaten og veilederen, som også kunne følge med på fremgangen mot målet. For hver møte ble det satt opp oppgaver som skulle bli gjort til neste møte. Eksempler på det kan være å skrive utkast for et kapittel på oppgaven eller implementere en komponent. Komponenten ble så delt ned i mindre oppgaver, som klasser eller funksjonalitet. Dette gjorde det enkelt å holde fokus på hva som skulle bli gjort.

Trac har også en wiki, og den ble brukt til å forklare forskjellige deler av av DPG og hvordan DPG fungerte. Wikien var til veldig god hjelp første gangen man bygget DPG. Selv om bruken av Trac er ment for utvikling av applikasjoner og for å rapportere problemer, så var det veldig nyttig å bruke den som en fremdriftsplan. Det var fordi den var enkel å oppdatere og det var motiverende å se at man nærmet seg målet for hver oppgave man gjorde.

8.2.3 Maven

DPG har brukt Ant [63] i mange år, men i fra høsten 2010 har DPG brukt Maven [5]. Maven blir brukt til prosjekthåndtering og bygging. Maven bruker *Project Object Model* (POM), en XML-fil, som man definerer avhengigheter til andre komponenter. Alle bibliotekene som prosjektet er avhengig av blir dynamisk lastet ned til et lokalt *arkiv* (eng. *repository*). Dette gjør at prosjekter kan dele biblioteker, fordi de er lagret i arkivet og ikke i hvert prosjekt. Maven bruker plugins til å utføre forskjellige handlinger. Eksempel på dette er å kjøre testing, *utrulle* (eng.*deploying*) til en server og integrering med Jenkins [33].

Sammenlignet med Ant, er Maven strengere på strukturen til prosjektet. Det må være en fast mappestruktur og alt må definerest i POM. I Ant kan man ha en friere struktur, men det kan være negativt. For alle prosjekter kan være forskjellige. Nye programmere vil da bruke tid på sette seg inn i strukturen. Maven er enkelt å bruke og det har vært en positiv opplevelse. Kandidaten mener det er enklere å bruke enn Ant. Man må bare forstå hvordan man bruker POM. Det negative med Maven, er om man trenger biblioteker som ikke finnes i Maven-arkiver. Maven-arkiver er arkiver på Internett som man laster ned avhengighetene fra. Finnes ikke bilblioteket i Maven-arkiver, så må man legge det til i det lokale-arkivet. Dette var en tidkrevende prosess.

8.2.4 Direct Web Remoting

Direct Web Remoting (DWR) [65] blir brukt i BetterForm. Hvordan DWR blir integrert i DPG, blir forklart i kapittel 6. DWR er et *Remote procedure Call* (RPC)-bibliotek. DWR gjør det mulig å kalle JavaScript-metoder fra Java-metoder og Java-metoder fra JavScript-metoder. DWR består hovedsaklig av to hoveddeler:

- En Java-sevlet som kjører på serveren som tar i mot meldinger fra nettleseren.
- JavaScript i nettleseren som sender meldinger og dynamisk oppdaterer brukergrensesnittet.

Denne teknologien kan være nyttig om man trenger å oppdatere brukergrensesnittet dynamisk gjennom en modell man har på serversiden. Serversiden vil bruke Java, men klientsiden vil bruke JavaScript. Da kan man oppdatere brukergrensesnittet uten at brukeren vil legge merke til det. Det er også mulig å sende dedikert JavaScript-kode til nettleseren fra Java. DWR var lett å sette opp og kan anbefales om man bruker Java og trenger å oppdatere nettsider asynkront. Biblioteket støtter også mange sorter strukturtyper, som for eksempel lister og strenger.

8.2.5 Git

Git [12] er et *distribuertversjonskontrollsystem* (eng. *Distributed Version Control System*). Linus Torvalds startet prosjektet, som opprinnelig var laget for Linux. Det støttes nå av de mest brukte operativsystemene. DPG bruker Subversion (SVN) [75] , som versjonskontrollsystem. Kandidaten har derfor ikke brukt Git omfattende, men oppdaget at stadig flere prosjekter har begynt å bruke dette systemet. Eksempler på det er Orbeon [60], BetterForm [36] og OpenXdata [55]. Dette kan være noe DPG er interessert i å bruke i fremtiden.

Hovedforskjellen mellom Git og andre versjonskontrollsystem er hvordan data blir behandlet. I SVN blir for eksempel informasjon den beholder basert på filer og forandringer gjort til hver enkelt fil over tid. Git ser på dataene som et *bilde* (eng. *snapshot*) av alle dataene i prosjektet. Hver gang du legger inn forandringer til versjonskontrollen, tar Git et bilde av hvordan dataene dine er. Det blir lagret en referanse til det bildet. Er det filer som ikke har forandret seg siden sist, vil ikke disse filene bli lagret igjen. Det er bare lenken til den gamle filen som blir lagret. Dette vil spare harddisk plass.

Git baserer seg på lokallagring av data. Dette gjør Git mye raskere enn andre versjonskontrollsystem. Hele versjonshistorien til Git blir lagret lokalt, så man kan jobbe og se forandringer uten å være koblet til et nettverk. Siden man ofte lagrer lokalt, vil det være en fare for at harddisken vil bli ødelagt. Dataene som ikke er overført til permanentlagring vil da være tapt. Git kan være interessant å bruke for fremtidige masterstudenter, som ønsker å ta i bruk nye teknologier.

8.3 Konklusjon

Denne masteroppgaven beskriver vurderinger og løsninger for å få DPG til å samle inn komplekse inndata med bruk av XForms. Det har vært en lærerik opplevelse, fordi DPG er et større system som det tar tid å lære seg. Det har også vært spennende å vurdere de forskjellige XForms-implementasjonene. Dokumenteringen av de forskjellige XForms-implementasjonene var varierende, mange fokuserte bare på bruk av egen web-applikasjon og lite om integrering med andre systemer. Derfor har det gått mye tid til å undersøke hvordan de forskjellige implementasjonene var uviklet, og om de hadde en struktur som fornuftig kunne integreres i DPG.

Det har vært en en ny opplevelse å jobbe i lag med andre som et *lag* (eng.*team*) på et større prosjekt. Det har vært nyttig å ha noen å diskutere problemer, arkitektur og teknologier med. Man lærer mye om hvordan man effektivt kan formidle ideér. Eksempel på det kan være bruk av grafiske fremstillinger.

I utviklingen har det blitt brukt agile-metoder med *test-drevet utvikling* (eng. *Test Driven Development*), enkleste løsning først, refaktorering og bruk av designmønster. Test-dreven utvikling har blitt praktisert hvor det er mulig. Det var vanskelig å praktisere for integreringen av BetterForm. Testing av funksjonalitet i nettleseren, som er avhengig av mange teknologier for å kunne kjøre, er ikke enkelt å teste. Det forskjellige delene er blitt testet hver for seg nøye. Bruken av test-dreven utvikling under utviklingen av nye metoder for ressurshåndteringen av plugins var svært nyttig. Man kunne se med en gang om funskjonaliteten var som forventet.

Det har vært møter nesten hver uke med veilederene, med status og fremdriftsplan. Dette har drevet utviklingen jevnt fremover og vært en stor hjelp. Bruken av et problemsporingssystem har hjulpet med å holde fokus oppgavene som skal gjørest.

DPG og konseptet med presentasjonsmønster har vært interessant å lære og jobbe med. Det har potensiale til å bli et veldig bra internettinnholdssystem. Pluginarkitekturen gir muligheten til å legge til ny og spennende funksjonalitet sømløst. Styrken til DPG er å presentere data. Kandidatens bidrag til DPG er å kunne samle inn komplekse data. Det blir derfor spennende å se om det blir utviklet noe som kan presentere disse dataene på en god måte. I den forbindelse, har kandidaten foreslått å lage en løsning som kan vise data fra et området, som blir markert på et kart. Da kan man lett se hva det gjelder og hvor dette området er. Dette blir nærmere forklart i avsnitt 8.4.

Svakheten til DPG er at det er vanskelig å utvikle nye presentasjonsmønster. Jostein Bjørge utviklet PPDev [10] i sin masteroppgave høsten 2010. Det er en webapplikasjon som gjør det enklere å utvikle nye presentasjonsmønstre, men PPDev er bare en prototype. Det er også vanskelig å lage XSLT-transformasjoner og Velocity-maler [22], som DPG er avhengig av. Det er mulig å utvikle en applikasjon som tar hånd om disse svakhetene. Noen forslag til en slik applikasjon er å finne i Bjørn C. Sebak sin masteroppgave [68]. Da ville brukere uten kunnskap om konseptene og teknologiene kunne laget nye presentasjonsmønster. Dette ville igjen føre til at DPG ville blitt mer attraktiv for flere brukere.

8.4 Videre arbeid

8.4.1 Plugin-livssyklus

I kapittel 3 blir det gjennomgått tre svakheter med pluginarkitekturen:

- Manglende livssyklus for plugins.

- `PluginManager` bør være en kontrakt i stedet for en abstrakt-klasse.
- Definering av plugins i entitetetsinstansene manuelt.

De to første punktenene vil blir gjennomgått her, det siste vil bli gjennomgått i neste avsnitt. Det er vanlig for plugin-arkitekturer å ha livssykluser. Det vil si at pluginen har forskjellige tilstander etter som pluginen kjører, stopper, installert eller avinstallert. Alle disse tilstandene skal kunne oppnås mens vertapplikasjonen kjører som normalt. Det er viktig at forskjellige koblinger som fil- og databasestrømmer blir lukket når vertapplikasjonen blir avsluttet. I DPG vil ikke plugins som har slike strømmer åpne, kunne lukke de når DPG avsluttes. Det er heller ikke mulig å stoppe og avinstallere plugins som kjører i DPG. Dette er ikke bra om man vil skifte ut en plugin, med en nyere versjon, i en applikasjon som har mange brukere. Man vil at applikasjonen skal kunne kjøre selv om man skifter ut pluginen.

Hadde DPG sine plugins hatt livssyklus ville det gjort applikasjonen mer robust. Sluttet en plugin å virke kunne man fjernet den og startet den på nytt, i stedet for å starte hele applikasjonen på nytt. Det finnes Java-rammeverk for plugins-livssyklus. Eksempler er *Open Services Gateway initiative framework* (OSGI), *Java Simple Plugin Framework* (JSPF) og *Java Plugin Framework* (JPF). OSGI er et ganske omfattende rammeverk, som kan inneholde for mye funksjoner som DPG ikke har bruk for. JSPF og JSP, ser begge ut som gode alternativer til DPG. De tilbyr den funksjonaliteten som er nødvendig. Man kan også utvilke sin egen plugin-livssyklus, men med etablerte biblioteker, som de nevnt ovenfor, finner man mer dokumentasjon og det kommer stadig nyere og bedre versjoner.

Om man velger å utvikle livssyklus for plugins i DPG vil man også lage en ny `PluginManager`. Da bør man sørge for at den er en kontrakt som DPG bruker, i stedet for en abstrakt-klasse slik den er i DPG 2.1. Dette vil gjøre det enklere å skifte ut `PluginManager` og lage nye plugins-typer. Utviklingen av livssyklus for plugins kan være en del av en masteroppgave. Det er en viktig prioritering om man ønsker å tilby støtte for mer kompliserte funksjonalitet gjennom plugins, som for eksempel støtte for datainnsamling med XForms-skjemaer. DPG 2.1 har enda ikke tatt i bruk mer avanserte plugins i fjernundervisningen, enn plugin som håndterer typer for felt. Når man skal bruke plugins i produksjonversjonen er det nyttig å kunne stoppe, avinstallere, installere og starte plugins uten at det påvirker driften av applikasjonen.

8.4.2 Pugins må defineres manuelt i entitetinstansen

Plugins må definerest manuelt i entiteinstansen, dette er en svakhet fordi dette er den eneste plassen man definere plugins i presentasjonen i stedet for i presentasjons-

mønsteret. Dette burde kunne håndteres automatisk, fordi entitetinstanser har alltid samme felter som i entiteten. Når man definerer en entitet bør det kunne lages en entitetinstans automatisk av DPG.

Dette kan gjøres når DPG starter opp, ved at man sjekker om feltene er i entiteten også er i entitetinstansen. Delsystemet Presentation Manager (PM) er forklart i kapittel 2, og har ansvaret for presentasjons administrering. PM har derfor all funksjonalitet som skal til for å kunne sjekke og legge til felter i entitetinstansen.

Dette er en programmeringsoppgave som ikke trenger å prioteres med det første, siden det bare vil gjøre det enklere for utvikleråå legge til plugins. I masteroppgavene til Jostein Bjørge og Ole Henning Vårdal [78] nevnes det at PM bør flyttes ut av DPG, svakheten kan da bli fikset i forbindelse med dette. Da må alle plugins være definerte før man bruker presentasjonen i DPG.

8.4.3 DPG for smarttelefoner

Det er blitt veldig vanlig å eie en smarttelefon i Norge. Brukere av smarttelefoner bruker de til en rekke tjenester, som for eksempel å sjekke bankkontoen, bestille billetter og sjekke været. Smarttelefoner gjør at man få nesten like mange tjenester som med en vanlig datamaskin. Måten man tilbyr tjenester til smarttelefoner er litt annerledes enn med en datamaskin. Med en datamaskin vil man hovedsaklig bruke nettleseren. Med smarttelefoner er det vanlig å tilby tjenester gjennom applikasjoner. Støtte av smarttelefoner i DPG ville vært bra for visjonen for samarbeidet mellom DPG og OpenXdata, fordi bruken av OpenXdata er hovedsaklig datainnsamling fra mobiler.

Det er en rekke metoder som kan tilby støtte for smarttelefoner i DPG:

- Utvikle applikasjoner i et programmeringsspråk smarttelefonen er kompatibel med.
- Utvikle applikasjoner gjennom Java Platform, Micro Edition (Java ME) [57].
- Lage nettsider som passer smarttelefoner sine nettlesere.
- Utvikle applikasjoner med rammeverk som støtter flere typer av smarttelefoner.

Utvikling av applikasjoner for smarttelefoner er krevende for man trenger å lage en applikasjon per operativsystem. De tre mest vanlige operativsystemene er [7], Android [4] og Windows Phone 7 [49]. Det vil si at man må utvikle og vedlikeholde tre separate applikasjoner, noe som vil koste dyrt. Man kan utvikle med Java ME,

som vil kunne brukes av smarttelefoner og andre mer enkle mobiler. Det må fremdeles utvikles en applikasjon, og man må undersøke hvordan man vil bruke den med DPG.

Det er mulig å utvide plugins-arkitekturen til å lage en ny type plugin. Dette blir forklart i kapittel 3. Da kan man utvikle en plugin som overkjører den normale prossesen med å sette sammen nettsider, med en prosses som lager nettsider tilpasset smarttelefoner. Det er også mulig å utvikle applikasjoner i rammeverk, som håndterer de forskjellige operativsystemene, slik at man trenger bare å lage en applikasjon. Det hadde vært en interessant oppgave å funne en bra løsning for DPG for mobileenheter.

Dette kan være en masteroppgave, fordi det er må gjøres en grundig vurdering for å finne den beste måten å gjøre dette på og utvikling vil være omfattende. Denne oppgaven bør prioriterest om man ønsker at DPG skal være konkurransedyktig mot andre internettinnholdssystemer. Systemer som Joomla [37] og Wordpress [91] har allerede utvidelser for å tilby denne funksjonaliteten.

8.4.4 Interaksjon med andre systemer

DPG kan nå samle inn komplekse data med bruk av XForms-skjema. Det er viktig å kunne dele disse dataene med andre systemer. Siden dataene er lagret i XML, er det enkelt for andre systemer å bruke disse dataene. Med tanke på det mulige samarbeidet med OpenXdata, er det viktig å kunne dele dataene DPG samler inn med OpenXdata.

Hvordan skal man dele disse dataene? Det er mulig å bruke web-tjenester for å la DPG dele dataene sine med andre applikasjoner. Da kan man bruke for eksempel Representational State Transfer(REST) [66] eller [85]. Begge disse løsningene er plattformuavhengige og kan enkelt brukes for å dele XML-data med andre applikasjoner.

Om man først utvikler interaksjon med andre systemer, vil det også være nyttig å ta med autorisert tilgang til eksterne ressurser og tillate uautorisert tilgang til ressurser. Disse to forslagene blir forklart i Peder Skeidsvoll sin masteroppgave [70]. Kandidaten sitt forslag og Peder Skeidsvoll sitt forslag kan sammen være en masteroppgave. Den bør prioriterest om det er nøvendig for DPG å dele data med andre applikasjoner.

8.4.5 Presentering av inndata

Når man har samlet inn data, så er det ønskelig å kunne presentere disse dataene. Det kan gjøres på en rekke måter som er avhengig av hvilke data man skal presentere. Man kan for eksempel lage grafer og diagrammer for tall, bruke kart til å presentere

geografiske data og for tekst kan man bruke en ordsky *tag cloud*. Figur 8.1 viser en ordsky av dette kapittelet. Ordene som er brukt mest er større enn ord som er brukt færre ganger. En ordsky kan være nyttig om man har mye tekst og vil se hvilke ord som går igjen for å beskrive noe.

Det finnes mange måter presentere data på, og DPG bør ha en god måte å presentere data som er blitt samlet inn. Om man utvikler støtte for smarttelefoner i DPG, så er det mulig å bruke Global Positioning System (GPS)-data [29] fra telefonen og samtidig samle inn data og bilder med bruk av XForms-skjemaer. Da kan man for eksempel dokumentere hærverk, som tagging på offentlig eiendom, og rapportere det inn med bilder og nøyaktig posisjon. Da vet de som skal fjerne taggingen nøyaktig posisjon og hvordan det ser ut.

Det finnes Java-biblioteker for å lage grafer, som for eksempel jgraph [34] og JUNG [74]. For å lage ordskyer har man biblioteker som OpenCloud [14]. For geografiske data kan man bruke Google Maps API-en [28]. Plugin-arkitekturen i DPG vil gjør det enkelt å integrere slike biblioteker.

Å utvikle en plugin som kunne vise data i kart kunne vært en del av masteroppgave, men kan kombineres med å uvikle livssyklus for plugins i DPG for å bli en fullverdig masteroppgave. Prioriteringen av denne problemstillingen må bli vurdert etter hvilket formål man vil DPG skal brukes til.

Figur 8.1: Eksempel på en ordsky av dette kapittelet

8.4.6 Håndtering av ressursene til plugins

Det er fremdeles noen svakheter med håndteringen av plugins sine ressurser. Plugins som har tilgang til andre plugins sine ressurser bør bare ha lesetilgang. I DPG kan plugins, som har tilgang til andre ressurser, kunne slette og endre ressurser. Det bør også være enklere å definere mer spesifikt hvilke plugins som har tilgang til andre plugins sine ressurser. Denne svakheten blir forklart i kapittel 7. En mulig løsning er å knytte ID-er til hver enkelt instans av plugins. Dette vil og gjort at plugins ikke er

singletons lenger. Slik det er foreslått i Tobias Olsen sin masteroppgave [54].

Jostein Bjørge [10] nevner i sin masteroppgave om å bruke validering i persistenslogikken i sammenheng med en mulig refaktorering av persistenslaget. Persistenslaget til DPG har vist seg å ha svak ytelse og høyt minneforbruk, så det bør bli gjort en fullstendig gjennomgang og forbedring av persistensdelen. Det vil også være naturlig å vurdere ressurstilgangen til plugins i DPG. Spesielt undersøke hvordan plugins skal gi ID-er til sine ressurser. Vurderingen og refaktoreringen kan utgjøre en masteroppgave, persistenslaget er en omfattende del av DPG som må vurderest og det er mye utvikling i refaktoreringen, spesielt om man ønsker å tilby validering. Denne problemstillingen bør prioriterest, fordi dette påvirker brukere, utviklere og drifting av DPG.

A

JavaScript lagt til i hovedmalen til et presentasjonsmønster

Her vises JavaScript-ene som må legges til i hovedmalen til et presentasjonsmønster for å kunne rendre XForms-skjema. Dette trenger man bare gjøre for hovedmalen i presentasjonsmønsteret. Alle presentasjoner som blir laget ut i fra dette presentasjonsmønsteret vil ha med disse JavaScript-ene.

Listing A.1: JavaScript i hovedmalen

```
1
2  <script type="text/javascript"
3    src="../getPatternResource.html?patternId=inf219_pattern&type=
4    JAVASCRIPT&fileId=/scripts/release/dojo/dojo/dojo.js">
5      var djConfig = {
6                           debugAtAllCost:false,
7                           locale:'en',
8                           isDebug:false,
9                           parseOnLoad:false
10                        };</script>
11 <script type="text/javascript"
12   src="../getPatternResource.html?patternId=inf219_pattern&type=
13   JAVASCRIPT&fileId=/scripts/release/dojo/dojo/dojo.js"></script>
14 <script type="text/javascript"
15   src="../getPatternResource.html?patternId=inf219_pattern&type=
```

```
16    JAVASCRIPT&fileId=/scripts/release/dojo/betterform/betterform.js"></script>
17  <script type="text/javascript" src="/dpg2/Flux/engine.js">  </script>
18  <script type="text/javascript" src="/dpg2/Flux/interface/Flux.js">  </script>
19  <script type="text/javascript"
20    src="/dpg2/Flux/interface/XFormsModelElement.js">  </script>
21  <script type="text/javascript" src="/dpg2/Flux/util.js">  </script>
22
23  <script type="text/javascript">
24
25     function getXFormsDOM(){
26        Flux.getXFormsDOM(document.getElementById("bfSessionKey").value,
27     function(data){
28         console.dirxml(data);
29                   }
30                     );
31              }
32
33      function getInstanceDocument(instanceId){
34           var model = dojo.query(".xfModel", dojo.doc)[0];
35            dijit.byId(dojo.attr(model, "id")).getInstanceDocument(
                instanceId,
36      function(data){
37             console.dirxml(data);
38                   });
39              }
40
41
42
43
44       var hideLoader = function(){
45            dojo.fadeOut({
46                node:"fluxProcessor",
47                duration:400,
48                onEnd: function(){
49                  dojo.style("fluxProcessor", "display", "none");
50                  dojo.style(dojo.body(),"overflow","auto");
51                      }
52                 }).play();
53              }
54
55         dojo.addOnLoad(function(){
56
57            dojo.addOnLoad(function(){
58               dojo.require("dojo.parser");
59
60                  dojo.parser.parse();
61
62                  Flux._path = dojo.attr(dojo.byId("fluxProcessor"), "
                      contextroot") + "/Flux";
```

```
63                  Flux.init( dojo.attr(dojo.byId("fluxProcessor"),"↩
                        sessionkey"),
64                     dojo.hitch(fluxProcessor,fluxProcessor.applyChanges)↩
                           );
65
66              });
67      });
68
69  </script>
```

B

Eksempel på et XForms-skjema

Dette er et XForms-skjema som er i en XHTML-fil [83]. Skjemaet samler inn informasjon for å registrere en bruker i et forum. Hvordan skjemaet ser ut når det blir rendret er vist og forklart i avsnitt 6.5. Viktige elementer som blir vist her er instansdataene som blir definert i linje 13-17, som er selve dataene som blir sendt inn til serveren. Elementet `bind` er hvor man setter begrensninger på de forskjellige feltene, de er vist fra linje 19-28. Fra linje 46-107 blir brukergrensesnittet definert, hvor man kan se de forskjellige hjelpetekstene og advarslene man får om man skriver inn ugyldig data i skjemaet. XForms-skjemaer blir forklart grundig i kapittel 4.

Listing B.1: XForms-skjema eksempel

```
1  <html xmlns="http://www.w3.org/1999/xhtml"
2    xmlns:xf="http://www.w3.org/2002/xforms"
3    xmlns:ev="http://www.w3.org/2001/xml-events"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6  <head>
7  <title>Registrering til forum</title>
8  </head>
9  <body>
10 <div id="xforms">
11 <div id="registration" class="InlineRoundBordersAlert">
12 <div style="display: none"><xf:model>
13   <xf:instance>
```

```
14      <data xmlns=""> <firstname></firstname> <lastname></lastname> <↩
            alias></alias>
15      <email></email> <birthdate /> <pass1 /> <pass2 /> <value>1</value> ↩
            <os />
16      <pc /> </data>
17    </xf:instance>
18
19    <xf:bind nodeset="alias" constraint="string-length(.) &gt; 3"
20      required="true()" />
21    <xf:bind nodeset="email" type="email" required="true()" />
22    <xf:bind nodeset="pass1" constraint="string-length(.) &gt;= 6"
23      required="true()" />
24    <xf:bind nodeset="pass2" constraint=". = ../pass1" required="true()" ↩
          />
25    <xf:bind nodeset="agreement" type="boolean" />
26    <xf:bind nodeset="birthdate" type="date" />
27    <xf:bind nodeset="value" type="integer" />
28    <xf:bind nodeset="os" relevant="/data/pc = 'ja'" />
29
30    <xf:submission id="s-send" replace="instance" resource="echo:test"
31      method="get">
32      <xf:action ev:event="xforms-submit-error">
33        <xf:message>Registrering gikk ikke. Fiks feilene</xf:message>
34      </xf:action>
35      <xf:action ev:event="xforms-submit-done">
36        <xf:message>Du har blitt registrert</xf:message>
37      </xf:action>
38    </xf:submission>
39
40
41    <xf:setfocus control="first" ev:event="xforms-ready" />
42
43
44  </xf:model></div>
45
46  <xf:group appearance="bf:verticalTable">
47    <xf:label>Meld deg inn i forumet:</xf:label>
48    <xf:input id="first" ref="firstname">
49      <xf:label>Fornavn:</xf:label>
50      <xf:hint>skriv inn fornavnet ditt</xf:hint>
51    </xf:input>
52    <xf:input ref="lastname">
53      <xf:label>Etternavn:</xf:label>
54      <xf:hint>skriv inn etternavnet ditt</xf:hint>
55    </xf:input>
56    <xf:input ref="alias">
57      <xf:label>Brukernavn:</xf:label>
58      <xf:hint>velg et brukernavn</xf:hint>
59      <xf:alert>m   innholde mer en 3 bokstaver</xf:alert>
60    </xf:input>
61    <xf:secret ref="pass1">
```

```
62      <xf:label>Passord:</xf:label>
63      <xf:alert>passordet  m   v re  minst 6 bokstaver</xf:alert>
64      <xf:hint>passordet  m   v re  minst 6 bokstaver </xf:hint>
65    </xf:secret>
66    <xf:secret ref="pass2">
67      <xf:label>Skriv inn passord  p   nytt:</xf:label>
68      <xf:hint>Skriv inn passord  p   nytt</xf:hint>
69      <xf:alert>passordet passer ikke</xf:alert>
70    </xf:secret>
71    <xf:input ref="email">
72      <xf:label>Email:</xf:label>
73      <xf:alert>er ikke en gyldig E-post adresse</xf:alert>
74      <xf:hint>skriv in en gyldig E-post adresse</xf:hint>
75    </xf:input>
76    <xf:select1 ref="pc" appearance="full">
77      <xf:label>Eier du en PC?</xf:label>
78      <xf:item>
79        <xf:label>Nei</xf:label>
80        <xf:value>nei</xf:value>
81      </xf:item>
82      <xf:item>
83        <xf:label>Ja</xf:label>
84        <xf:value>ja</xf:value>
85      </xf:item>
86    </xf:select1>
87    <xf:input ref="os">
88      <xf:label>Hvilket OS bruker du?</xf:label>
89      <xf:hint>Skriv inn OS</xf:hint>
90    </xf:input>
91    <xf:range ref="value" start="1" step="1" end="10" incremental="true">
92      <xf:label>Hvor ofte vil du sjekke<br /> forumet  p   en dag?</↩
            xf:label>
93      <xf:hint>hvor ofte sjekke forumet om dagen?</xf:hint>
94      <xf:help>hvor ofte sjekke forumet om dagen</xf:help>
95      <xf:alert>ugyldig</xf:alert>
96    </xf:range>
97    <xf:input ref="birthdate">
98      <xf:label>F dt:</xf:label>
99      <xf:alert>er ikke en gyldig dato</xf:alert>
100     <xf:hint>skriv inn gyldig dato</xf:hint>
101   </xf:input>
102   <xf:trigger appearance="triggerMiddleColumn">
103     <xf:label>Lag min konto</xf:label>
104     <xf:hint>trykk for    lage konto</xf:hint>
105     <xf:send submission="s-send" />
106   </xf:trigger>
107 </xf:group></div>
108 </div>
109 </body>
110 </html>
```

Bibliografi

[1] AJAXForms. Ajaxforms - documentation. http://ajaxforms.sourceforge.net/docs/index.html, 2006.

[2] Paul C. Kocher Alan O. Freier, Philip Karlton. The ssl protocol version 3.0. http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00, November 1996.

[3] OSGi Alliance. Osgi. http://www.osgi.org/Main/HomePage.

[4] Android. Android. http://www.android.com/.

[5] Apache. Welcome to apache maven. http://maven.apache.org/.

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

[7] Apple. ios dev center. http://developer.apple.com/devcenter/ios/index.action.

[8] Karianne Berg. Persistensproblematikk i Dynamic Presentation Generator. Master's thesis, Universitetet i Bergen, 2008.

[9] Mark Birbeck. Mixins. http://code.google.com/p/ubiquity-xforms/wiki/Mixins, 2009.

[10] Jostein Bjørge. PPDev: Et nettbasert verktøy for utvikling og validering av presentasjonsmønstre i Dynamic Presentation Generator. Master's thesis, Universitetet i Bergen, 2010.

[11] Phil Booth. formsplayer. http://www.formsplayer.com/, 2007.

[12] Scott Chacon. Git - the fast version control system. http://git-scm.com/.

[13] Stefano Debenedetti Claus Wahlers. Deng modular x-browser. http://sourceforge.net/projects/dengmx/, 2009.

[14] Open Cloud. Opencloud. http://opencloud.mcavallo.org/.

[15] Contentmanager.eu.com. What is a Content Management System, or CMS? http://www.contentmanager.eu.com/history.htm, 2008.

[16] Alain Couthures. Xsltforms. http://www.agencexml.com/xsltforms.htm, 2009.

[17] Arnaud Le Hors Dave Raggett and W3C Ian Jacobs. Html 4.01 specification. http://www.w3.org/TR/html401/, 1999.

[18] Dojo. Dojo toolkit. http://dojotoolkit.org/, 2011.

[19] Alessandro Vernet Erik Bruchez. Xml pipeline language (xpl) version 1.0 (draft). http://www.w3.org/Submission/2005/SUBM-xpl-20050411/, 2005.

[20] Ian Evans. Your first cup. http://download.oracle.com/javaee/6/firstcup/doc/gcrkq.html#gcrmb, Mars 2011.

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

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

[23] The Apache Software Foundation. Apache license, version 2.0. http://www.apache.org/licenses/LICENSE-2.0.html, Januar 2004.

[24] Martin Fowler. Inversion of control containers and the dependency injection pattern. http://martinfowler.com/articles/injection.html.

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

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

[27] Google. Google web toolkit overview. http://code.google.com/intl/no/webtoolkit/overview.html, 2010.

[28] Google. Google maps api family. http://code.google.com/intl/no/apis/maps/index.html, 2011.

[29] U.S. Government. What is gps? http://www.gps.gov/systems/gps/, 2011.

[30] Jason Hunter. Jdom. http://www.jdom.org/.

[31] Bjørn Ove Ingvaldsen. Multimedia i dynamisk presentasjons generator 2.0. Master's thesis, Universitetet i Bergen, September 2008.

[32] JAFU. JAva for FjernUndervisning. http://nettkurs.uib.no/.

[33] Jenkins. Jenkins. http://jenkins-ci.org/.

[34] jgraph. The most popular java graph visualization library. http://www.jgraph.com/jgraph.html.

[35] Lars Windauer Joern Turner. Chiba. http://chiba.sourceforge.net/index.html, 2008.

[36] Lars Windauer Joern Turner. Betterform - developer guide. http://www.betterform.de/doc/betterFormDeveloperGuide.pdf, 2010.

[37] Joomla! Joomla! - the dynamic portal engine and content management system. http://www.joomla.org, 2010.

[38] jQuery. jquery - write less, do more. http://jquery.com/, 2010.

[39] Kajaani. X-smiles 0.6 technical specifications. http://www.xsmiles.org/TechSpec/TechSpecPlain.html, 2002.

[40] John Kugelman. Formfaces. http://sourceforge.net/projects/formfaces/, 2009.

[41] Yves Lafon. Http - hypertext transfer protocol. http://www.w3.org/Protocols/, 2011.

[42] Markku Pekka Mikael Lain. XFormsDB—An XForms-Based Framework for Simplifying Web Application Developmen. Master's thesis, Aalto University, Januar 2010.

[43] Kristian Skønberg Løvik. Webucator 3.0 - Brukerhåndtering og aksesskontroll for DPG 2.0. Master's thesis, Universitet i Bergen, 2008.

[44] Dubinko M. *XForms essentials*. O'Reilly, City, 2003.

[45] John Boyer Mark Birbeck. ubiquity-xforms - xforms in web browsers and presentational ajax libraries. http://code.google.com/p/ubiquity-xforms/, 2010.

[46] Dave Marshall. Remote procedure call. http://www.cs.cf.ac.uk/Dave/C/node33.html, 1999.

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

[48] Wolfgang M. Meier. eXist - Open Source Native XML Database. http://exist.sourceforge.net/documentation.html, 2009.

[49] Microsoft. Windows phone. http://www.microsoft.com/windowsmobile/nb-no/default.mspx.

[50] Mozilla. Mozilla xforms project. http://www.mozilla.org/projects/xforms/, 2010.

[51] MozzIE. Mozzie. http://sourceforge.net/projects/mozzie/, 2010.

[52] Khalid A. Mughal. Presentation Patterns: Composing Web-based Presentations. Technical report, Universitetet i Bergen, Institutt for Informatikk, 2003.

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

[54] Tobias Rusås Olsen. Interaksjon og søk i Dynamic Presentation Generator. Master's thesis, Universitetet i Bergen, 2010.

[55] openXdata. openxdata - documentation. http://doc.openxdata.org/, 2011.

[56] Oracle. Apache shiro. http://shiro.apache.org/.

[57] Oracle. Java me and java card technology. http://www.oracle.com/technetwork/java/javame/index.html.

[58] Oracle. Javatm authentication and authorization service (jaas) reference guide. http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html.

[59] Oracle. Java servlet technology. http://www.oracle.com/technetwork/java/javaee/servlet/index.html, 2010.

[60] Orbeon. Orbeon forms wiki. http://wiki.orbeon.com/forms/, 2011.

[61] PicoForms. Picoforms. http://www.picoforms.com/wiki/index.php?title=Main_Page.

[62] Mark Pilgrim. A form of madness. http://diveintohtml5.org/forms.html.

[63] The Apache Ant Project. Apache. http://ant.apache.org/.

[64] The GNU Project. Gnu lesser general public license. http://www.gnu.org/licenses/lgpl.html, Juni 2007.

[65] Direct Web Remoting. Dwr: Easy ajax for java. http://directwebremoting.org/dwr/introduction/index.html, 2011.

[66] Alex Rodriguez. Restful web services: The basics. https://www.ibm.com/developerworks/webservices/library/ws-restful/, 2008.

[67] Øystein Lund Rolland. Integrasjon av Orbeon Forms Designer i DPG (Tittel tentativ). Master's thesis, Universitetet i Bergen, 2011.

[68] Bjørn Christian Sebak. Dynamic Presentation Generator 2.0 – Utvikling av ny dynamisk presentasjonsgenerator og presentasjonsmønsterspesifikasjon. Master's thesis, Universitetet i Bergen, 2008.

[69] Applied Testing Shane McCarron and W3C) Technology, Masayasu Ishikawa. Xhtml™ 1.1 - module-based xhtml - second edition. http://www.w3.org/TR/xhtml11/, 2010.

[70] Peder Lång Skeidsvoll. Støtte for rike klienter i DPG. Master's thesis, Universitetet i Bergen, 2010.

[71] Edgewall Software. Welcome to the trac open source project. http://trac.edgewall.org/.

[72] Spring Source. Spring security. http://static.springsource.org/spring-security/site/, 2011.

[73] Spring. Spring framework. http://www.springsource.org/, 2010.

[74] JUNG Framework Development Team. Jung - java universal network/graph framework. http://jung.sourceforge.net/.

[75] Tigris.org. Subversion. http://subversion.tigris.org/.

[76] Movable Type. This is our developer community. http://www.movabletype.org/.

[77] Pål Unanue-Zahl. Politets nettsider en verktøykasse for hackere. http://www.vg.no/teknologi/artikkel.php?artid=578201, 2009.

[78] Ole Henning Vårdal. Sikkerheitsaspekt i Dynamic Presentation Generator: Vurdering, implementasjon og strategi. Master's thesis, Universitetet i Bergen, 2010.

[79] W3C. Document Object Model. http://www.w3.org/DOM/.

[80] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/2008/REC-xml-20081126/.

[81] W3C. HTML 4.01 Specification. http://www.w3.org/TR/REC-html40/.

[82] W3C. World Wide Web Consortium. http://www.w3.org/.

[83] W3C. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). http://www.w3.org/TR/xhtml1/.

[84] W3C. Xml path language (xpath). http://www.w3.org/TR/xpath/, 1999.

[85] W3C. Soap version 1.2 part 1: Messaging framework. http://www.w3.org/TR/soap12-part1/, 2007.

[86] W3C. Xforms 1.1 test suite. http://www.w3.org/MarkUp/Forms/Test/XForms1.1/Edition1/driverPages/html/, 2008.

[87] W3C. Html5 the input element. http://www.w3.org/TR/html5/the-input-element.html, 2011.

[88] World Wide Web Consortium (W3C). XSL Transformations (XSLT) - Version 1.0. http://www.w3.org/TR/xslt, 1999.

[89] W3Counter. April 2011. http://www.w3counter.com/globalstats.php?year=2011&month=4, April 2011.

[90] Dave Winer. Xml-rpc specification. http://www.xmlrpc.com/spec, 1999.

[91] Wordpress. Wordpress > blog tool and publishing platform. http://wordpress.org/, 2010.

[92] Yahoo! Yui library. http://developer.yahoo.com/yui/, 2011.