Støtte for Geodata i Dynamic Presentation Generator

Aleksander Vatle Waage

Institutt for Informatikk
Universitetet i Bergen
Noreg

Lang Masteroppgåve
2011

Forord

Denne masteroppgåva er eit resultat av kandidatens mastergrad i informatikk ved Universitetet i Bergen.

Oppgåva undersøker moglegheitene for å presentere geodata i Dynamic Presentation Generator 2.1, som resulterer i ei vurdering og forbetring av pluginarkitekturen.

Kandidaten vil begynne med å takke sine rettleiarar Khalid A. Mughal og Torill Hamre for deira samarbeid, engasjement og råd gjennom heile masterstudiet. Ein stor takk må og rettast til Haakon Nilsen for teknisk hjelp og råd i startfasen. Kandidaten ønsker å gje ein stor takk til medstudenten Kelly Aleksander Teigland Whiteley for eit bra samarbeid i masteroppgåva og kursa tilknytta mastergraden.

Takk til Morten Høiland, Øystein Rolland, Jostein Bjørge og Aleksander Vines som alle er tilknytta JAFU. Alle desse masterstudentane har gjort kandidatens masterstudie til ei triveleg og minnerik oppleving.

Heile denne oppgåva vil kandidaten dedikere til sin far, Helge Waage, som gjekk bort i starten av kandidatens masterstudie. Til slutt vil kandidaten takke vennar og familie for all støtte og omsorg gjennom heile studietida. Ein ekstra stor takk til kandidatens sambuar, Jennifer Camilla Skarning, for all støtte og ikkje minst for korrekturlesinga av heile oppgåva.

Til sjuande og sist, ligg ansvaret for oppgåvas innhald hos kandidaten.

Aleksander Vatle Waage
Bergen, 25. Januar 2012

Innhald

Figurar

Tabellar

Eksemplar

Forkortingar

API:	Application Programming Interface [51]
CSS:	Cascading Style Sheets [52]
DAO:	Data Access Object [53]
DPG:	Dynamic Presentation Generator [39]
HTML:	Hypertext Markup Language [48]
JAFU:	JAva for FjernUndervisning [27]
JAR:	Java Archive [54]
JDOM:	Java Document Object Model [24]
JPA:	Java Persistence API [55]
PCE:	Presentation Content Editor [39]
PM:	Presentation Manager [39]
PV:	Presentation Viewer [39]
SIC:	Satellite Imaging Corporation [8]
UiB:	Universitetet i Bergen [47]
UML:	Unified Modeling Language [23]
URI:	Uniform Resource Identifier [56]

1 Innleiing

Java fjernundervisning (JAFU) er eit nettbasert fjernundervisningstilbud som Institutt for Informatikk ved Universitetet i Bergen (UiB) tilbyr [27]. JAFU har drive dette tilbodet siden 1999. Det er to fag i dette tilbodet, "INF-100F - Grunnkurs i programmering" og "INF-101F - Videregående programmering"

Fjernundervisninga blei gjennomført ved at forelesingar, notat, oppgåver og anna kursinformasjon blei publisert i DPG sitt kursmønster. Då det begynte å bli tungvindt å publisere denne informasjonen i statiske nettsider, blei det utvikla ei dynamisk løysing.

Den første løysinga som brukte dynamiske presentasjonsmønster blei utvikla av masterstudenten Kevin Chruickshank [9]. Denne løysinga er kalla Java Presentation Generator (JPG) og blei nytta i fjernundervisninga. Konseptet presentasjonsmønster er beskreve av Khalid A. Mughal i artikkelen "Presentation Patterns: Composing Web-based Presentations" [32].

I 2005 utvikla masterstudenten Yngve Espelid [10] den første *Dynamic Presentation Generator* (DPG 1.0) som bygger på det same konseptet som JPG. Men her og var det eit stort forbetringspotensial.

Derfor bestemte masterstudentane Karianne Berg [4], Bjørn C. Sebak [39] og Bjørn O. Ingvaldsen [26] seg for å vidareutvikle DPG til DPG 2.0. Denne versjonen bygger på det same prinsippet som førre, men den har fleire modular samt ny presentasjons-

spesifikasjon. Samtidig tok dei i bruk fleire kjente teknologiar og gav DPG ei mykje bedre testdekning.

Vidare utvikling av DPG blei gjort av Peder Skeidsvoll [41] og Tobias Olsen [34] som er utviklarane bak DPG 2.1. I denne versjonen er alle basistypar som `string` og `date` handtert i ein eigen modul i systemet. Det er denne versjonen oppgåva bygger på og kandidaten vil vidare referere til denne versjonen ved forkortelsen DPG.

1.1 Mål for oppgåva

1.1.1 Overordna mål

Kandidatens overordna mål for oppgåva er å implementere ein dynamisk kartfunksjon i DPG. Med dette meines at ein skal kunne bruke eit kart til å presentere, legge til og endre innhald.

1.1.2 Delmål

For å kunne nå det overordna målet må følgjande delmål være oppnådd:

- Vurdere pluginarkitekturen
  - Er dagens løysing tilrettelagt for kartimplementasjon?
- Forbedre pluginarkitekturen
  - Gjere endringar slik at kartfunksjonaliteten kan implementeres.
- Vurdere kart-tjenester
  - Vurdere 3 API-er og vurdere kva som passar best til dette formålet, samt denne implementasjonen.
- Presentere Geodata
  - Implementere kartet frå valgt kartteneste samt handtere innhaldet som skal ligge i kartet.
- Innlegging av Geodata
  - Gje kartet funksjonar som å kommentere og karaktersette markørar.

1.1.3 Motivasjon

Det har blitt meir og meir vanleg å presentere geodata på nettstader, og dei tilgjengelege kart API-ene er veldig gode. I DPG hadde det vore nyttig å kunne presentere slik data ved bruk av kart.

For at denne typen presentasjon skal vere mogleg, kan det vere ein idé å sjå på DPG, og om nødvendig gjere om på løysinga til å innehalde den nødvendige funksjonaliteten. Viss denne løysinga kan gjennomføres og generaliseras, vil den gje store fordelar for alle utviklarar som ønsker å presentere geodata.

1.2 Utviklingverktøy

I utviklinga av DPG har kandidaten brukte utviklingsverktøyet *Eclipse* [17]. For å kjøre DPG lokalt har det blitt brukt *Apache Tomcat* [40] og *Jetty* [28]. Sistnevnte er ein plugin i Eclipse som i samarbeid med byggeverktøyet *Apache Maven* [1] bygger DPG gjennom Eclipse fortløpande etter lagring av endringar i koden.

Til avlusningsverktøy (Eng. *debugging tool*) har kandidaten brukt nettlesarane *Google Chrome* (med funksjonen "Sjå kildekode") [20] og *Mozilla Firefox* [13] (med utvidinga *Firebug* [11]).

For å halde kontroll på utviklinga har det blitt brukt Trac [42] som planlegging- og framdriftsverktøy. I tillegg er heile utviklinga versjonert i versjoneringsverktøyet SVN [15].

Det er brukt designverktøy som Google Drawings [21] og Microsoft Visio [6] for å lage figurar og modellar. Oppgåva er skrive i LaTeX [36].

1.3 Utviklingsmetode

Under heile utviklingsprosessen har kandidaten følgt metodar frå smidig utvikling (Eng. *Agile development*) . Hovedsakleg er metodar og prinsipp henta frå eXtreme Programming (XP) [30], men det er og tatt hensyn til Martin Fowler's heurestikker og retningslinjer som er beskreve i boka *Clean Code - A Handbook of Agile Software Craftsmanship* [31].

Dei mest sentrale smidige metodane kandidaten har tatt i bruk er: par programmering (Eng. *pair programming*), refaktorering (Eng. *refactoring*), kollektiv kodeeierskap (Eng. *collectiv code ownership*) og moduldesign (Eng. *modular design*). Kandi-

daten fant det naturleg å ta i bruk desse metodane fordi dei sikrar god kvalitet på kjeldekoden, forenklar utviklinga og gjer det enklare å sette seg inn i systemet for dei neste utviklarane.

1.4 Organisering av oppgåva

Oppgåva er delt opp i seks forskjellige kapittel, som tar for seg kvart sitt hovedtema.

Kapittel 2: Problembeskriving av tilrettelegging av Geodata i DPG Dette kapittelet gjennomgår dei viktigaste komponentane i DPG, samt konsepta bak DPG som eit innhaldshandteringssystem. Det blir og gjennomgått dei teknologiane DPG bruker. Dette er informasjon som er viktig for at lesaren enklare skal forstå det som blir diskutert vidare i oppgåva. Vidare blir problembeskrivinga og forskningspørsmåla presentert.

Kapittel 3: Vurdering av pluginarkitekturen Dette kapittelet er eit samarbeid mellom kandidaten og Kelly Alexander Teigland Whiteley [50].Kapittelet omhandlar kandidatanes vurdering av det noverande systemets pluginarkitektur og listehandtering. Kapittelet er skrive kvar for seg.

Kapittel 4: Forbetring av pluginarkitekturen Dette er og eit samarbeidskapittel mellom kandidaten og Kelly Alexander Teigland Whiteley [50]. Her vil kandidatane gå gjennom anbefalingar og endringar i pluginarkitekturen basert på funn i kapittel 3. Dette kapittelet er også skrive kvar for seg.

Kapittel 5: Geodata i DPG Kandidaten vil i dette kapittelet gjennomgå si problemstilling på området rundt implementasjon av Geodata i DPG, samt si endelege løysing.

Kapittel 6: Evaluering, konklusjon og vidare arbeid I det siste kapittelet i oppgåva vil kandidaten presentere ein konklusjon samt ei vurdering av måla for oppgåva. Det vil deretter bli presentert ei vurdering på vidare arbeid samt teknologiar som kandidaten meiner kan vere nyttige i vidare utvikling av DPG.

2

Problembeskriving av tilrettelegging av Geodata i DPG

2.1 Innhaldshandteringssystem

Eit *innhaldshandteringssystem* (Eng. *Content Managment System*) er eit system som handterer fleire typar data. Med data meiner ein alt fra tekst til mulitmediatypar som bilde, lyd og video. Data som behandlast av systemet blir kontrollert og strukturert i eit sentrallager.

Innhaldshandteringssystem som i all hovudsak er laga for å presentere innhaldet via ein nettstad blir kalla *internettinnhaldshandteringssystem*. Målet med slike system er å gjere det mogleg for brukare å administrere eit slikt system utan at dei treng kunnskapen rundt teknologiane systemet brukar. Eksempel på teknologiar som blir brukt er Hypertext Markup Language (HTML) [48] og Cascade Style Sheets (CSS) [52], desse blir brukt for å presentere innhald til ein nettlesar. Men systemet gir brukaren eit grensesnitt som gjer det mogleg å handtere innhald utan å måtte bry seg om slike teknologiar. Ettersom blogging er blitt ein heit aktivitet i samfunnet har det dukka opp fleire slike system den siste tida. Dei mest populære systema er Wordpress [57], Joomla [46] og Drupal [2].

DPG er eit internettinnhaldshandteringssystem. Ein viktig del av eit slikt system er å gje innhaldet ein logisk struktur. Dette gjer eit intrenettinnhaldshandteringssystem

Figur 2.1: Instansiering av presentasjonar

til eit fleksibel presentasjonverktøy. I dette kapittelet blir det gjennomgått ei rekke av dei viktigaste konsepta DPG er bygd på.

2.2 Presentasjonsmønster

I 2003 presenterte Khalid A. Mughal konseptet *presentasjonsmønster* [32]. Eit slikt mønster bygger på at innhald og struktur er separerte. Dette gjer at ein kan forandre presentasjonene av innhaldet utan å endre innhald, og motsatt.

Eit eksempel på kvar slik funksjonalitet gjer store fordelar er nettaviser. Ei nettavis inneheld mange artiklar. Artiklane har stort sett same struktur, men forskjellig innhald. Det vil vere ein fordel om ein kan bruke den same strukturen, i staden for å definere strukturen for kvar oppretta artikkel. Sjå figur 2.1.

2.3 Presentasjonsmønsterspesifikasjonen

Presentasjonsmønsterspesifikasjonen heldt eit regelsett som artikkelmønsteret brukar. Regelsettet seier kva for nokre element som skal nyttast, og kva syntaksen på desse skal vere. Eit presentasjonsmønster blir laga av mønsterspesifikasjonen, og ein presentasjon instansierast av presentasjonsmønsteret.

Figur 2.1 viser at `Artikkelmønster` lages ut fra presentasjonsmønsterspesifikasjonen, og at artiklar er instansar av `Artikkelmønster`. Det kan naturlegvis opprettast så mange instansar av `Artikkelmønster` som ønskelig. Ein ser då at kvar instans av `Artikkelmønster` nyttar same presentasjon, uavhengig av type artikkel og innhald.

2.3.1 Oppbygginga av eit presentasjonsmønster

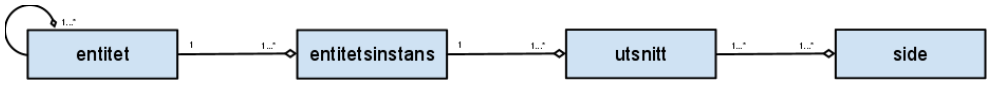
I fila `pattern.xml` blir presentasjonsmønsteret definert. Denne fila har fire hovudelement presentasjonsmønsteret bygger på:

- *entitet* (Eng.*entity*)
- *entitetsinstans* (Eng. *entity-instance*)
- *utsnitt* (Eng. *view*)
- *side* (Eng. *page*)

Grunnelementet er ein entitet, det kan for eksempel vere ein streng. Denne entiteten har *type* atributtar som fortell kva plugin som skal handtere innhaldet.

Figur 2.2 viser korleis dei overnemnde fire elementa heng saman. Ein entitet kan referere til andre entitetar og ein entitetsinstans kan, som namnet tilseier, vere ein instans av ein entitet. Eit utsnitt kan då referere til ein entitetsinstans. Vidare kan eit utsnitt referere til ein entitetsinstans og ei side til fleire utsnitt. Det er viktig å merke seg at ein entitetsinstans og kan ha fleire utsnitt, utan at innhaldet i entiteten vert endra på nokon måte. Sjølve oppbygginga av `pattern.xml` vil bli nærare gjennomgått i kapittel 3 og 4.

Kandidatane skal vurdere om det er hensiktsmessig å endre/utvide presentasjonsmønsterspesifikasjonen i DPG. Dette for å tilrettelegge for større mengder data. En eventuell forbedring skal da presenteres.



Figur 2.2: Forholdet mellom element i pattern.xml

2.4 Dynamic Presentation Generator 2.1

Dynamic presentation Generator (DPG) er ein implementasjon av det overnemde presentasjonsmønsteret og er eit såkalla internettinnhaldshandteringsystem. Den er tilknytta til JAFU-prosjektet ved Institutt for Informatikk på Universitetet i Bergen, og studentar ved dette prosjektet har utvikla DPG fram til DPG 2.1 saman. Den byggjer på Spring rammeverket [44] og benytter seg i dag av Spring MVC [43] og Spring Security [45]. Alt innhald er lagra i XML-filar og XSLT [49] står for transformasjonen av dette. Velocity [16] blir brukt for å komponere utsnittet.

DPG er bygd opp på fire hovudkomponentar; *Presentation Content Editor*, *Presentation Viewer*, *Presentation Manager* og *Lobby*. Vidare vil desse samt eit sidesystem for autentisering, kalla *Webucator*, bli gjennomgått.

2.4.1 Presentation Content Editor

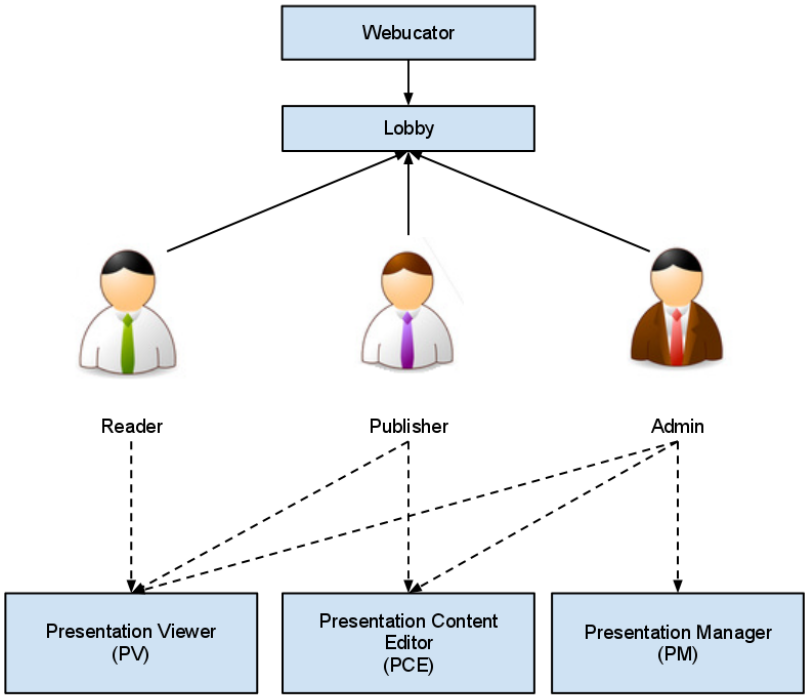
Presentation Content Editor (PCE) er ansvarleg for handtering av data. Den bruker presentasjonsmønsterspesifikasjonen for å finne ut kva informasjon som skal lagrast, og brukare med *admin*- og *publisher*-roller kan gå inn i PCE for å legge til bilder, filer og tekst i presentasjonen.

2.4.2 Presentation Viewer

Presentation Viewer (PV) er det delsystemet som tar seg av transformeringa av XML til HTML. Transformeringa skjer ved bruk av XSLT og Velocity-maler. Dette er det systemet som rendrer alt innhald til alle brukarroller.

2.4.3 Presentation Manager

Presentation Manager (PM) er det berre brukaren med "Admin" rolla som har tilgang til. Dette systemet brukast til å redigere, slette og opprette presentasjonar.



Figur 2.3: Oversikt over brukerollene og rettighetar i DPG

Opprettelsen skjer då ved eksisterende presentasjonsmønstre.

2.4.4 Lobby og Webucator

Lobby er det delsystemet som tar seg av innlogging og handtering av rettigheiter til dei forskjellige brukarrollene. Lobby kommuniserer med Webucator som er eit eige system som tar seg av brukahandteringen i DPG. Lobby komuniserer med Webucator og gir dei forskjellige brukarollene sine respektive tilganger. Som figur 2.3 viser vil ein admin-brukar få tilgang til PM, PV og PCE, i motsetning til ein publisher som berre har tilgang til PV og PCE. Og tilslutt reader som berre har tilgang til PV.

2.4.5 Pluginarkitekturen

DPG har ein pluginarkitektur som gjer det mogleg å tilføre ny funksjonalitet på ein enkel måte. Både plugins og arkitekturen blir gjennomgått i kapittel 3. Siden DPG

er veldig sentrert rundt pluginarkitekturen er det naturleg for kandidatane å vurdere dagens løysing, og forbetre denne, samt tilrettelegge for ny funksjonalitet.

2.5 Forskningsspørsmål

I samanheng med utviklinga av DPG har kandidaten og Kelly Alexander Teigland Whiteley [50] formulert tre forskningsspørsmål. Kandidatane meiner dette er element som bør vere vurdert før eit eventuelt val av implementasjon. Og gjennom heile kapittel 4 blir alle val tatt med hensyn på desse spørsmåla.

Kva er problemet med dagens pluginløysing? Kandidatane vil kartlegge problema DPG har med pluginløysinga si i dag, og finne alternative løysingar som vil betre systemet. Kartlegginga av desse problema og løysinga vil kunne hjelpe andre å finne meir effektive løysingar i tilsvarande situasjoner og system.

Korleis kan ein oppdatere/vidareutvikle dagens pluginløysing utan at det gjer for stort utslag for dei andre delane av systemet? DPG er delt opp på ein slik måte at alle plugins bruker ein felles kontrakt (Eng. *Interface*). Kandidatane vil derfor fokusere på å utvide kontraktene fremfor å endre dei. Dette vil halde systemet bakoverkompatibelt. Det er viktig at kandidatane tenker på heile systemet i si løysing, då det vil vere vanskeleg å endre på denne seinare uten at det resulterar i å knekka DPG.

Korleis kan ein designe et API i DPG? Det er viktig at DPG har ein bra og oversiktleg API, spesielt med tanke på dei som skal utvikle den vidare i framtida. Det er og viktig for effektivitet, portabilitet og smidighet. Det er og viktig at ein set seg inn i korleis andre har designet sin API i lignande system, og eventuelle problem dei støtte på. API-design i DPG er ein omfattende oppgåve, og eit resultat vil fremme ideer som vil gjere designet best mogleg.

2.6 Korleis tilrettelegge for Geodata i DPG

DPG har no moglegheiter til å vise eit statisk kart i sitt mønster. Støtte for dynamisk kart, samt at interaksjon i kart finst ikkje. Fleire og fleire leverandørar av dynamiske karttenester har komme til over kort tid. Eit behov for å kunne presentere informasjon, samt legge inn informasjon via kart er blitt veldig vanleg.

2.6.1 Velge rett kartteneste

Ei vurdering av karttenester er ei naturleg oppgåve når det finst så mange gode alternativ. Det stiller krav til kva som trengs og kva oppgåver som skal bli løyst. Dette er ei viktig vurdering.

2.6.2 Presentasjon av Geodata

Som tidlegare nemnd har DPG til no berre ein enkel presentasjon av geodata i eit statisk kart. Svakheiter i DPG har gjort det vanskeleg å utvide kartpluginen, men no som dette blir vurdert er det mogleg å utvide pluginens funksjonalitet og.

2.6.3 Innlegging av Geodata

Eit naturleg steg vidare er å sjå på muligheiter for å legge inn funksjonalitet for brukerinnlesning for roller som *Reader*. Ei vurdering på korleis dette skal handterast vert eit fokus her og.

3

Vurdering av pluginarkitekturen og listehandtering i DPG

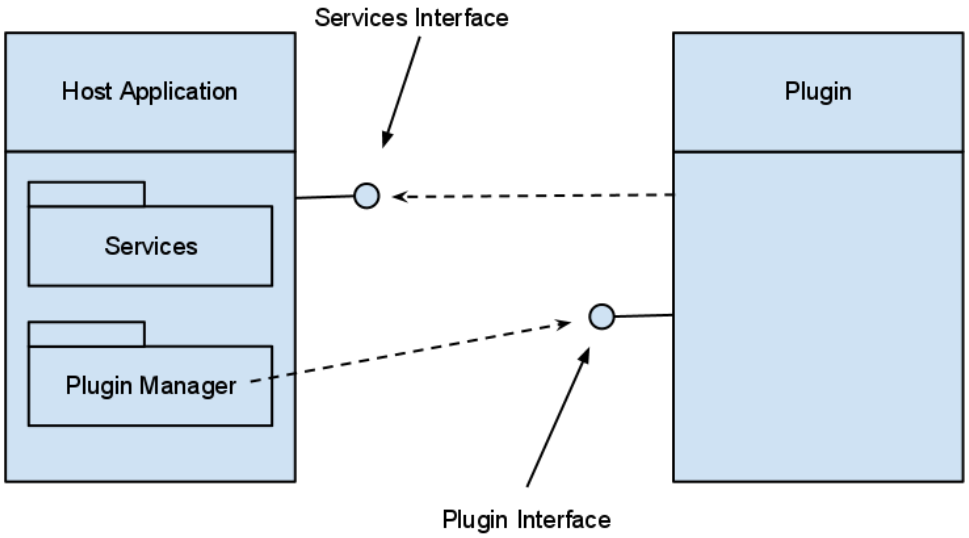
Den noverande pluginarkitekturen i DPG har klare svakheiter slik den er i dag. I dette kapittelet vil det bli presentert ei evaluering som vil setje fokus på desse svakheitene.

Kapittelet er eit samarbeid mellom kandidaten og Kelly Whiteley [50]. Kapitla er skrive kvar for seg, men er eit naturleg samarbeid då begge kandidatane ser store fordelar ved å forbetre dagens løysing.

3.1 Generelt om pluginarkitekturen

Pluginarkitekturen blei først utvikla i Bjørn Ove Ingvaldsens masteroppgåve [26], her med eit hovudfokus på multimediahandtering i DPG. Denne arkitekturen har så blitt eit fokus gjennom fleire masteroppgåver sidan. Arkitekturen blei så vidareutvikla av Tobias Olsen [34] og Peder Skeidsvoll [41]. Arkitekturen har blitt brukt i Morten Høyland [25] og Øystein Rolland [38] sine masteroppgåver i deira utvikling av X-Forms støtte i DPG.

Den noverande arkitekturen er basert på Martin Fowler sitt designmønster [19], som igjen omfattar prinsipp som Open Closed Principle [29]. Ved å følgje dette mønsteret brukar ein *kontrakt* (Eng. *interface*) som gir plugins funksjonalitet. Open-Closed



Figur 3.1: Pluginarkitekturen

prinsippet blir ein naturleg del av dette, då ein med dette prinsippet støtter opp om det å tilføre systemet nye funksjonar i kontrakten i staden for å implementere ny funksjonalitet ved å modifisere på kjeldekoden. Dette betyr at ein bør unngå forandringar på kontrakten, då det heilt klart kan øydeleggje nødvendig funksjonalitet i DPG. Dess meir funksjonalitet/plugins som blir implementert, dess meir kan bli øydelagt. I det noverande systemet er det ein slik kontrakt; `FieldPlugin`.

Som du ser i figur 3.1, er det ein stor fordel med denne løysinga at utviklaren av plugins ikkje trenger store kunnskapar om korleis DPG heng saman, då plugins berre skal forhalde seg til DPG sin plugin-kontrakt.

3.2 Entitet felttypar (Eng. *Entity Field Types*)

I overgongen mellom DPG 2.0 og 2.1 var hovudfokuset satt på korleis DPG handterte plugins. DPG 2.0 handterte enkle typar som "`string`" sjølv, plugins blei brukt for å handtere multimedia. I overgongen til DPG 2.1 blei plugins mykje meir sentral i DPG. No blir alt fra dei enklaste typane, til avanserte multimedia-typar handtert gjennom ein plugins. Med dette meines at alle *felt* (Eng. *field*) har ein felttype (Eng. *field type*) som definerer kva plugin som skal handtere dette. Felttypane har ein

respektiv plugin med same namn, til eksempel `string` felttypen har ein `string`-plugin. I overgangen blei ein altså så avhengig av plugins at DPG no ikkje fungerer utan.

3.3 Plugin-kontrakten

Plugin får sin funksjonalitet gjennom DPG sin plugin-kontrakt, `AbstractPlugin`. Dette gjer ei laus kopling mellom plugins og DPG. Metoden `generateElement()` sin funksjonalitet er å generere elementet av feltet. Elementet er ein HTML-klar representasjon av feltet og blir returnert for å bli presentert i ein nettlesar.

I motsetning til DPG 2.0, der ein plugin kunne implementere ein av to pluginkontrakter; `SingleInputFieldPlugin` og `EntityInputFieldInputPlugin` er det berre ein pluginkontrakt i DPG 2.1: `EntityInputFieldPlugin`. Denne kontrakten er i utgangspunktet tilsvarande DPG 2.0 sin `SingleInputFieldPlugin`. Det betyr at ein ikkje har moglegheit for at DPG 2.1 kan handtere fleire felt på ein gong. Denne endringa vil bli diskutert seinare i oppgåva.

3.4 Lage ein plugin

Når ein skal lage ein plugin i det noverande systemet, kan ein gjera det på forskjellige måtar:

- Spesifisere namnet, plugin og parametrar i fila `pluginconfig.xml`.
- Spesifisere namnet til pluginen med Java-annotasjon, slik eksempel 3.1 viser på linje 1.
- Berre la pluginen implementere kontrakten og la systemet hente namnet fra DPG sin filbane.

Eksempel 3.1: Definering plugin sitt namn med Java-annotasjon.

```
1  public class StringPlugin extends AbstractPlugin {
2    ...
```

Dersom ingen annotasjon blir funne, vil namnet på pluginane bli klassenamnet, så lenge den implementerer `FieldPlugin`-kontrakten. Den einaste måten ein kan spe-

sifisere parametrar for ein plugin er gjennom `pluginconfig.xml` fila. Eksempel 3.2 viser på linje 5 - 10 korleis ein spesifiserer dette.

Eksempel 3.2: Korleis ein kan spesifisere parametrar for ein plugin

```
1  <plugin-configurations>
2
3  ...
4
5  <plugin-config id="poll" plugin-name="PollPlugin">
6      <param name="pollName">poll_aske</param>
7      <param name="pollHeader">Frykter du asken?</param>
8      <param name="pollChoices">Ja|Nei</param>
9      <param name="pollChoicesVerbose">Ja, jeg frykter asken|Nei, det har↩
           ikke p virket meg.</param>
10 </plugin-config>
11
12 ...
13
14 </plugin-configurations>
```

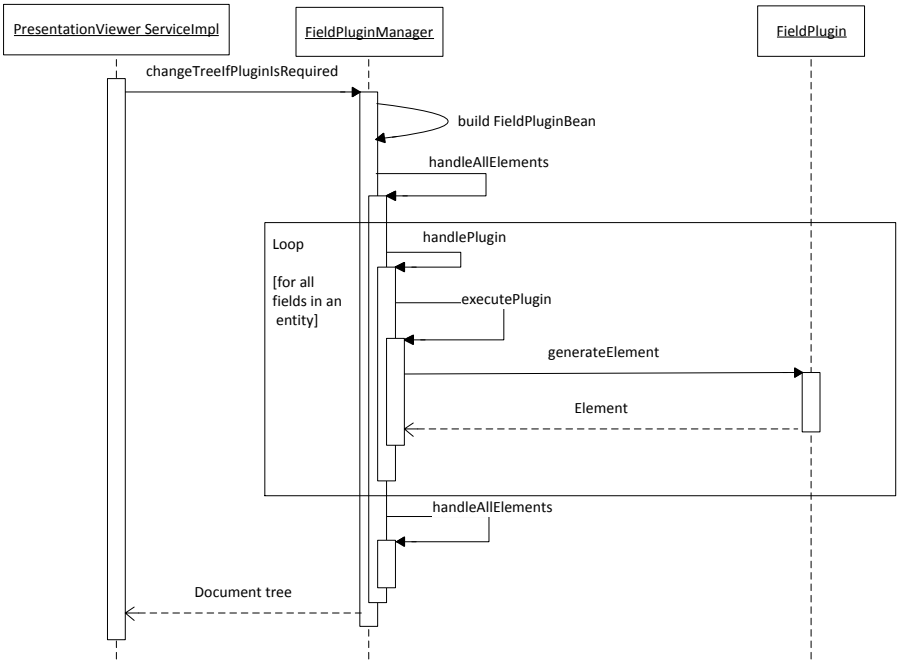
Blir ein plugin definert på ein av dei tre overnemnte måtane, vil dei bli lasta inn til plugin-manageren gjennom `ClasspathPluginLoader` bønna (Eng. *bean*). Den implementerer `PluginLoader` kontrakten. Det er også mulig å laste inn plugins i JAR-fil fra ei mappe, då brukast `JarPluginLoader` klassen.

3.5 Plugin-manageren

Plugin-manangeren har ei sentral rolle i den noverande plugin-arkitekturen. Plugin-managerens oppgåve er å handtere kvart felt, og sende dette til sine respektive plugins. Som figur 3.2 viser blir dette handtert ved at plugin-manageren bruker sin hovudmetode `changeTreeIfPluginIsRequired()`. Denne metoden tar inn heile dokumentet frå PCE for deretter å rekursivt gå gjennom kvart felt i dette dokumentet. Kvart felt blir så sendt til sin respektive plugin, og derfra returnert i ferdig formatert HTML-kode. Når heile dokumentet er konvertert til eit dokument me HTML-klare felt, blir det sendt til PV og er klar til bruk i XSL-transformasjonar.

3.6 Svakheiter i noverande pluginarkitektur

Sidan målet med oppgåva er å implementer støtte for geodat i DPG blei det gjort eit forsøk på å implementere eit dynamisk Google Maps kart gjennom ein plugin.



Figur 3.2: Metoden `changeTreeIfPluginIsRequired()` i Plugin-manageren

Gjennom dette forsøket fant kandidatane ein del svakheiter i den noverande pluginarkitekturen. Dette gjorde utslaget for at kandidatane valde å sjå nærare på denne delen av systemet.

Kandidatane vil bruke pluginen (`dynamicMapPlugin`) som eit eksempel vidare i dette kapittelet. Denne pluginen bruker ei enkel implementasjon av Google Maps [22].

3.6.1 Fleirfeltsplugin (Eng. *Multiple Field Input Plugin*)

Som tidlegare nemnt blei funksjonaliteten som gav plugins moglegheit for å handtere fleire felt tatt vekk i forbindelse med overgongen til DPG 2.1.

I utviklinga av ein `dynamicMap` plugin dukka det opp eit behov som den noverande kartpluginen (`mapPlugin`) ikkje dekker, nemleg å kunne handtere fleire felt i ein plugin. Pluginen skal generere ein JavaScript-kode [12] som skal vise eit dynamisk kart i nettlesaren. Problemet er at dette kartet skal innehalde fleire markørar med følgjande informasjon:

- Namnet på markøren
- Breiddegraden
- Lengdegraden

Dette fungerer ikkje no som det kan takast inn berre eitt og eitt felt(markør). Ein kan generere ein JavaScript-kode med ein markør, slik som den noverande pluginen og kan.

Ei løysing kan vere å la pluginen handtere eitt og eitt felt. Eksempel 3.3 på linje 12 - 21 viser korleis ein då hente alle desse felta når ein treng dei i Javascriptet som XSLT heldt. Så blir dette sett saman til eit ferdig JavaScript, klar for presentasjon. Men då må utviklaren som bruker denne pluginen og setje inn dette Javascriptet for kvar gong det blir brukt. Dette motarbeider meininga med ein plugin, nemleg at det ikkje skal være nødvendig å kunne mykje om sjølve pluginen, berre kontrakten. Dette er hovudmålet til kandidatane og. Ei løysing på dette problemet vil bli diskutert i kapittel 4.

Eksempel 3.3: Måten å generere eit dynamisk kart utan forandring i pluginarkitekturen.

```
1 ...
2
```

```
3  <xsl:template match="testDynamicMap">
4  <div class="content">
5    ...
6
7    <script type="text/javascript">
8      ...
9
10   </script>
11
12   <xsl:for-each select="marker">
13     <script type="text/javascript">
14
15       name = ']]><xsl:element name="name"><![CDATA[';
16       latitude = ']]><xsl:value-of select="longitude" /><![CDATA[';
17       longitude = ']]><xsl:value-of select="longitude" /><![CDATA[';
18       point = new google.maps.LatLng(parseFloat(latitude),parseFloat(↩
             longitude));
19
20     </script>
21   </xsl:for-each>
22   ...
23
24   <script type="text/javascript">
25
26     marker = new google.maps.Marker({
27     map: map,
28     position: point,
29
30     icon: 'http://labs.google.com/ridefinder/images/mm_20_red.png',↩
           shadow:  'http://labs.google.com/ridefinder/images/mm_20_shadow.↩
           png'});
31     ...
32
33   </script>
34   ...
35
36 </div>
37 </xsl:template>
38 ...
```

3.6.2 Handtere fleire entitetsinstanser (Eng. *Entity Instance*) igjennom Plugin Manageren

Kvar entitetsinstans peiker til ei visning (Eng. *view*). Kvar entitetsinstans i DPG har eit eige dokument. Det er dette dokumentet som blir sendt til Plugin Manageren for å rendres til presenterbar HTML-kode.

Kartplugins som `dynamicMapPlugin`, er ein representasjon av ei samling markø-

rar. Desse samlingane vil bli representert i kvar sine entitetsinstansar, som igjen betyr at ein kan lage forskjellige samlingar. Eit eksempel kan være ei samling kalla *Restauranter* og ei *Barer*. Desse kan då presenterast i kvart sitt kart. Her burde det og være mogleg å kombinera desse samlingane i eitt kart. Eit kart som har oversikta over både barar og restaurantar vil ikkje være mogleg med dagens system (DPG 2.1) eller presentasjonsmønster, fordi det ikkje er mogleg å vise to instansar i eitt og same utsnitt. Ei løysing vil bli diskutert kapittel 4.

3.6.3 Lister og sub-entiteter

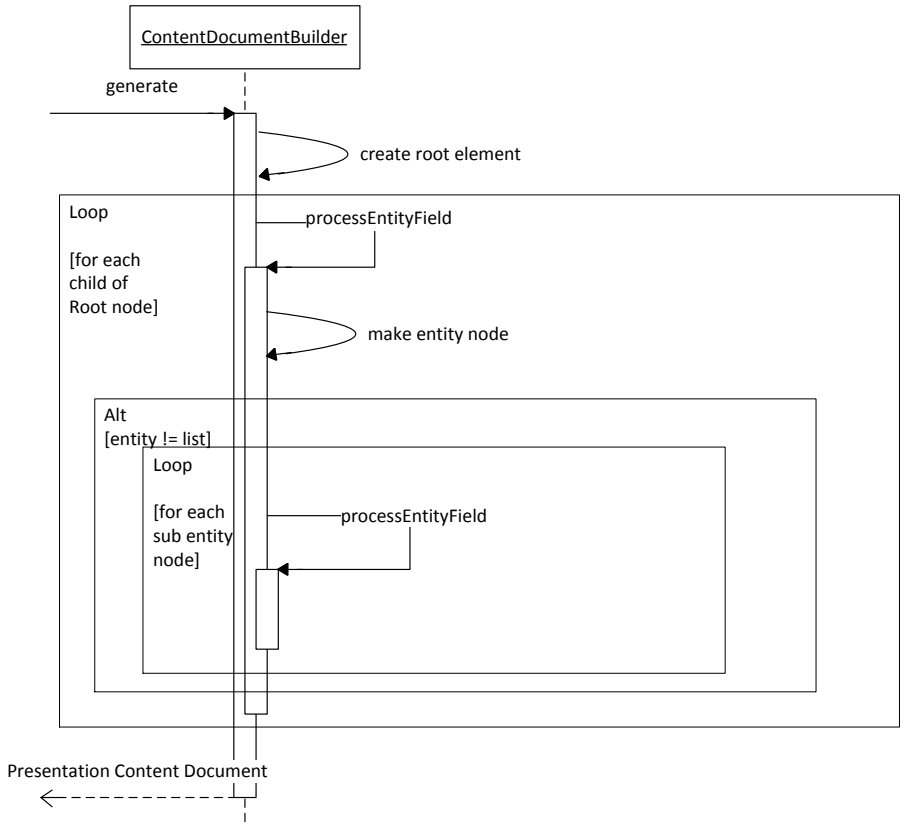
Under utviklingsprosessen til kandidatane var det veldig uklart korleis lister og sub-entitetar blei handtert. Som eksempel 3.4 viser på linje 12 må ein setje `type` atributtet til `list` og så referere til ein annan entitet ved å setje `entity-ref` attributtet til namnet på den andre entiteten. Slik det blir referert til `url` på linje 4 til 7 i samme eksempel.

Eksempel 3.4: Slik ein lagar lister i DPG 2.0

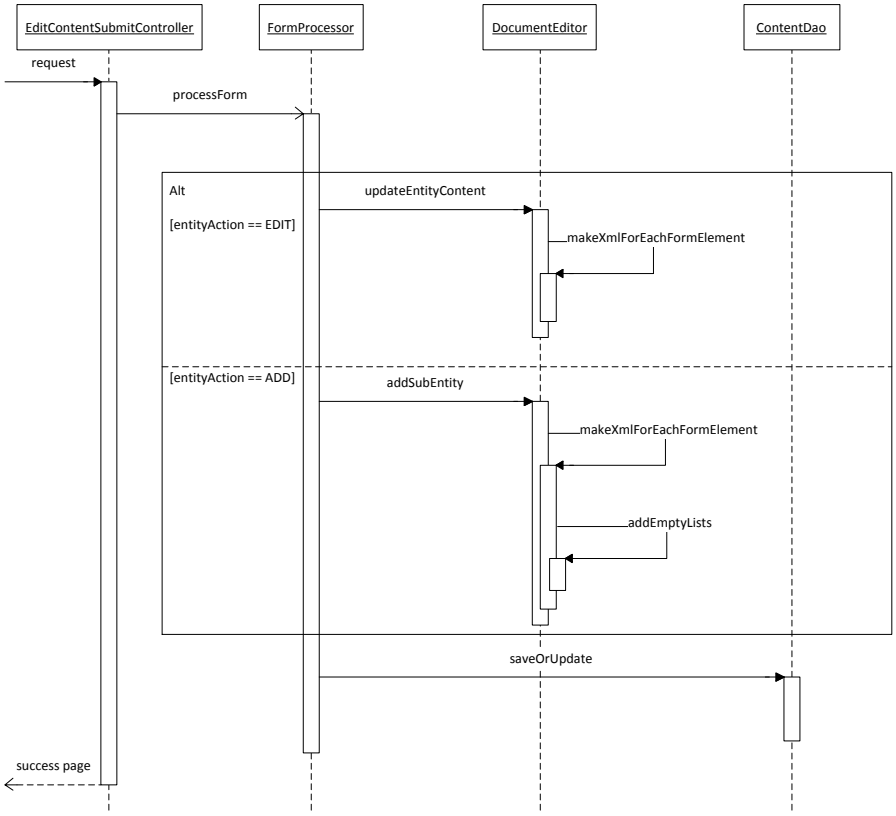
```
1
2  <entities>
3
4    <entity id="url">
5      <field type="string">address</field>
6      <field type="string">name</field>
7    </entity>
8
9    <entity id="softwareEntity">
10     <field type="string" required="true">title</field>
11     <field type="xhtml">description</field>
12     <field type="list" entity-id="url">urls</field>
13     <field type="entity" entity-id="phoneEntity">phone</field>
14     <field type="form2">form2</field>
15     <field type="savedComment2">savedComment2</field>
16   </entity>
17
18 </entities>
```

Plugin-manageren ser dette atributtet og sender dei underliggande nodane til `list`-pluginen. Denne pluginen returnerer desse nodane tilbake til plugin-manageren. Noko som var veldig forvirrande og vanskeleg å forstå, då det heller ikkje finst dokumentasjon på dette området.

Under utviklinga av DPG blei det gradivs klart korleis dette fungerte. Handteringen av lister og sub-entitetar (begge blir behandla på liknande måtar) skjer på 5-6



Figur 3.3: `ContentDocumentBuilder` klargjer alle felt til HTML-presentasjon



Figur 3.4: Form Processor bygger skjema for presentasjon i PCE

forskjellige plassar i systemet, inkludert PM, PCE og PV. Som figur 3.3 viser er det `ContentDocumentBuilder` klassen som går gjennom kvart felt og klargjer desse til HTML-presentasjon og sender det til `PresentationContentDocument` klassen.

Figur 3.4 viser korleis `FormProcessor`-klassen bygger skjemaet som skal presenteres for brukere i PCE. Skjemaet gjer brukaren moglegheit for å legge inn innhald. Gjennom ein intern sjekk finn `FormProcessor`-klassen finn ut korleis skjemaet skal presenteres. Klassen sjekker om attributtet `type` inneholder verdien `list` og vidare kva attributtet `entity-id` har som verdi. Denne verdien er namnet på ein anna entitet, det er denne entiteten som lista skal innehalde. Det er viktig å få med seg at denne logikken ikkje ligg i ein plugin, men i kjernen i DPG.

I eksempel 3.5 viser ser ein korleis PCE legg til eit tomt felt i innhaldsdokumentet for kvar gong brukeren vel *ADD* i PCE. For så å legge inn informasjon i den ved å velje `EDIT`.

Eksempel 3.5: PCE handterar brukarval i `FormProcessor`.

```
1  // Perform either update existing entity or adding of a sub entity
2      switch (formParameters.getEntityAction()) {
3      case EDIT:
4              logger.debug("Editing an entity.");
5              documentEditor.updateEntityContent(doc, form);
6              break;
7      case ADD:
8              logger.debug("Adding an entity.");
9              documentEditor.addSubEntity(doc, form);
10             break;
11     default:
12         throw new FormProcessorException("Unsupported entity action '" ↩
               + formParameters.getEntityAction()
13                 + "'. Only edit and add are supported");
14     }
```

I listepresentasjonen i PV blir det behandla spesialtilfelle i `FieldPluginManageren` klassen, som har som oppgåve å gje kvart felt til sin plugin. Vidare blir lista handtert i `list` pluginen. Å handtere slike spesialtilfelle i fleire delar av systemet bryt med poenget til plugin-arkitekturen. Det gjer det vanskeleg å forandre arkitekturen eller dei pluginane som handterer desse tilfella utan at DPG vil krasje i andre delar av systemet.

Kandidatane er einige i at både systemet og vidareutviklinga vil ha store fordelar ved å innføre ein meir generell måte å handtere lister og sub-entitetar på. Dette vil forenkle vidareutviklinga og gjere systemet meir robust ved andre større endringar seinare samt gjere koden mykje meir oversiktleg og rein.

4

Forbetring i pluginarkitekturen

Heile dette kapittelet er eit samarbeid og vidareføring av førre kapittel mellom Kelly Alexander Teigland Whiteley [50] og kandidaten. Kandidatane har skrive desse kapitla kvar for seg.

Dette kapittelet er ein gjennomgang av kandidatanes forbetringer i den noverande pluginarkitekturen og listehandteringen. Etter ein gjennomgang av desse, vil det bli presentert ein konklusjon på kva kandidatane meiner er den beste forbetringa.

Gjennom heile kapittelet blir det lagt vekt på å lage ein generell løysing, som både er fordelaktig kandidatane si utvikling, men også andre utviklarar og brukarar av DPG.

Som tidlegare nemnd er ein god abstraksjonen mellom dei forskjellige lag ein vikitg del av DPG. Dei forskjellige utviklarane skal sleppe å kjenne til forskjellige delar av systemet. Ein mønsterutviklar skal sleppe å tenke på kva plugin han bruker, og ein pluginutviklar skal sleppe å setje seg inn i mønsteret.

Kandidatane vil både legge vekt på abstraksjonen og prøve å følge Martin Fowler sine beste praksisar [19]. Ein naturlig framgangsmåte for kandidatane med tanke på å finne den beste løysinga for alle parter.

4.1 Fleirfelts-plugins (Eng. *Multiple field plugins*)

DPG har som tidlegare nemnd ikkje hatt støtte for at ein plugin skal kunne handtere fleire felt på ein gong. Det er dette som blir dette kapittelets hovudfokus.

4.1.1 Gå tilbake til DPG 2.0

Ei løysing kandidatane har vurdert, er løysinga som var i DPG 2.0. Då må ein legge til ein kontrakt `EntityInputFieldPlugin` som handterer fleire felt. Og dei pluginane som skal ha denne funksjonaliteten må da implementere denne kontrakten.

I masteroppgåva til Peder Lång Skeidsvoll [41] og Tobias Rusås Olsen [34] har dei vurdert ei slik løysing. Deira konklusjon var at desse kontraktane var for like, og ein heller skulle slå desse saman til ein kontrakt. Noko kandidatane er einige i, då det er føretrekke å ha ei uniform behandling av alle plugins i ein arkitektur.

Det er nødvendig å legge til at det ser ut som Tobias Rusås Olsen og Peder Lång Skeidsvoll si samanslåing har gått ut over funksjonaliteten. Det er nemleg ikkje lengre mogleg å handtere av fleire felt i ein plugin.

I tillegg vil kandidatane tilføye at ein mønsterdesignar også må vite korleis ein plugin forventar at innhaldet skal sjå ut. Dette betyr at mønsterdesignaren må tileigne seg unødvendig kunnskap før han kan lage eit nytt mønster.

4.1.2 Utvide plugin-kontrakten

Å utvide ein kontrakt er ei generell løysing. Eit eksempel på ei slik utviding er dette delkapittelets hovudfokus. Viss ein utvidar kontrakten som allereie er der til å kunne ta inn fleire felt, kan ein plugin få alle dei felta den treng. Dette vil gje moglegheit for å handtere ei samling av felt i ein og same plugin, og pluginen vil då kunne generere ferdig HTML-kode for alle felta. Dette bidrar til at ein må stille krav til mønsteret, då møsteret må definere kva felt som høyrer til kva plugin.

4.1.3 Referere ved instansiering av eit nytt attributt

Ei anna løysing kan være at ein refererer til ein plugin, ved å inføre eit nytt attributt. Som eksempel 4.1 på linje 5 viser kan dette gjerast ved å innføre `"plugin-ref"` atributtet som gir plugin-manageren instruksjoner på kva plugin som skal handtere denne vidare. Plugin-manageren må sjølvsagt modifiserast litt, slik at den gjer frå seg

eit element med alle barna i for at pluginen får alle felt den treng. Dette er imidlertid den einaste store modifikasjonen ein treng.

Eksempel 4.1: Referer til plugin ved attributt

```
1
2  <specification>
3    ...
4    <entities>
5      <entity id="map" plugin-ref="dynamicMapPlugin">
6    </entities>
7    ...
8  </specification>
```

Dette vil vere ein måte å få listehandtering i systemet på, men det er og den funksjonaliteten den vil gi. Den vil ikkje fremme mykje ekstra funksjonalitet for annan handtering. Men her og må mønsterdesignaren ha innsikt i kva pluginen forventar å få inn.

4.1.4 Pluginen definerer strukturen

Som nemnd øvst i dette kapittelet er det viktig for kandidatane at dette er ei løysing som gjer fordelar for alle som brukar/utviklar i DPG. Abstraksjonen som først var tenkt i DPG, har ikkje blitt fullkommen i det noverande systemet. Ein mønsterdesignar er avhengig av kunnskap om korleis pluginen forventar strukturen, på lik linje som plugin-designaren avhenger av at mønsterdesignaren skriv strukturen rett. Sjå eksempel 4.2 for å sjå korleis ein definerer denne struktren i fila `pattern.xml`

Eksempel 4.2: Løysning på bruken plugin i pattern.xml fila utan forandringar.

```
1  <specification>
2    <entities>
3      <entity id="marker">
4        <field type="String" required="true">name</field>
5        <field type="String" required="true">longitude</field>
6        <field type="String" required="true">latitude</field>
7      </entity>
8      <entity id="dynamicMap">
9        <field type="list">marker</field>
10     </entity>
11     ...
12   </entities>
```

```
13    ...
14  </specification>
```

Eit alternativ for å unngå dette er å la pluginen sjølv definere sin struktur. Då vil ein plugin alltid ha kontroll på den strukturen den får inn. Det vil vere ein stor fordel for mønsterdesignaren og, som då ikkje treng bry seg med korleis strukturen skal sjå ut. Då er det nok for mønsterdesignaren å setje inn i mønsteret kor denne pluginen skal brukast og så ordnar pluginen med strukturen sjølv, slik som presentert på linje 4 i eksempel 4.3.

Eksempel 4.3: Løysing på bruker plugin i pattern.xml med forandringar.

```
1  <specification>
2    <entities>
3      <entity id="dynamicMap">
4        <field type="dynamicMapPlugin">map</field>
5      </entity>
6      ...
7    </entities>
8    ...
9  </specification>
```

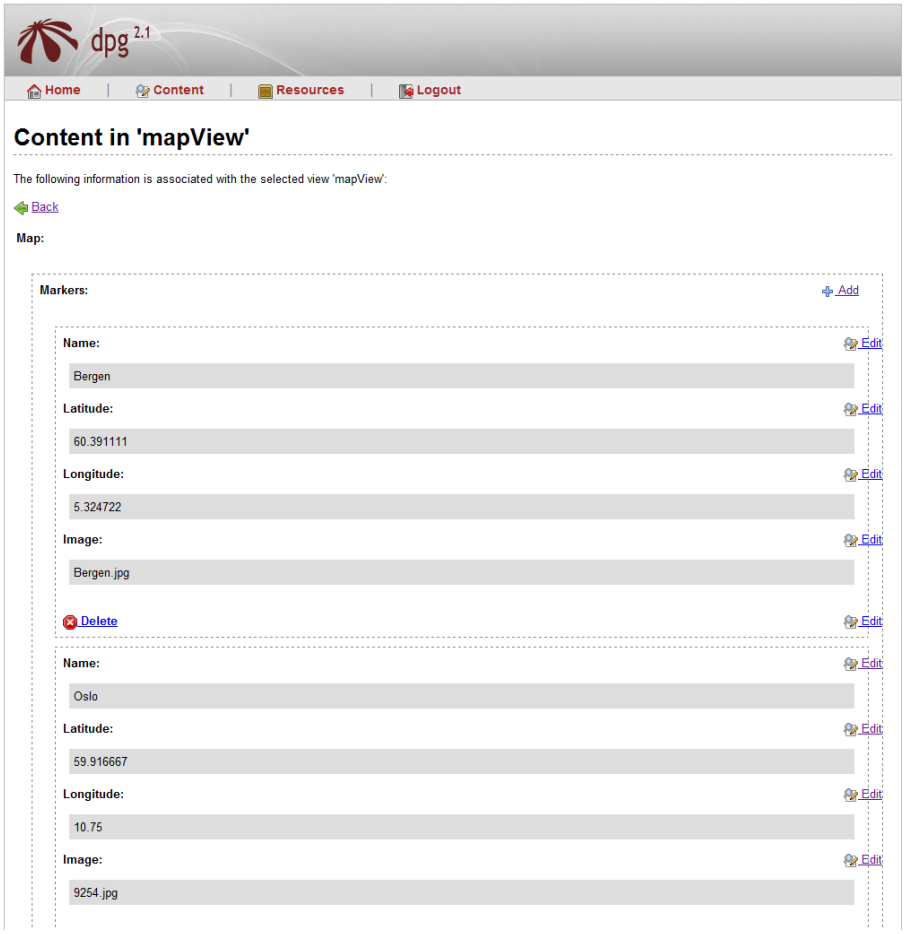
4.1.5 Endeleg løysing og implementasjon

Den endelege løysinga kandidatane har valt, er å la ein plugin definere sin eigen struktur. Denne løysinga har kandidatane valt på grunnlag av den funksjonaliteten som systemet får. I tillegg får ein ei generell løysing som gjer ein bedre abstraksjon enn det noverande systemet har. Ein vil også halde systemet bakoverkompatibelt sidan gamle plugins vil fungere.

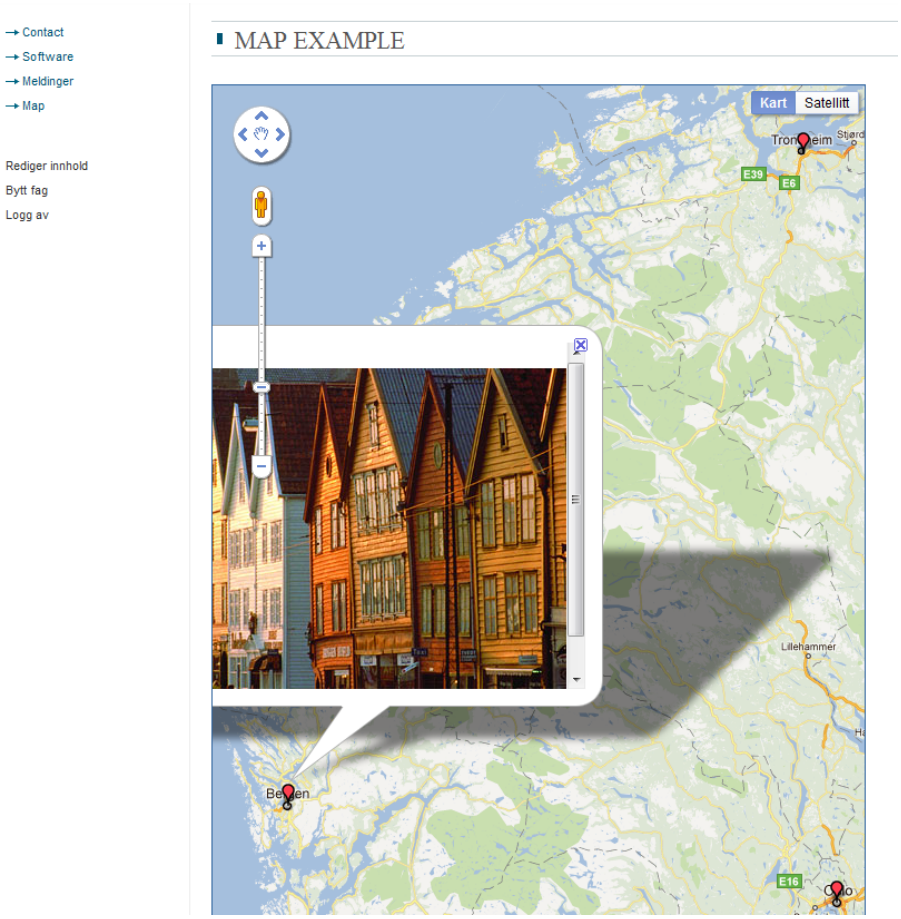
Forandringar i mønsterspesifikasjonen

Ein mønsterdesignar vil som tidlegare måtte bruke gamle plugins på same måte som tidlegare. Ein må kunne strukturen den treng, slik som eksempel 4.2 viser. Men som i eksempel 4.3, treng ikkje dei nye pluginane som genererer sin eigen struktur meir enn ei linje med kode, uansett kor omfattande struktur den forventar.

Figur 4.1 viser korleis det vil sjå ut i PCE sjølv om det ikkje vises i mønsteret. Innholdet i dokmunetet vi altså vere det same uavhengig av om strukturen er definert i mønsteret eller pluginen.



Figur 4.1: Entitet generert av plugin `DynamicMapPlugin` i PCE



Figur 4.2: Kart generert av `DynamicMapPlugin` med innhald fra PCE

I figur 4.2 presenterer PCE innhaldsstrukturen, men alt er generert av ein plugin. På same måte som andre felt blir bestemt må mønsterdesignaren berre bestemme dokumentnoden i XSLT-fila.

Forandringar i plugin kontrakten

Med tanke på den uniforme måten å handtere ein plugin i den noverande DPG, blei den beste løysinga ei utviding av kontrakten. Dette gjer og at ein sikrar bakoverkompabiliteten. Metoden `generatePatternStructure()` blei tilført `Plugin`-kontrakten, som vist i eksempel 4.4. Systemet sjekkar alltid denne metoden ved bruk av plugins. Viss metoden returnerer `null`, vil systemet behandle dette som om det er ein av dei tidlegare pluginane. Viss det derimot er ein ny plugin, vil metoden

returnere ei liste med JDOM-element, som definerer mønsterstrukturen.

Eksempel 4.4: Den nye kontrakten for plugins

```
1
2  public interface Plugin {
3
4      /**
5       * Generate/render the XML element produced by this plugin for the ↩
            views of a presentation.
6       *
7       * @param bean
8       * @param input call-scoped input values, typically from a form ↩
            submit
9       * @throws PluginException
10      */
11     Element generateElement(FieldPluginBean bean, Map<String, Object> ↩
           input) throws PluginException;
12
13     /**
14      * @return a list of parameters accepted by this plugin in ↩
            pluginConfig.xml.
15      */
16     List<String> getParameters();
17
18     /**
19      * @param value The field value
20      * @param field
21      * @return a suitable Form implementation for this plugin
22      */
23     FormElement getFormElement(String value, Field field);
24
25     /**
26      * Based on the FormElement input, return an XML representation,
27      * such as wrapping it in CDATA, putting other kinds of elements
28      * or attributes around it, transforming some values etc.
29      * @param formElement
30      */
31     Content getXmlContent(FormElement formElement, PluginContext ↩
           context);
32
33     /**
34      * Sets the Plugin Resource Dao for the spesific plugin
35      *
36      * @param pluginResourceDao
37      */
38     void setPluginResourceDao(PluginResourceDao pluginResourceDao);
39
40     /**
```

```
41      * Generate the element structure this plugin requires. If it is a ↩
            single field input plugin, return null.
42      * @return list of Elements defining the pattern structure for this↩
             plugin. The first element must always be the one intented to ↩
            be an entity-instance.
43      */
44     Element[] generatePatternStructure();
45
46 }
```

Slik plugin-manageren fungerer er dette ei forholdsvis enkel implementering. Ein sjekk på om pluginen returnerte struktur eller ikkje, var alt som måtte implementerast her. Altså vil DPG først sjekke mønsteret for ein struktur. Om denne ikkje fins der sjekker den også mønsteret som pluginen returnerer.

Forandringar i DPG arkitekturen

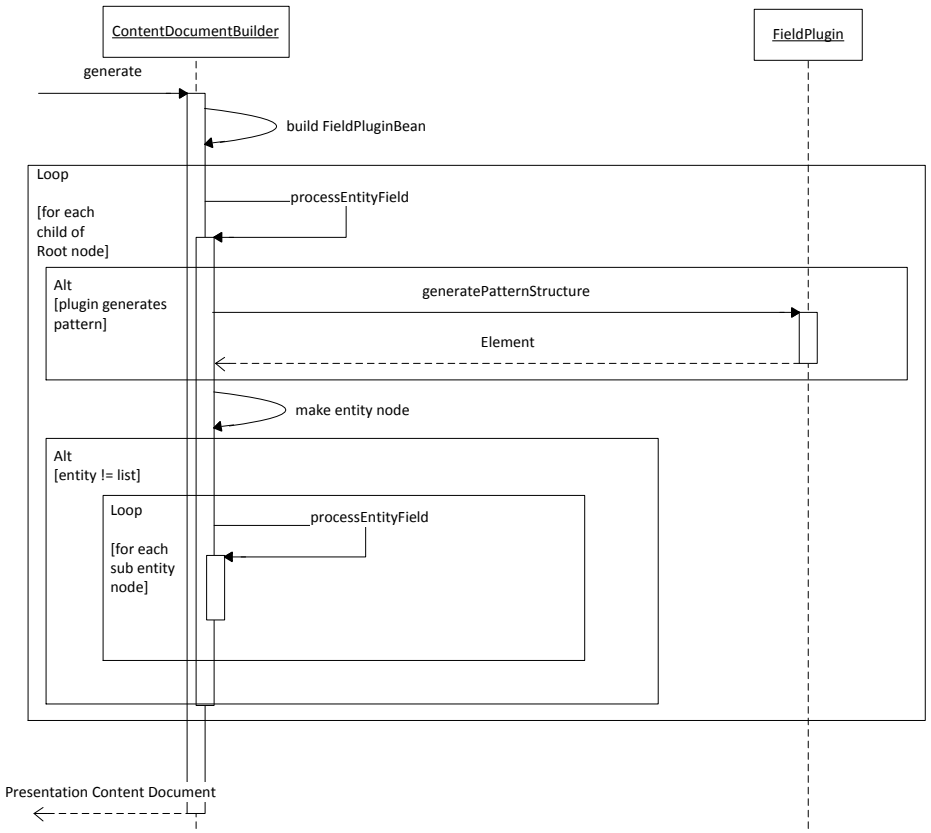
Kandidatane ville gjere forandringane så tidleg som mogleg og prøve å gjere forandringane så integrert som mogleg. Utforskingen begynte derfor på det lågaste laget i DPG, nemleg i Data Access Object (DAO), og så gå videre oppover i DPG.

Det var ikkje mogleg å gjere forandring på på dei låge laga i DPG fordi plugins blir handtert på høgare lag. Nødvendige endringar måtte skje i PCE, PV og PM.

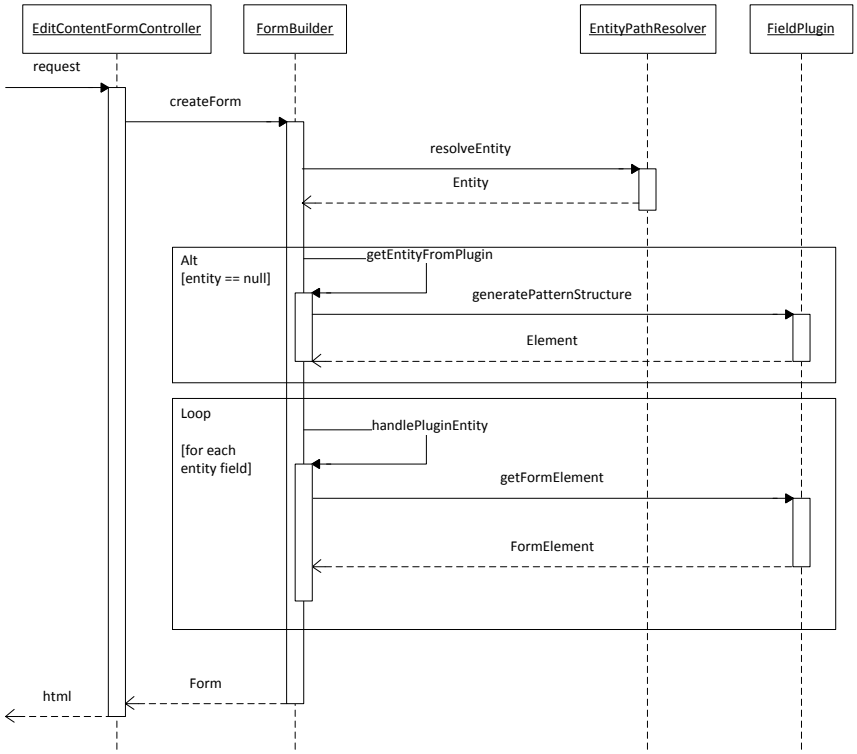
Når ein presentasjon blir laga bygger DPG først eit innhaldsdokument med berre basisstrukturen som er tomt for innhald. Dei vidare stega i kjeldekoden sjekker dette dokumentet for struktur. `ContentDocumentBuilder` klassen i PM er den første plassen som sjekker mønsteret for strukturen og dette måtte ein naturlegvis endra på. Dei nye pluginane har ikkje struktur i mønsteret, men definerar dette sjølv og figur 4.3 viser korleis sjekken også må ta hensyn til at pluginens `generatePatternStructure()` metode kan returnere struktur og.

Etter at ein presentasjon er laga er det PCE som overtar. Hovedfunksjonaliteten her ligg i klassane `FormBuilder` og `DocumentEditor` (kalla på av `FormProcessor` klassen). Som figur 4.4 viser var det kun nødvendig å gjere endringar i klassen `FormBuilder`. Det er her systemet bygger opp HTML-skjema (Eng. *HTML Form*) som skal brukas til å legge inn og redigere innhold. Her blir det naturlegvis sjekka opp mot mønsteret for å finne strukturen, som då krevde at ein la til ein sjekk for og om pluginen genererte strukturen sin sjølv. Deretter kunne pluginen opprette sitt skjema med dei entitetane den fann.

Eit problem som dukka opp under utviklinga var at fleire feltinstansar kan peike til ein og same plugin. Viss denne pluginen igjen genererer sin eigen struktur vil det



Figur 4.3: Ny sjekk i `ContentDocumentBuilder` for om plugins definerer strukturen sjølv



Figur 4.4: Ny sjekk i `FormBuilder` for om plugins definerer strukturen sjølv

ligge to instansar med same namn i innhaldsdokumentet. Eit eksempel på dette er eit kart med to typar markører. Det vil skape ein kollisjon når ein prøver å hente ein entitet. Dette er eit problem som har oppstått i tidlegare utgåver av DPG, då var løysinga å setje inn eit unikt identifikasjonsnummer i tillegg til namnet på entiteten. Den løysinga valde kandidatane for dette problemet og.

Siste forandring i DPG måtte gjerast i PV, som bruker `FieldPluginManager` til å transformere innhaldet til HTML. Det `FieldPluginManager` gjer, er å gå gjennom dokumentet rekursivt. For kvart felt som dukkar opp sender han den til sin respektive plugin gjennom metoden `generateElement()`. Her måtte det og implementeres nok ein sjekk for om pluginen genererte sin eigen struktur. Gjorde den det måtte også denne strukturen gjennomgås rekursivt. Deretter blir alle barn som er definert i pluginen transformert før pluginen transformerer feltet. Dermed er ein sikker på at alle felt blir prossesert av sin respektive plugin. Dette forhindrer at ein må duplisere kode inn i plugins og gjere dei meir kompliserte enn nødvendig.

4.2 Fleire entitetsinstansar i ei og same visning

Dette delkapittelet omhandlar ei rekke løysingar på svakheiter som bli presentert i kapittel 3.6. Som omhandler moglegheiter for å setje saman fleire innhaldsdokument til same visning, og presenterte det komponerte utsnitt i ei visning.

4.2.1 Sannasettning av visningar i ein plugin (Eng. *View composition plugin*

Ein plugin kunne handtert fleire entiter i ei visning. DPG måtte då ha tatt inn fleire spesialtilfelle som `list` og `subEntity`, noko som er ei omfattande implementering og eit steg i feil retning når det gjeld abstraksjon.

4.2.2 Enkel utsnittsliste (Eng. *Single view list*)

Ei anna løysing er å setje fleire entitetinstansar inn i ei liste og la denne lista peike til ei visning. Dette krever at ein må forandre presentasjonsmønsterspesifikasjonen , på grunnlag av relasjonane mellom entitetsinstansar, sjå kapittel 3.6. Det er to løysingar som då er relevante:

- Å implementere handtering for at ein entitetsinstans kan referere til andre entitetsinstansar. Som igjen kan peikast til eit view, som er ei liknande løysing

til korleis lister blir handterte i dag.

- Å referere fleire entitetsinstansar direkte til eit utsnitt i mønsterspesifikasjonen.

Denne implementasjonen med visninger som peiker til XSLT-dokumentet er forholdvis enkel, fordi PV berre kan sette dokumenta saman og sende dei til ei visning, og mønsterdesigneren kan bruke denne informasjonen som den vil. For at plugins som tar inn fleire felt gjennom plugin-manageren blir løysinga meir komplisert. Her må ein mønsterdesignar vite kva entitetar som skal presenteres saman, då alt som skjer i pluginen er skjult for mønsterdesignaren.

Denne løysinga vil vere både bakoverkompatibel og halde på abstraksjonen.

4.2.3 Endeleg løysing

Den anbefalte løysinga er å peike fleire entitetsinstansar til ei konkret visning i mønsteret. På grunnlag av bakoverkompabilitet og at abstraksjonsnivået er overhalde, er dette eit klart val av løsying.

Forandringar i presentasjonsmønsterspesifikasjonen

Presentasjonsmønsterspesifikasjonen blir utvida til å handtere ei liste av entitetsinstanser som peiker til ei enkel visning. Ved å innføre eit attributt som heiter `entity-instance-ref`. Denne atributten kan ein legge inn fleire entitetsinstansar separert på eit semikolon slik eksempel 4.5 viser på linje 29. Semikolon blei valt fordi det ikikkje bryt syntaksen i XML-strukturen.

Eksempel 4.5: Løysing i Presentasjonsmønsterspesifikasjonen på korleis ein kan presentere fleire entitetar i ei enkel visning

```
1  ...
2
3      <entity id="markerEntity">
4        <field type="string">name</field>
5        <field type="string" required="true">latitude</field>
6        <field type="string" required="true">longitude</field>
7      </entity>
8
9      <entity id="mapEntity">
10       <field type="list" entity-ref="markerEntity">markers</field>
11     </entity>
12     ...
```

```
13
14      <entity-instance id="barMarkersInstance">
15        <entity-ref>markersEntity</entity-ref>
16      </entity-instance>
17
18      <entity-instance id="restaurantMarkersInstance">
19        <entity-ref>markersEntity</entity-ref>
20      </entity-instance>
21    </entities>
22    ...
23
24    <views>
25      <view id="markersView">
26        <description>Bars and Restaurants</description>
27        <entity-instance-ref>barMarkersInstance;restaurantMarkersInstance↩
              </entity-instance-ref>
28        <transformation>barAndRestaurantsTransformer</transformation>
29      </view>
30    <views>
31  <spesification>
```

Forandringar på DPG-arkitekturen

Forandringar skal hovudsakleg skje i *PV*. Når det gjeld *PCE* skal ikkje ein publisher ha tilgang til å forandre samansatte entitetsinstanser, men har selvfølgelig tilgang til kvar enkelt entitetsinstans separat.

Forandringane i *PV* startar i klassen `PresentationViewerServiceImpl` i metoden `createViewContent()`. Det er her innhald frå kvar entitetsinstans handterast og bli sendt til ei visning. Vidare blir innhaldet sendt til plugin-manageren og transformert derfra. Samansettinga bør derfor skje rett før det blir sendt til plugin-manageren slik at dette blir transformert og derfor vist i ei og same visning og.

5

Geodata i DPG

Den nye funksjonaliteten opnar for at kandidaten bruke den for sitt overordna mål om geodata støtte i DPG. Kandidaten her fokusert på to områder:

- Presentasjon av geodata i DPG (Delkapittel 5.2)
- Innlegging av geodata i DPG (Delkapittel 5.3)

Desse punkta har sin naturlege plass i ei implementering av eit kart i DPG.

I tillegg til dette tar også kapittelet for seg dei forskjellige kart API-ene som har blitt vurdert, samt ein gjennomgang av av korleis implementasjonen har blitt gjort og korleis kandidaten løyste problem undervegs.

5.1 Vurdering av kart API-er

Fleirtallet forventer at nokon funksjonar er tilstades i eit nettbasert kart. Desse standard funksjonane er blandt anna Zoom, Perspektiv, Veibeskrivelser og Lag. Dette er kriterier som både Google Maps [22], Nokia Maps [33] og Bing Maps [5] oppfyller. Dette delkapitelet vi gjennomgå tre kart API-er med hensyn på dei overnemnde funksjonane samt at API-ene vil vurderast på tre punkt og:

- Kva kontrakten tilbyr av funksjonalitet som for eksempel zoom.
- Kva utviklingshjelpemiddel som er tilgjengelige.
- Kor mykje detaljar kartet har, både i forhold til 3D-modellen og bildekvaliteten.

5.1.1 Google Maps API

Dei fleste er kjent med Google Maps sitt kart som vert brukt på både nettsider, i applikasjoner og på mobile einhetar. Kartet kan brukes i alle ukommersielle samanhenger utan tillatelser.

Google Maps sitt API tilbyr eit mangfold av moglegheiter for implementering. Google Maps tilbyr både JavaScript og Flash som implementeringsteknologi for kartet. I tillegg finst det mange nettenester som gir funksjonalitet som:

- Å innehalde informasjon om plasser
- Oversette adresser til lengdegrad og breiddegrad
- Moglegheiter for å gje høgdemeter
- Gi kalkuleringer på både korteste veg i kilometer fra eit punkt til eit anna, samt raskeste veg i tid.

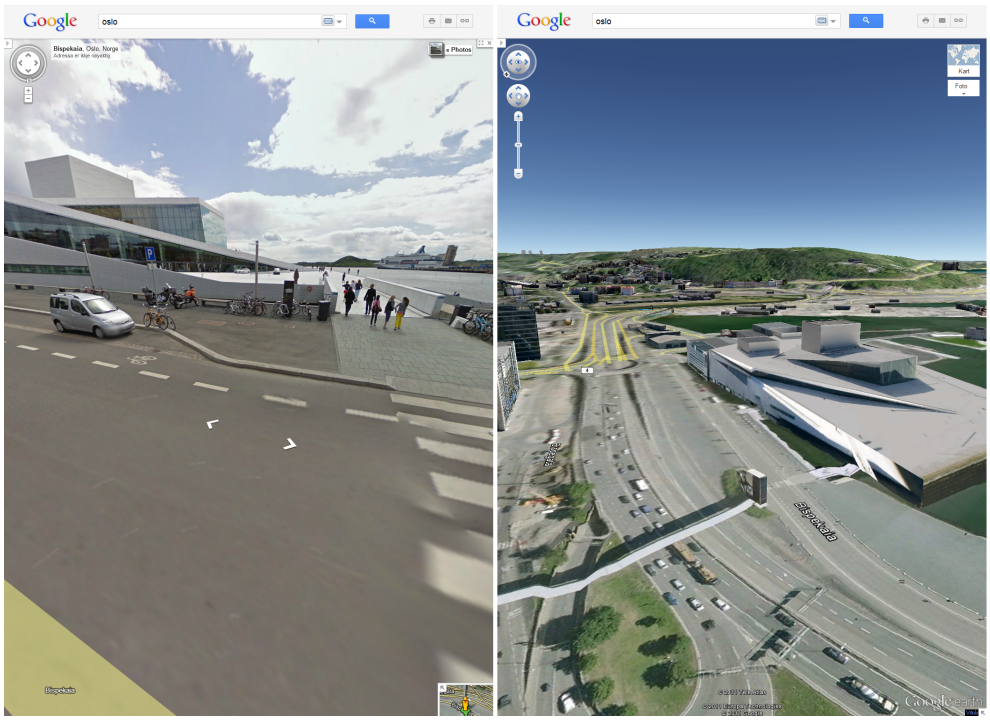
API-et har naturlegvis god dokumentasjon på alle område i tillegg til eit titals gode nettbaserte opplæringsprogram som gir den informasjonen ein måtte trenge for å implementere eit slik kart. Google Map har lagt ut tallause eksempel samt ein slags testarena for utviklarar med moglegheiter for å sjå korleis kartet oppfører seg.

Detaljar er og viktig for kart. Dei siste fem åra har det skjedd ei enorm utvikling i detaljnivået til kart. Google Maps har eit stort prosjekt gåande når det gjeld detaljar i sin gatevisningsteneste (Eng. *Streetview*). Prosjektet har blitt gjennført ved at Google Maps har kjørt i dei største byane og tatt 360 graders bilder kvart sekund. Dette har resultert i ein detaljert gatevisning slik figur 5.1 viser til venstre.

I tillegg er det også mogleg for samtlige brukare av Google Maps å sende inn geotagga bilete til Google Map. Som så blir lagt i ein database og presentert i kartet som eit eige lag.

Google har og kjøpt bilder fra Satellite Imaging Corporation (SIC) [7] sin GeoEye 2 satelitt [8], som gir gode bilete av jordas overflate.

Google har dessutan lansert Google Earth som er ei 3D-framvisning av dei fleste storbyane i verda. Google Maps vil ikkje gå ut med kva teknologi dei har tatt i bruk



Figur 5.1: Gatevisning og 3D-visning i Google Maps

for å lage denne 3D-framvisninga, men som ein ser i figur 5.1 til høgre har dei ikkje modellert gangbrua frå Oslo S til Operabygget. Denne blir kun representert som ein tjukk kvit strek, så dei har eit stykke igjen i 3D-framvisninga enno.

Det er og mulig for "vanlige folk" å sende inn 3D-modeleringen av valfrie bygningar som må blir godkjend for å bli lagt ut i Google Earth.

5.1.2 Nokia Map API

Nokia Maps (tidlegare Ovi Map) er Nokia sin eigen kart-kontrakt og er vel i hovedsak meint som ei navigasjonsløysing, både for mobil og GPS-enheter. Det er også laga eit API for å bruke Nokia Maps på nettsider og.

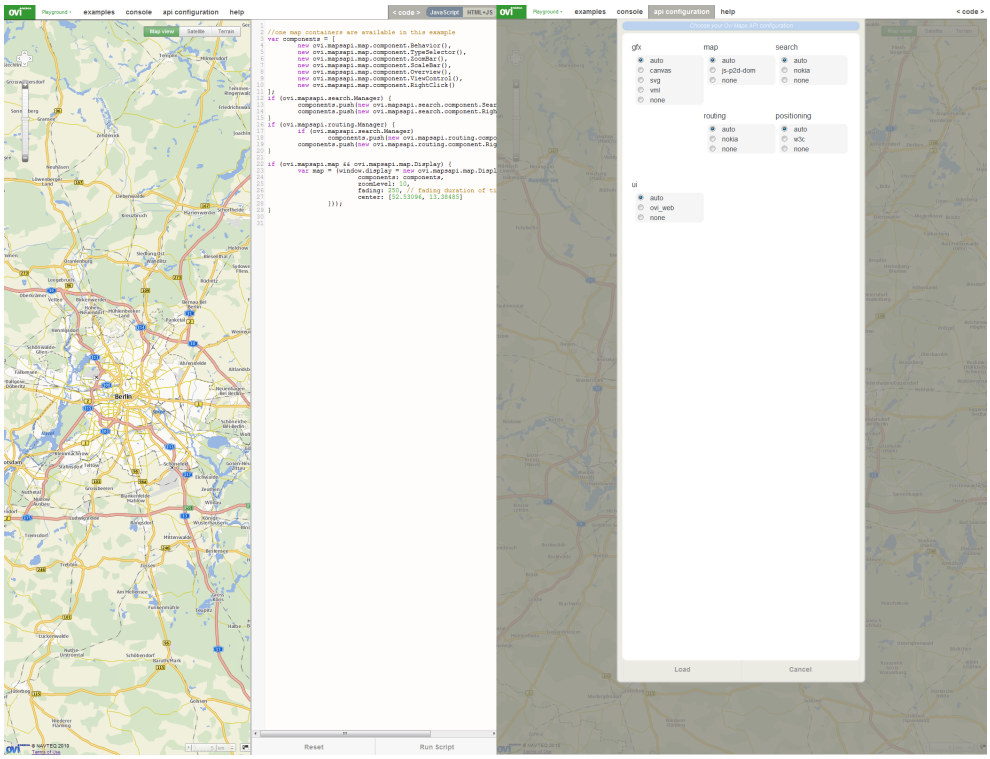
Nokia har satsa på si kartløysing som eit navigasjonskart, og det er heilt klart eit veldig bra kart for den bruken. Men den er og ein god kandidat til internett bruk. Med alle dei eigenskapane som den vanlege mann i gata krev. På same måte som Google Maps kan denne løysinga og implementerast med ein JavaScript-kode.

Her og finnes det funksjonalitet som å oversette adresser til lengdegrad og breddegrad, gi kalkuleringer på raskeste vei til og fra punkt. Ikkje like funksjonsrik som Google Map, men tilbyr dei mest vanlige funksjonane.

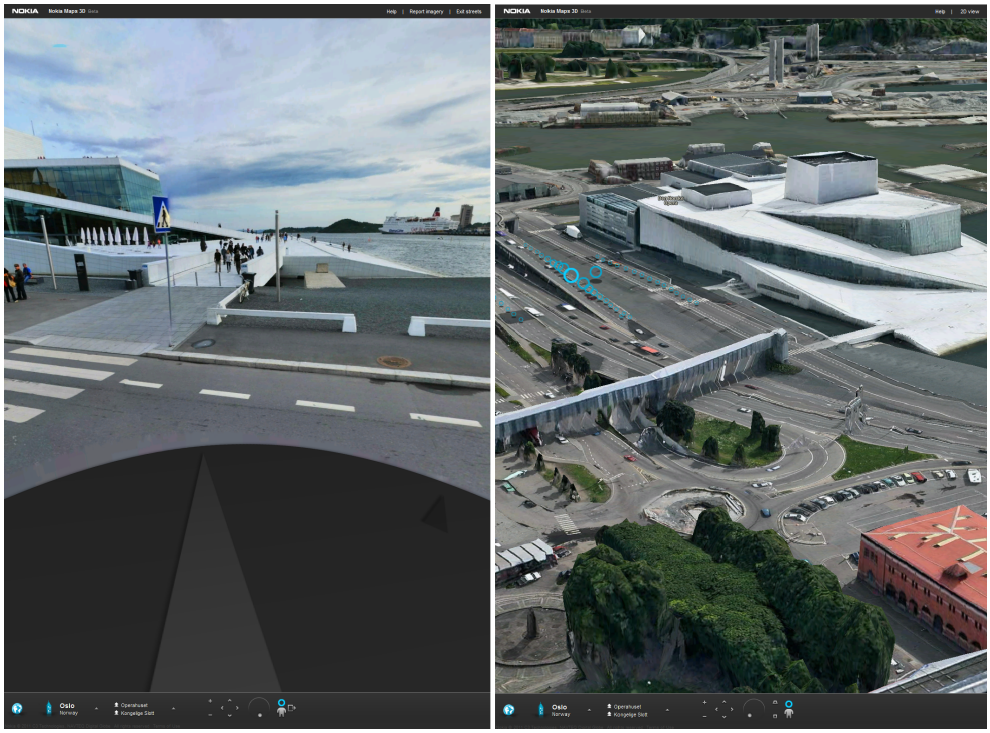
Ein stor fordel for karttuviklarar er at Ovi Map tilbyr alle kartutviklare ein testarena som dei kallar Playground der dei kan få prøve ut forskjellige alternativ til kartet. Testarenaen er ikkje feilfri og presenterte fort feilmeldingar som berre gir ein "ny" kartutviklar unødvendig hovudbry. Som figur 5.2 viser har testarenaen funksjonar for å gje tilbake heile JavaScript-koden til utvikleren og presenterer moglegheiter og alternativ på ein god måte.

Også Nokia Maps har gode detaljar og si eiga gatevisning. Som figur 5.3 viser til venstre er det ikkje stor forskjell fra Google Maps til Nokia Maps når det kjem til gatevisning.

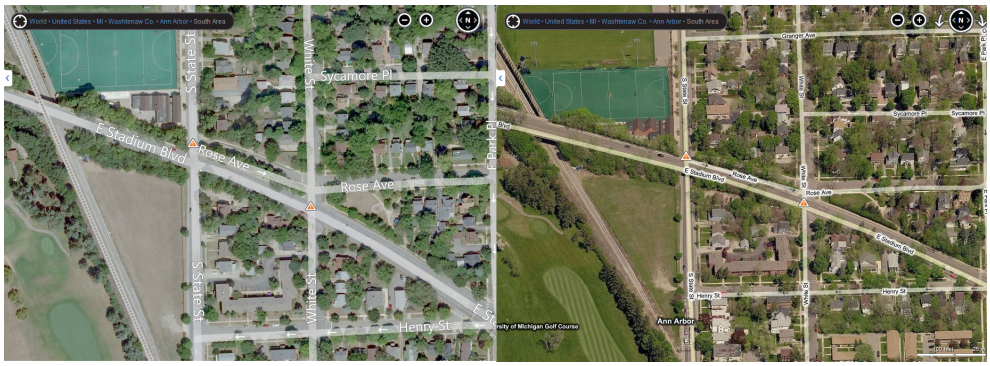
Nokia Maps handterer 3D-framvisninga bedre, med 3D-modellering av motorvei/-overganger og andre detaljar som tre o.l. Her har dei brukt ein teknologi som heiter C3, utvikla av det svenske firmaet C3 Technologies. Teknologien tar opp til 100 bilete av eit objekt og genererer 3D representasjonen ut fra dei. Som figur 5.3 viser til høgre har løysinga svakheiter ved samme gangbru som Google Maps, men her har Nokia Map prøvd å 3D-modellere gangbrua. Så Nokia Map har selvfølgelig sine svakheter og. Ein ser likevell at Nokia Map har ein tanke bak alle utviklingsval som ligg i retning av navigering, framfor eit allsidig kart.



Figur 5.2: Testarenaen i Nokia Maps, til høgre med kode og venstre med alternativ



Figur 5.3: Gatevisning og 3D-visning i Nokia Maps



Figur 5.4: BirdsEye i Bing Map, BirdsEye til høgre og vanleg til venstre

5.1.3 Bing Maps API

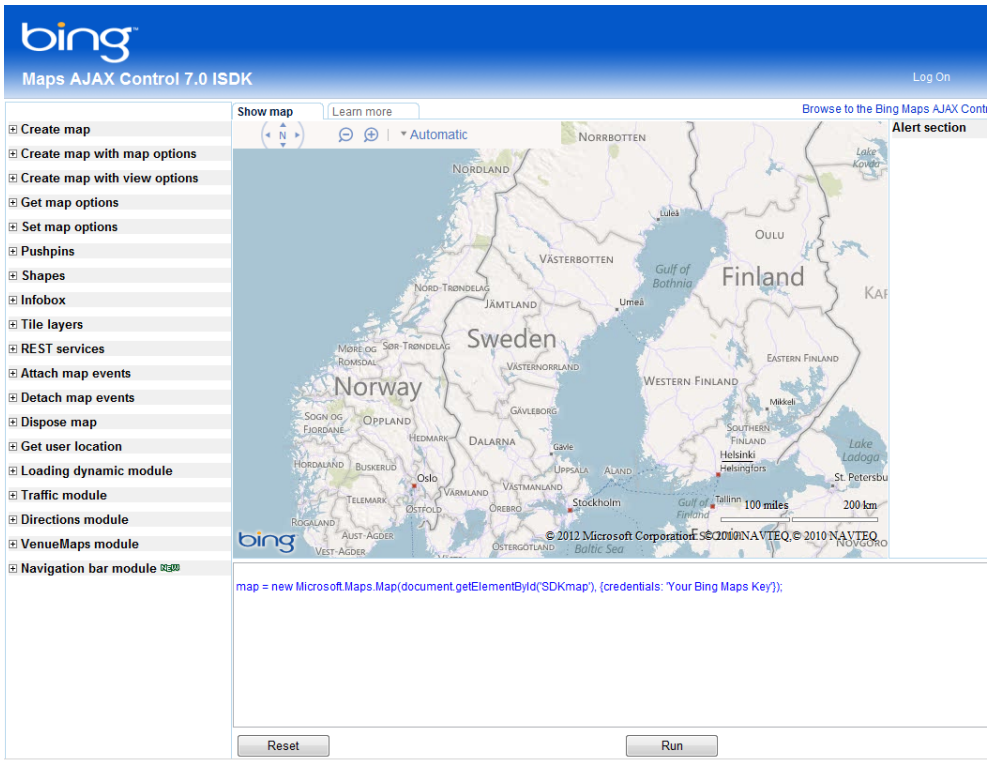
Den siste karttenesten som blir vurdert i denne oppgåva er Bing Map, og er ein direkte konkurrent til Google Maps. Dei gir Google konkurranse og.

Dette kartet er og laga på eit JavaScript, og er like enkel å bruke som dei tidlegare vurderte karttenestene i denne oppgåva. Som i dei to tidlegare nemnde karttenestene finst det funksjonalitet for å omsetje addresser til lengdegrad og breiddegrad samt gje deg kalkuleringer på kjappest veg til og fra punkt.

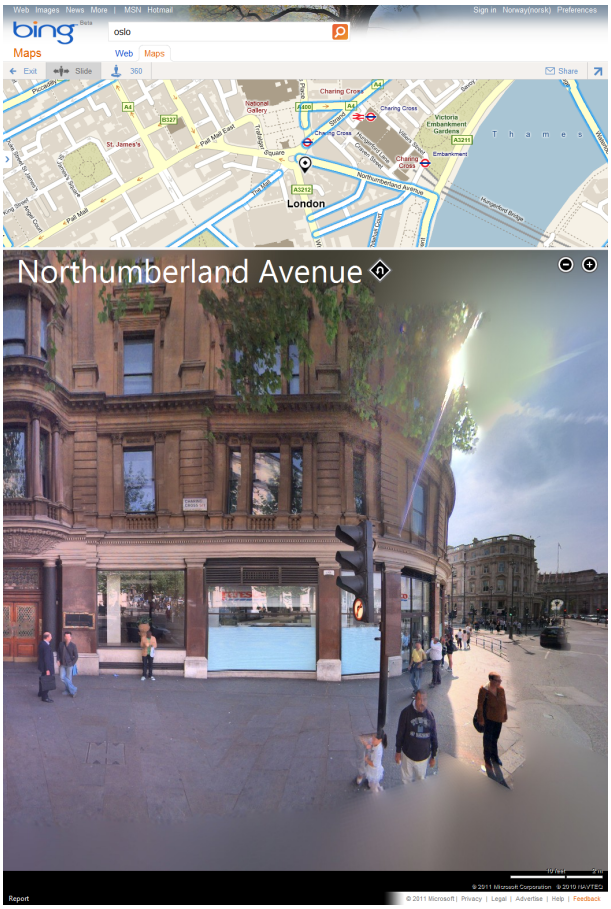
På lik linje med Nokia Map tilbyr Bing og ein slags testarena for sine kartutviklarar. Denne var etter alt å døme utan uforklarlege feilmeldingar men litt tungvindt laga med hensyn på å teste kartkoden. Som figur 5.5 viser har den alle dei samme funksjonane som Nokia-kontrakten tilbyr. Men menyoppsettet her er det forholdsvis rotete og uoversiktleg og viljen til å teste forsvinn i eit svakt brukargrensesnitt.

Heller ikkje i dette kartet er det store mangler å setje fingeren på når det gjeld detaljar. I motsetning til Google og Nokia har Microsoft fjerna sin 3D visning frå sine kart. Dei tilbyr derimot noko som heiter BirdsEye, som bygger på å setje ein lavere innfallsvinkel på perspektivet til brukaren og presentere eit satelittbilde med innfallsvinkel. Dette gir brukeren ein følelse av å sjå ting frå eit fugleperspektiv. Mange vil nok sjå på dette som unødvendig og unyttig, men som figur 5.4 viser er det ein betydelig forskjell, som gjer eit betre perspektiv på korleis landskapet er strekt.

Gatevisninga dei tilbyr er ikkje av same kvalitet som dei andre tilbydarane i denne vurderinga. Bing Maps har hatt eit liknande prosjekt som Google Maps og. Dei har og kjørt gjennom dei store byane, men berre tatt bilde til høgre og venstre. Som figur 5.6 viser gjer dette ein flat struktur på bilete og tilbyr ikkje funksjonar som å



Figur 5.5: Utviklarens testarena i Bing Map



Figur 5.6: "Streetview" i Bing Map

snu seg fram- eller bakover i gata. Det gjer eigentlig ein følelse av å sitje i bilen å sjå ut sideruta. Ikkje ei optimal løysing, dog sikkert mykje rimelegare enn "ordentlig" gatevisning. Dekninga er også minimal per dags dato, og gjeld kun eit titals byar i verda.

5.1.4 Endeleg konklusjon på kartval

Som det kjem fram i dei tidlegare delkapitla er alle karttenestene gode kandidatar til å brukes i ei nettside. Men nokon kandidater er gjerne betre enn andre. Som tabell 5.1 viser er det ikkje dei største forskjellene på desse karttilbyderane.

Tabell 5.1: Samanlikning av kart.

Kartleverandørar/ Funksjonar	Google Map	Nokia Map	Bing Map
Nøyaktighet (Karakter)	Middels	God	Middels
Gatevisning	Ja, gode detaljer	Og i tunnellar og undergangar	Har berre sidevisningar
3D	Google Earth	Nokia 3D	Nei, men tilbyr BirdsEye
Mobilversjon*	Ja, Android og iOS	Ja, Nokiatelefoner	Ja, Windows Mobile
Frakobla modus	Mengde grensa	Ingen mengde grense	Ingen mengde grense
Oppstartsår	2005	2009	2009
Hjelpemidler	Uendelig mengder eksempler	Egen testarena	Egen testarena

*= finnes uoffisielt på dei fleste plattformer

Kartvalet er tatt på grunnlag på kva funksjonar det skulle ha i DPG og kva kart som ville gje minst hovudbry i implementasjonen. Samtidig som det er lagt ned tanker på kva den i framtida kan brukast til.

Bing Maps blir nok den første karttenesten som forsvinn frå denne vurderinga. Grunnlaget for dette ligg i tilleggsfunksjonane som at 3D er fjerna, og erstatta med BirdsEye. Samt at gatevisning har både dårlig dekning og ikkje minst ein veldig dårlig kvalitet.

Når det gjeld Google Maps og Nokia Map er dei begge veldig gode på detaljar og det er berre små forskjellar som skil dei ellers. Nokia har ein betre 3D- modellering som også fungerer i underganger og overganger og. Men dette har Google ignorert og

heller starta eit eige prosjekt der folk kan legge inn sine eigne modellar av forskjellige bygningar. Dei modellane som ligg ute, anten dei er laga av Google Maps sjøl eller av brukarar, er veldig gode. Gatevisning har dei begge løyst på ein god måte, dog er Google komme lengre i sin kartlegging av gater.

Vidare er det utviklingsmoglegheitene som avgjer kva kart som blir kandidatens val. Her stiller Nokia opp med ein testarena for utviklere som gjer fullstending kode for valfritt kart med valfrie alternativ. Denne har ei og anna feilmelding, men Google Maps har sin eiga løysing på ein Playground. Denne er ikkje like interaktiv, men utan uventa feilmeldinger.

Konklusjon

Kandidatens val fall på Google Maps, då denne er mest utbredt både med hensyn på kartets detaljar og tenester samt at den er godt utprøvd og enklast å finne løysinger på alle typar problem som måtte oppstå.

Det faktum atNokia Map er meir presis i sine addressesøk og navigasjon ser ikkje kandidaten på eit problem for den bruk kartet skal ha i DPG. Eit kart i DPG kjem aldri til å omhandle navigasjon på eit høgare nivå enn at ikkje Google Maps mestrar det veldig fint.

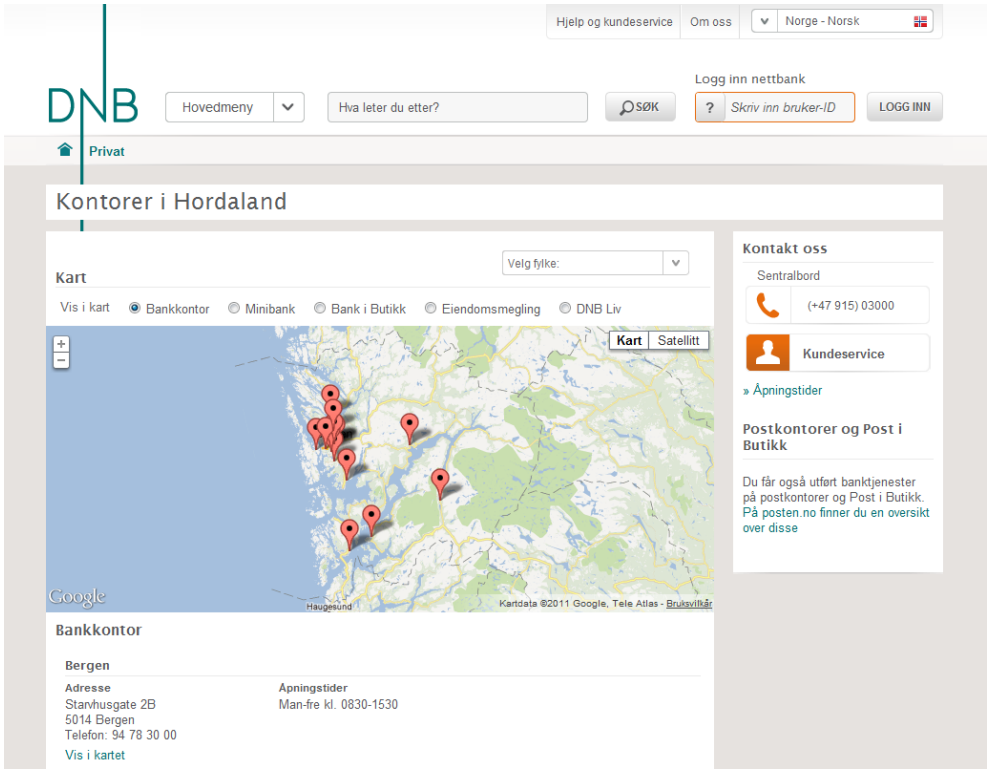
Kandidaten meiner Google Maps ikkje berre er den beste tenesten akkurat no, men og på lang sikt. Den er den lengstlevande karttenesten av ein slik størrelse, og i motsetning til dei andre tenestene er det ikkje annonsert kutt i satsninga framover, heller ikkje på delar av systemet. Kandidaten er sikker på at Google Maps er den beste løysinga for DPG.

5.2 Presentasjon av Geodata i DPG

DPG som eit CMS har stor bruk for moglegheiter til å presentere geografiske plasseringer i form av ein kartrepresentasjon.

Ein kan for eksempel ta for seg det prosjektet som DPG hovudsakleg blir brukt til per dags dato. Her kan ein lokalisere eksamenslokaler og hovudkontor for fjernundervisninga. Men viss ein ikkje tenker på det begrensa utvalet mønster som er i dag, kan ein sjå for seg større samlinger data som for eksempel:

- samling restaurantar/barar
- samling togstasjonar med avgangar



Figur 5.7: DNB sin presentasjon av lokaler i Hordaland

- alle kontor for ei bedrift
- alle plassar ein har vore på ferie

Som figur 5.7 viser har Den Norske Bank (DNB) ein slik presentasjon av sine lokaler, minbankar og bankar i butikk, noko som er nyttig for deira kunder og samarbeidpartnarar. Det er da ikkje vanskeleg å tenkje seg til andre tilfelle der kartet vil vere nyttig. DNB bruker Google Maps [22] sine karttenester.

Kandidaten har valt å implementere desse to tilfella:

- ein presentasjon av resturantar og barar i Bergen
- ein presentasjon av alle bybanestasjonane i Bergen

Dette fordi det vil vise to bruksområder `DynamicMap`-pluginen vil fungere bra til, samt at dette er to tilfeller som er vil vere nyttige for vidare bruk av DPG.

5.2.1 Pluginen

Dei to implementasjonsvala bygger naturlegvis på same plugin, berre med små modifikasjonar. Dette delkapittelet vil dreie seg om sjølve pluginen og korleis den er bygd opp.

Kontraktens arva metodar

Ein plugin kontrakt i DPG arvar ein del metodar gjennom ein utvidelse med kontrakten `Plugin`. Metodane som blir gitt i utvidinga er presentert i eksempel 5.1.

Eksempel 5.1: Kontrakten for plugins

```
1
2  public interface Plugin {
3
4      /**
5       * Generate/render the XML element produced by this plugin for the ↩
            views of a presentation.
6       *
7       * @param bean
8       * @param input call-scoped input values, typically from a form ↩
            submit
9       * @throws PluginException
```

```
10       */
11      Element generateElement(FieldPluginBean bean, Map<String, Object> ↩
            input) throws PluginException;
12
13      /**
14       * @return a list of parameters accepted by this plugin in ↩
             pluginConfig.xml.
15       */
16      List<String> getParameters();
17
18      /**
19       * @param value The field value
20       * @param field
21       * @return a suitable Form implementation for this plugin
22       */
23      FormElement getFormElement(String value, Field field);
24
25      /**
26       * Based on the FormElement input, return an XML representation,
27       * such as wrapping it in CDATA, putting other kinds of elements
28       * or attributes around it, transforming some values etc.
29       * @param formElement
30       */
31      Content getXmlContent(FormElement formElement, PluginContext ↩
            context);
32
33      /**
34       * Sets the Plugin Resource Dao for the spesific plugin
35       *
36       * @param pluginResourceDao
37       */
38      void setPluginResourceDao(PluginResourceDao pluginResourceDao);
39
40      /**
41       * Generate the element structure this plugin requires. If it is a ↩
             single field input plugin, return null.
42       * @return list of Elements defining the pattern structure for this↩
              plugin. The first element must always be the one intented to ↩
             be an entity-instance.
43       */
44      Element[] generatePatternStructure();
45
46  }
```

Av desse er det nokon som må overskrivast i `DynamicMap`-pluginen. Den første som må overskrivas er `generateElement(FieldPluginBean, Map<String, Object>)`-metoden. Det er denne metoden som blir ansvarleg for å presentere kartet ut fra det innhaldet ein vel å legge inn. Det er her ein skal generere heile kartet, noko som delkapittel 5.2.2 vil gjennomgå.

Eksempel 5.2: Metoden `generatePatternStructure()` som genererer strukturen til ein enkel kartplugin i DPG 2.1

```
1
2  public Element[] generatePatternStructure() {
3
4      // Create an insance: markerEntity
5      Element markerEntity = new Element("entity");
6      markerEntity.setAttribute("id", "markerEntity");
7
8      // Create an insance: nameField
9      // and add it to markerEntity
10     Element nameField = new Element("field");
11     nameField.setAttribute("type", "string");
12     nameField.addContent("name");
13     markerEntity.addContent(nameField);
14
15     // Create an insance: latField
16     // and add it to markerEntity
17     Element latField = new Element("field");
18     latField.setAttribute("type", "string");
19     latField.setAttribute("required", "true");
20     latField.addContent("latitude");
21     markerEntity.addContent(latField);
22
23     // Create an insance: longField
24     // and add it to markerEntity
25     Element longField = new Element("field");
26     longField.setAttribute("type", "string");
27     longField.setAttribute("required", "true");
28     longField.addContent("longitude");
29     markerEntity.addContent(longField);
30
31     // Create an insance: mapEntity
32     Element mapEntity = new Element("entity");
33     mapEntity.setAttribute("id", "mapEntity");
34
35     // Create an insance: listField
36     //and add it to mapEntity
37     Element listField = new Element("field");
38     listField.setAttribute("type", "list");
39     listField.setAttribute("entity-id", "markerEntity");
40     listField.addContent("markers");
41     mapEntity.addContent(listField);
42
43     // Make an element with the list of elements
44     // to return
45     Element[] el = { mapEntity, markerEntity };
46
47     return el;
48
```

```
49    }
```

Den einaste metoden som må overskrivast er `generatePatternStructure()`. Det er i denne metoden ein spesifiserer den informasjonen pluginen skal bruke. I delkapittel 4.1.4 blir det gjennomgått to måtar ein kan spesifisere dette på, men i dette tilfelle er det altså metoden `generatePatternStructure()` som må overskrivast. Denne returnerer berre eit tomt objekt som standard, men i dette tilfelle skal den returnere ein trestruktur med eit rotelement på toppen, og med alle markørar under.

På linje 45-47 i eksempel 5.2 kan ein sjå korleis elementet `el` blir satt saman av elementa `mapEntity` og `markerEntity`, og videre returnert. Slik eksempel 5.3 viser, blir det laga ein struktur for brukarane i `generatePatternStructure()` metoden i pluginen. Denne strukturen gir DPG informasjon om kva informasjon pluginen treng/forventar å få inn. Så denne strukturen definerer kun ein enkel markør med namn, og to koordinatar for punktet i kartet.

Eksempel 5.3: Strukturen som `generatePatternStructure()` genererer

```
1   <entity id="markerEntity">
2     <field type="string">name</field>
3     <field type="string" required="true">latitude</field>
4     <field type="string" required="true">longitude</field>
5   <entity/>
6
7   <entity id="mapEntity">
8     <field type="list" entity-id="markerEntity">markers<field/>
9   <entity/>
```

5.2.2 Kartimplementasjonen

Som tidlegare nemnd er det `generateElement()`-metoden som returnerer det som skal presenteres i utsnittet. Dette skjer ved at plugin-manageren kallar på metoden i ein plugin og sender ved ei bønne kalla `FieldPluginBean` som inneheld heile lista med markørar. Vidare blir dette behandla slik dei respektive pluginene har blitt implementert. Kartpluginen skal handtere ei liste med markører og lage eit kart for å presentere dette. Det er då nødvendig å utvikle eit kart som kan gjere akkurat dette.

JavaScript-koden som kart-pluginen skal generere er bygget opp på retningslinjer frå Google Maps sin eigen API-dokumentasjon [22]. Som ein kan sjå i eksempel 5.4 er

ikkje sjølve kartet mange linjer kjeldekode, men så viser det berre eit kart og ikkje noko data utanom.

På linje 1 - 2 i same eksempel viser ein kva som må til av HTML-kode for å ta inn JavaScript-biblioteket til kartet samt på linje 21 den rette `div`-tagen med riktig "id". På linje 6 - 16 ligg JavaScript-koden som opprettar eit enkelt kart samt tileignar kartet til rett `div`-tag.

Eksempel 5.4: Generert JavaScript-kode utan markører eller liknande.

```
1  <script type="text/javascript"
2  src="http://maps.google.com/maps/api/js?sensor=false"/>
3
4  <script type="text/javascript">
5
6  var map;
7
8  function loadMap() {
9    map = new google.maps.Map(document.getElementById("googleMap"), {
10     center: new google.maps.LatLng(0,0),
11     zoom: 1,
12     mapTypeId: 'roadmap'
13   });
14
15 }
16 var t=setTimeout("loadMap()",10);
17
18 </script>
19
20 <div id="googleMap" style="width: 650px; height: 900px; border: #3367A0↩
       1px solid;"></div>
```

Videre er naturlig at kartet skal kunne vise både markører og eit informasjonsvindu for kvar markør. Dette implementerast ved å sette inn ekstra JavaScript-kode. Eksempel 5.5 viser korleis eit slikt skript er, på linje 4 - 15 ser ein korleis ein oppretter ein markør som vidare i linje 17 blir sett til kartet saman med eit informasjonsvindu (`infoWindow`). Informasjonsvinduets innhald blir, som vist på linje 8, laga ved at ein enkelt definere strukturen inni som heilt vanleg HTML-kode. Metoden `bindInfoWindow()` på linje 21 - 28 er metoden som bind alle elementa saman og sett det inn i kartet.

Eksempel 5.5: Enkelt javascript for å legge til markør i kart

```
1
2  ...
3
```

```
4  name = "Bergen";
5  latitude = 60.391263;
6  longitude = 5.322054;
7  point = new google.maps.LatLng(parseFloat(latitude),parseFloat(↩
       longitude));
8  html = '<div><strong>' + name + '</strong></div>';
9
10 var infoWindow = new google.maps.InfoWindow;
11
12 marker = new google.maps.Marker({
13   map: map,
14   position: point,
15   icon: 'http://labs.google.com/ridefinder/images/mm_20_red.png',
16   shadow: 'http://labs.google.com/ridefinder/images/mm_20_shadow.png'
17 });
18
19 bindInfoWindow(marker, map, infoWindow, html);
20
21 ...
22
23 function bindInfoWindow(aMarker, aMap, anInfoWindow, aHtml) {
24   google.maps.event.addListener(aMarker, 'click', function() {
25     anInfoWindow.setContent(aHtml);
26     anInfoWindow.open(aMap, aMarker);
27   })
28 };
29
30 ...
```

Denne implementasjonen gjer da at det vil bli presentert eit kart med ein markør for Bergen. Ideelt sett burde ein kunne satt utsnittet av kartet til å vise ei fin presentasjon av markørar med riktig zoom-nivå. Dette kan ein gjere ved benytte seg av metoden `LatLng()` som ligg i kart API-et. Denne finner ein optimal framstilling av kartet, med hensyn på dei markørane som er lagt inn. Kjeldekoden til denne delen er presentert i eksempel 5.6.

Eksempel 5.6: Enkelt JavaScript for å legge til grensesnitt i kartet

```
1 ...
2
3 bounds = new google.maps.LatLngBounds();
4
5 ...
6
7 ll = new google.maps.LatLng(parseFloat(latitude), parseFloat(longitude)↩
      );
8 bounds.extend(ll);
9 map.fitBounds(bounds);
```

```
10
11  ...
```

Då blir den samansette og endelege JavaScript-koden sjåande ut som i eksempel 5.7. Her blir både markører og riktig utsnitt handtert. I kapittel 5.4 blir det gjennomgått korleis denne metoden er refaktorert.

Implementasjonen av fleire markørar i eit kart er gjennomført ved å legge all JavaScript-kode som er lik for alle markører inn i ein streng. På linje 1 - 22 i eksempel 5.7 ser ein at strengen berre er for sjølve kartet og ikkje har noko med om det skal vere markørar og liknande i kartet. Vidare på linje 23 - 39 kjem den delen av skriptet som gjeld ein markør. Denne delen må naturlegvis komme om og om igjen for kvar markør. Løysinga er då å implementere ei løkke som gjennomgår alle markørar under rotnoden i `FieldPluginBean` og generere denne delen for kvar markør. Resten av skriptet under kan ein legge til strengen etter at alle markører er gjennomgått.

Eksempel 5.7: Javascript for eit basiskart

```
1  <script type="text/javascript">
2
3  var map;
4  var bounds;
5  var name;
6  var latitude;
7  var longitude;
8  var point;
9  var html;
10 var marker;
11 var ll;
12
13 function loadMap() {
14   map = new google.maps.Map(document.getElementById("googleMap"), {
15     center: new google.maps.LatLng(0,0),
16     zoom: 9,
17     mapTypeId: 'roadmap'
18   });
19
20   bounds = new google.maps.LatLngBounds();
21   var infoWindow = new google.maps.InfoWindow;
22
23   name = "Bergen";
24   latitude = 60.391263;
25   longitude = 5.322054;
26   point = new google.maps.LatLng(parseFloat(latitude),parseFloat(↩
         longitude));
27   html = '<div><strong>' + name + '</strong></div>';
28
```

```
29    marker = new google.maps.Marker({
30      map: map,
31      position: point,
32      icon: 'http://labs.google.com/ridefinder/images/mm_20_red.png',
33      shadow: 'http://labs.google.com/ridefinder/images/mm_20_shadow.png'
34    });
35
36    bindInfoWindow(marker, map, infoWindow, html);
37
38    ll = new google.maps.LatLng(parseFloat(latitude), parseFloat(↩
          longitude));
39    bounds.extend(ll);
40    map.fitBounds(bounds);
41  }
42
43  function bindInfoWindow(aMarker, aMap, anInfoWindow, aHtml) {
44    google.maps.event.addListener(aMarker, 'click', function() {
45      anInfoWindow.setContent(aHtml);
46      anInfoWindow.open(aMap, aMarker);
47    })
48  };
49
50  var t=setTimeout("loadMap()",10);
51
52  </script>
```

Det er verdt å nemne at heile dette skriptet ligg i ein streng-variabel. JavaScript-kode har naturlegvis mange felles tegn med Java som må handterast i kjeldekoden. Som i eksempel 5.8, henta frå den private metoden `generateBasicJavascriptStart()` i pluginen, er det blandt anna på linje 6 at ein må settje inn "\" framfor hermetegnet slik at Java-kompilatoren ikkje trur at strengen sluttar der. Dette er eit problem som går gjennom heile skriptet og som ein må være oppmerksom på under heile utviklingsstadiet.

Eksempel 5.8: Javascript for eit basis kart

```
1  String javascriptString = "\n\n"
2
3  + ...
4
5  + "function loadMap() {\n"
6  + "map = new google.maps.Map(document.getElementById(\"googleMap\"), {\↩
       n"
7  + "center: new google.maps.LatLng(0,0),\n" + "zoom: 9,\n"
8  + "mapTypeId: 'roadmap'\n" + "});\n"
9  + "bounds = new google.maps.LatLngBounds();\n"
10 + "var infoWindow = new google.maps.InfoWindow;\n\n";
11
```

```
12 + ...
```

5.3 Innlegging av geodata i DPG

Videre er målet å tilføre funksjonaliteten som kan gje interaksjon i kartet. Ideèn var å la brukar kommentere og setje karakter på ein restaurant/bar som er lagt inn på kartet, slik som for eksempel reiselivsguiden LonelyPlanet har. Men som ein ser i figur 5.8 er det ikkje interaksjon i sjølve kartet, men under kartet i eit enkelt HTML-skjema på sjølve HTML-siden. Ulikt kandidatens mål om å ha slik interaksjon inni informasjonsvinduet for den gitte markøren.

Kandidatens plan var å implementere eit skjema i markørens informasjonsvindu for deretter å lagre til fil, slik pluginressursar blir behandla i den noverande DPG.

5.3.1 Skjemahandtering i Spring

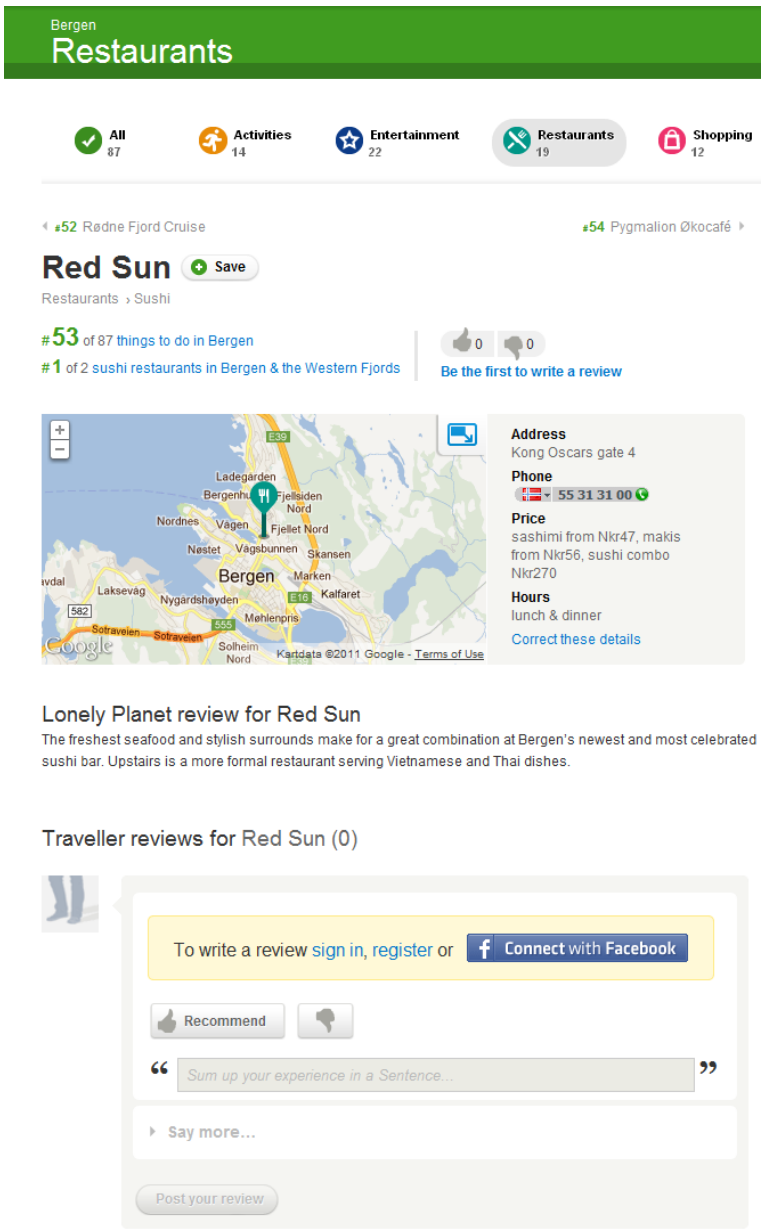
Eit slikt skjema må naturlegvis ha ein handling (Eng. *action*) og slik skjemahandtering fungerer bra i Spring. Ein kan referere til eit HTML-dokument som ikkje eksisterer og handtere denne førespurnaden ved å setje opp ein annotasjon i ei Java klasse. Eksempel 5.9 viser korleis ein kan bruke annotasjonen `@RequestMapping` for å fortelje kva fil som skal handtere den gitte førespurnaden. Vidare kan ein berre setje inn `formhandler.html` i handlinga til skjemaet som eksempel 5.12 viser på første linje. I dette tilfellet er det `PluginFormController` klassen som handterer skjemaet.Som figur 5.9 viser videresender klassen til `presentation.html` som blir behandla i `PresentationController` klassen som har si oppgave å rendre siden til nettlesaren igjen.

Eksempel 5.9: `RequestMapping` annotasjonen i `PluginFormController`

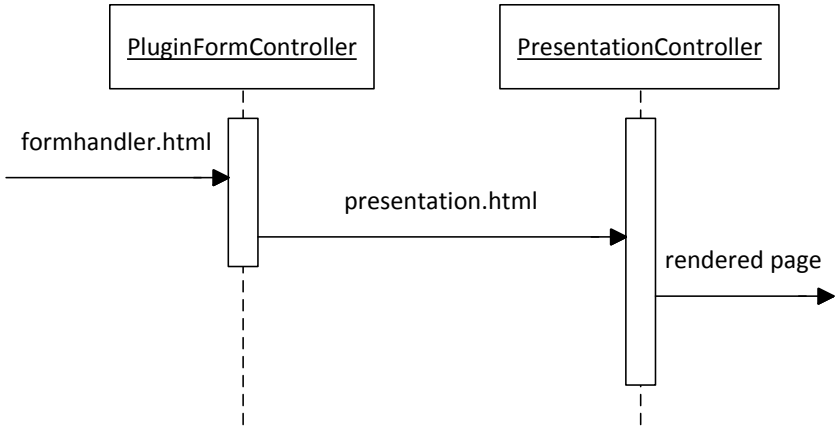
```
1 ...
2
3 @RequestMapping("/pv/formhandler.html")
4 public class PluginFormController {...}
```

5.3.2 Lagring av markør-anmeldelse

Eit anna problem er korleis ein har lagra informasjon frå brukar, gjennom plugin. Denne informasjonen har blitt lagra med eit filnamn som har ein struktur som knyter



Figur 5.8: Lonely Planet sin kommentarfunksjonalitet



Figur 5.9: Skjema handtering med Spring, frå nettlesar og tilbake til nettlesaren

det opp mot;

- kva utsnitt den høyrer til
- kva side den høyrer til
- kva presentasjon den høyrer til

Strukturen på filnamnet er generert på grunnlag av den informasjonen ein treng vite. Slik eksempel 5.10 viser er den første delen av filnamnet presentasjonen den tilhører, andre del er kva side i presentasjonen den tilhører og siste del er kva utsnitt på sida det tilhører. Dette er inspirert av REST nettjeneste [37] og deira måte å bygge opp ein URI (Uniform Resource Identifier) [56]. Etter at utsnittet er definert finn ein alltid ein teller, som berre skal skilje markør-anmeldelser til samme utsnitt frå kvarandre.

Eksempel 5.10: Gammel filnamnstruktur

```
1 dynamicmap_startPage_mapView_0
```

Men dette er ikkje nok informasjon for kartpluginen. For kva viss ein har to markørar i eit kart? Sidan dei markørane har samme plugin, side og utsnitt vil alle markør-anmeldelse tilhøre alle markører med dagens løysing. Ei utviding av filnamnet var då nødvendig. Filnamnet må og innehalde informasjon om kva markør det handlar om.

Den naturlege vegen for å finne ei løysing på problemet var å sjå på kva som er spesifikt for ein plugin, nemleg koordinatane. Det blei løyst ved å setje breddegrad og lengdegrad inn i filnavnet også, slik eksempel 5.11 viser.

Eksempel 5.11: Ny filnamnstruktur

```
1 dynamicmap_startPage_mapView_60.391263_5.322054_0
```

5.3.3 Skjemaet

Skjemaet må altså innehalde ein del informasjon. Denne informasjonen blir lagt inn i skjulte felt, så ein del skjulte variablar må implementeres. I det ferdige skjemaet som er presentert i eksempel 5.12 ser ein på linje 2 - 6 definerer skjulte felt for vidare behandling av brukarens interaksjon.

Eksempel 5.12: Anmeldelse-skjema sin struktur.

```
1  <form action="formhandler.html" method="post" name="formSubmit">
2    <input name="pluginName" type="hidden" value="DynamicMap">
3    <input name="view" type="hidden" value="mapView">
4    <input name="page" type="hidden" value="startPage">
5    <input name="pid" type="hidden" value="dynamicmap">
6    <input name="markerID" type="hidden" value="60.391263_5.322054">
7    <textarea name="review" type="text"></textarea>
8    <select name="rate">
9      <option>1</option>
10     <option>2</option>
11     <option>3</option>
12     <option>4</option>
13     <option>5</option>
14     <option>6</option>
15   </select>
16   <input type="submit" value="Lagre Review">
17 </form>
```

5.3.4 Endringar i pluginen

Som eksempel 5.12 viser på linje 7 - 15 er det lagt til felt for å handtere både ein tekst og ei karaktersetting. Dette blir då den nye funksjonaliteten som pluginen må ha, nemlig handteringa av desse to felta.

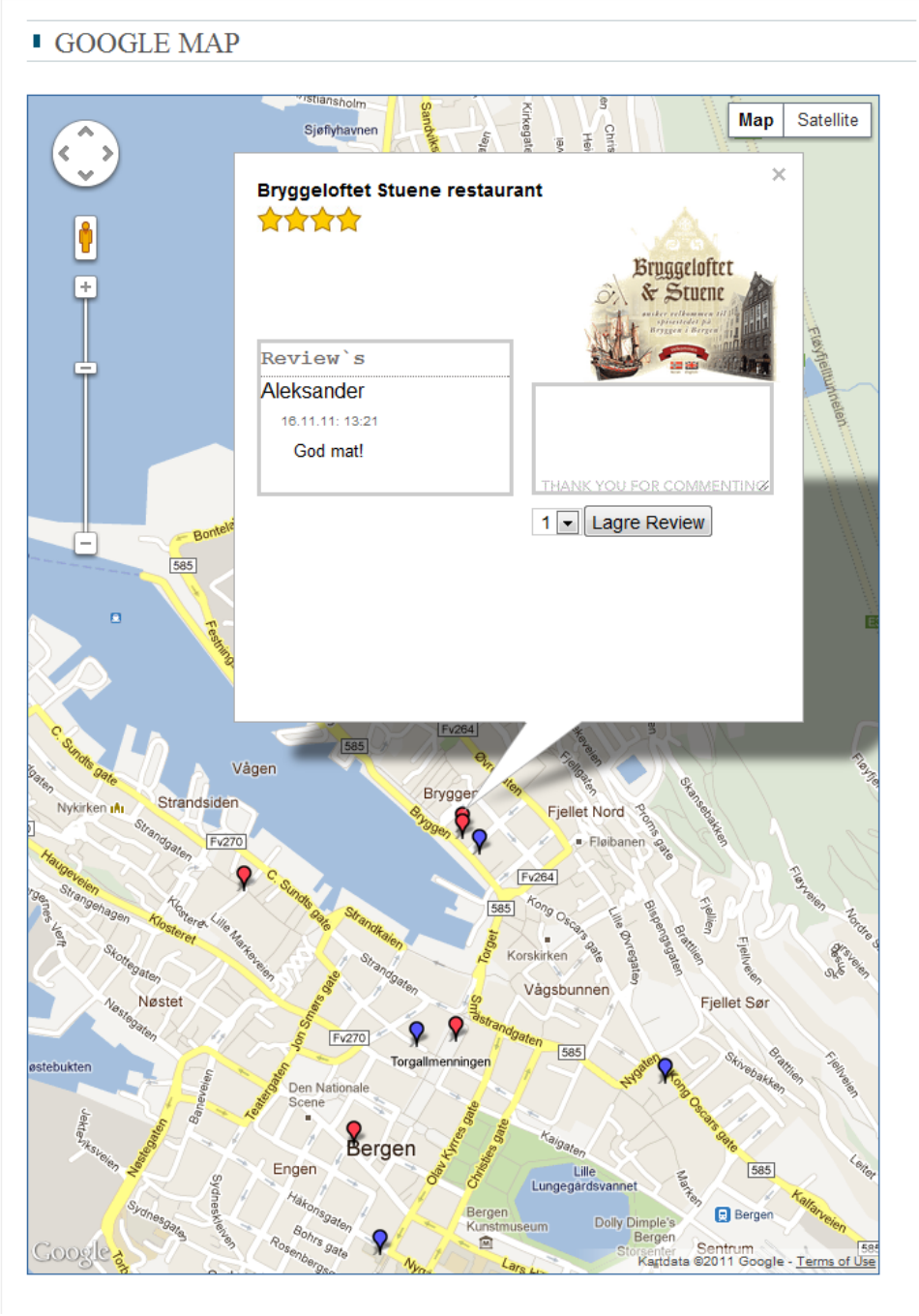
Eksempel 5.13: Den nye `generateElement`-metoden

```
1 @Override
2 public Element generateElement(FieldPluginBean bean, Map<String, Object↩
      > input)
3 {
4   if (input != null && input.get("review") != null) {
5     saveMarkerReviewToFile(bean, input);
6     saveMarkerReviewRateToFile(bean, input);
7   }
8   return generateElement(bean);
9 }
```

Måten dette blei gjort på er å legge til ein ny `generateElement()`-metode. Den har same navn som den andre metoden, men denne skal ha to parametrar. Som ein ser i eksempel 5.13 på første linje, tar den nye metoden inn både `FieldPluginBean` og `Map<String, Object> input`. Dette er fordi den må få inn felta som den skal handtere og. På linje 5 og 6 kallar metoden på to private metodar som tar seg av henholdsvis lagring av kommentaren og lagring av karaktersettinga.

Grunnen til at det er todelt, er fordi kandidaten meiner det ikkje er nødvendig å lagre karaktersettinga i same fil som kommentaren. Grunngivinga er så enkel som at det er meir tungvindt å åpne ei og ei fil, parse den og legge saman alle karakterer for kvar markør kvar gong du skal vise gjennomsnittskarateren i infovinduet, slik figur 5.10 viser. Derfor vil metoden `saveMarkerReviewRateToFile()` berre hente brukerens karakter for så å opne markørens respektive karakterfil og legge til karakteren og lagre filen på nytt. Filens innhald er berre summen av alle karakterar og summen av alle som har gitt karakter, splitta på ein understrek: (”summen av alle stemmer”_“antall stemmer”).

Som ein ser på linje 8 i eksempel 5.13, er returverdien laga ved berre å kalle på den første `generateElement()` metoden, etter at den er ferdig med lagring av både karakter og kommentar. Dermed blir kartet presentert i DPG igjen etter at informasjonen blir lagra.



Figur 5.10: Skjermbilde frå DPG med restaurantar i Bergen

5.4 Refaktoreringa

Gjennom dette delkapitlet blir det presentert ein del kodestank (Eng. *Code Smells*) henta frå Martin Fowler si bok *"Refactoring: Improving the Design of Existing Code"* [18]. Desse kjente problemas løysing er basert på Martin Fowlers forslag. Kandidaten har og deltatt på Karianne Bergs workshop om "Strukturert Refaktorering" [3] som omhandla måtar å refaktorere utan å ha for mykje nedetid samt ikkje miste oversikt og kontroll.

JavaScript-koden som no blir generert i `DynamicMap`-pluginen er blitt rimelig omfattande. Som eksempel 5.14 viser, er det no vanskelig å finne fram i koden som ligg i denne JavaScript-strengen. Dette er ein kodestank som heiter "Long Methods", som baserer seg på å berre ha ei arbeidsoppgåve per metode, noko ikkje `generateElement()`-metoden til `DynamicMap`-pluginen har.

Kandidatens løsying var å la `generateElement()`-metoden handtere sjølve oppretting av skript-tagger samt legge inn sjølve skriptet som innhold i denne taggen. Metoden skal vidare delegere generinga av sjølve skriptet til den private metoden `generateBasicJavascriptString()`. Problemet er no forminska, men JavaScript-koden er enno ein "Long Methods" stank.

Eksempel 5.14: Eit utdrag av koden før refaktorering

```
1  String javascriptString = "var map;\n"
2              + "var bounds;\n"
3              + "var name;\n"
4              + "var latitude;\n"
5              + "var longitude;\n"
6              + "var point;\n"
7              + "var html;\n"
8              + "var htmlFORM;\n"
9              + "var marker;\n"
10             + "var ll;\n"
11
12             + "function loadMap() {\n"
13             + "map = new google.maps.Map(document.getElementById(\"↩
                   googleMap\"), {\n"
14             + "center: new google.maps.LatLng(0,0),\n" + "zoom: 9,\n"
15             + "mapTypeId: 'roadmap'\n" + "});\n"
16             + "bounds = new google.maps.LatLngBounds();\n"
17             + "var infoWindow = new google.maps.InfoWindow;\n\n";
18
19             // Generate markers
20             for (int i = 0; i < bean.getDataElement().getChildren().↩
                   size(); i++) {
21               ...
```

```
22              javascriptString += " name = \""
23              ...
24            }
25
26    javascriptString  += "map.fitBounds(bounds);"
27            + "}\n"
28            + "function bindInfoWindow(aMarker, aMap, anInfoWindow, ↩
                  aHtml) {\n"
29            + "google.maps.event.addListener(aMarker, 'click', function↩
                  () {\n"
30            + "anInfoWindow.setContent(aHtml);\n"
31            + "anInfoWindow.open(aMap, aMarker);\n" + "})\n" + "};\n"
32            + "var t=setTimeout(\"loadMap()\",10);\n";
```

Tre private houedmetodar blei trekt ut av `generateBasicJavascriptString();`

- `generateBasicJavascriptStart()`
- `generateBasicJavascriptEnd()`
- `generateMarkers()`

Namna på desse fortel kva metoden gjer, og dei blei laga for å organisere genereringa av JavaScript-koden betre, samt redusere kodestank. Kandidaten har heile tida laga skriptet på ein slik måte at det skal være mogleg å ta vekk funksjonalitet som interaksjon og bildevisning. Det var defor eit stort poeng å dra dette ut i eigne private metoder, slik at nye utviklare lett skal kunne ta vekk slik funksjonalitet utan bekymringar for at det skal knekke skriptet. Følgjane private metoder blei trekt ut av `generateMarkers();`

- `getRateForReviews()`
- `getReviews()`
- `generateJavascriptReadyForm()`
- `getImageForMarker()`

Som eksempel 5.15 gjorde refaktoreringen metoden `generateElement()` med over 500 linjer uoversiktlig kjeldekode om til ein 20 linjers-metode med ei organsiering som var mogleg for ein utviklar å lese. Men det finst ei anna stank i oppgåva og. Denne blir definert av to kodestanker.

- Duplisert kode (Eng. *Duplicate Code*)
- Kode som ofte blir brukt i saman (Eng. *Data Clumps*)

Eksempel 5.15: Den refaktorerte metoden `generateElement()`

```
1   @Override
2   protected Element generateElement(FieldPluginBean bean) {
3
4     // Generating the javascript element
5     Element javascript = new Element("script").setAttribute("type",
6         "text/javascript");
7
8     // Generate the DIV element for the googlemap
9     Element element = new Element("div")
10        .setAttribute("id", "googleMap")
11        .setAttribute("style",
12            "width: 650px; height: 900px; border: #3367A0 1px solid;");
13
14    // Putting the javascript into the javascript element.
15    javascript.addContent(generateBasicJavascriptString(bean));
16
17    // Putting everything into the root-element and returning it.
18    bean.getRootElement().addContent(javascript);
19    bean.getRootElement().addContent(element);
20
21    return bean.getRootElement();
22  }
```

Dette gjeld i utgangspunktet markørar slik eksempel 5.16 viser. Det blir generert ein ID frå desse når ein hentar markør-anmeldelser og karakterar frå filsystemet. Ein må då setje inn både lengde og breiddegrad for markørane og lage denne ID-en i begge metodar.

Eksempel 5.16: Lengde og breiddegrad blir brukt fleire gongar.

```
1 ...
2 getRateForReviews(bean,aMarkerEntity.getChildText("latitude"),↩
      aMarkerEntity.getChildText("longitude"))
3 ...
4 getReviews(bean,aMarkerEntity.getChildText("latitude"),aMarkerEntity.↩
      getChildText("longitude"))
5 ...
```

Løysinga var å opprette endå ein privat metode kalla `getMarkerId()` som tar ein markør som parameter og generer ID-en slik at metoden no berre treng å ta inn sjølve ID-en. Eksempel 5.17 viser at ein då berre bruker eit metodekall til denne metoden som input. Gevinsten er ein mindre parameter, samt at koden ikkje blir duplisert i begge metodane.

Eksempel 5.17: Løysing på bruk av lengde og breiddegrad.

```
1 ...
2 getRateForReviews(bean, getMarkerId(aMarkerEntity))
3 ...
4 getReviews(bean, getMarkerId(aMarkerEntity))
5 ...
```

Etter refaktorering var kjeldekoden mykje meir oversiktleg og mindre komplisert. Metoden er redusert med fleire hundre linjer og kandidaten er sikker på at ein ny utviklar no skal kunne setje seg raskt inn i pluginen. Utviklarer skal og kunne velje kva kartfunksjonalitet som skal vere med ved å inkludere/ekskludere nokon private metodar frå genereringa av kartet, samt gi pluginen ny funksjonalitet utan at kartets skript knekk.

6

Evaluering, konklusjon og vidare arbeid

6.1 Evaluering av mål

Det overordna målet i oppgåva var å gje DPG støtte for geodata, men for å komme til dette målet blei det delt opp i fleire delmål;

- Vurdere pluginarkitekturen
  - Er dagens løysing tilrettelagt for kartimplementasjon?
- Forbedre pluginarkitekturen
  - Gjere endringar slik at kartfunksjonaliteten kan implementeres.
- Vurdere kart-tjenester
  - Vurdere 3 API-er og vurdere kva som passar best til dette formålet, samt denne implementasjonen.
- Presentere Geodata
  - Implementere kartet frå valgt kartteneste samt handtere innhaldet som skal ligge i kartet.
- Innlegging av Geodata

– Gje kartet funksjonar som å kommentere og karaktersette markørar.

Dei første tre punkta er løyst i samarbeid med Kelly Alexander Teigland Whiteley [50]. Under utviklinga av ein enkel kartplugin blei det oppdaga ein god del grove svakheiter i systemet og det blei satt i gong ei omfattande omstrukturering av pluginarktitekturen. I tillegg blei det lagt til ein ny og forenkla måte å referere til plugins på i presentasjonsspesifikasjonen, då begge kandidatane var einige om at den noverande løysinga var unødvendig tung. Detaljar om dette blei gjennomgått i kapittel 3 og 4.

Kandidaten er veldig fornøgd med resultatet av fellesarbeidet, då dette har økt moglegheiten for å utvikle meir funksjonalitet til DPG. Som i utgangspunktet betyr at det no er mogleg å ha større og mer avanserte plugins med tanke på innhald ein plugin no kan få tilgang til.

Dei tre siste punkta som omhandlar kart og kartimplementasjon blei gjennomgått i kapittel 5 og omhandlar utviklinga av pluginen. Her er og eit eige delkapittel om ein omfattande refaktorering av pluginen. Kandidaten har utvikla ein kart-plugin som både brukar innhald frå PCE samt kan legge til og bruke innhald frå sine eigne ressursar. Dette er den første pluginen i DPG som aktivt tar inn data (kommentarer til kvar markør akkurat i denne pluginen) frå brukar, samtidig som ein kan legge til innhald i PCE som forandrar presentasjonen.

Kandidaten har laga ein annan plugin som berre viser markørar henta frå PCE, men som viser strekninga mellom markørane i staden for å la bruker kommentere markørar. Dette viser at ein utviklar kan bruke same plugin som eit utgangspunkt til å lage ein annan kart-plugin. Det treng berre små forandringar for å bruke den til heilt andre formål.

Kandidaten er og veldig fornøyd med at alle val er gjort med tanke på at andre utviklare skal kunne gjere slike endringar og, utan for store utfordringar. Kandidaten har systematisk gått gjennom alle delmåla for å nå det overordna målet om å gi DPG støtte for geodata. Kandidaten er fornøyd med resultatet av denne oppgåva.

6.2 Vurdering av teknologiar

6.2.1 Trac

Trac er eit *problemsporingssystem* (Eng. *Issue Tracking System*). Med det meines at det skal fungere som eit verktøy i planleggingsprosesen av ei utvikling. Vidare i

utviklinga brukes det til å kartlegge framgang, samt spore opp kvar det måtte vere forbetringspotesialet.

Kandidaten brukte Trac som eit framdriftsverktøy for både ein eigen del og sine rettleiarar. Erfaringen med Trac er at i samanheng med oppgåva er Trac litt overdimensjonert.

Trac bruker Milestones og Tickets for å handtere prosjektframgongen. Milestones er tidbestemt og Tickets er alle oppgaver i ein Milestone. Men for eit prosjekt av oppgåvas størrelse var det naturleg å og ha ein frist på Tickets og bruke Milestones som eit overordna samlingspunkt for Tickets. Eit eksempel på dette kan være at to Milestones med underliggande Tickets er;

- Implementasjon Ferdig
  - Implementere lesing frå fil
  - Implementere skriving til fil
- Oppgåveskriving Ferdig
  - Skriver ferdig kapittel 2
  - Skriver ferdig kapittel 3

Kandidaten ser heilt klart fordelar med å bruke Trac for større prosjekt for organisering av kven som skal jobbe på kva og ikkje minst kva som er ferdig. Trac vil fungere bra som ein framdriftsplan og vil vere ei enkel løysing for å finne ut kva delar av prosjektet som tar meir tid enn forventa. Det er etterkvart og mogleg å gi eventuelle "kundar" eit betre tidsperspektiv ettersom ein drar erfaring frå kva som tar tid. Kandidaten gjorde dette ovenfor sine rettleiarar når det blei oppdaga at skriving på oppgåva tok lenger tid enn forventa.

6.2.2 Maven

I begynnelsen av kandidatens arbeid på DPG brukte systemet Ant [35] som utrullingsverktøy (Eng. *Deployment tool*). I 2010 blei Ant erstatta med Maven [1], som til samanlikning er strengare på bland anna mappestruktur. Maven krever at alle avhengigheiter i prosjektet blir definert i ei *Project Object Model* (POM) fil, kalla `pom.xml`.

Alle bibliotek som eit prosjekt er avhengig av, blir automatisk lasta ned i eit lokalt arkiv (Eng. *repository*). Første gongen du starter ei utrulling på ei maskin. Fordelen

med dette er først og fremst at ein utviklar ikkje treng å forhalde seg til alle biblioteka eit prosjekt treng. Men om eit prosjekt bruker same bibliotek som eit anna, kan dei no dele på desse i staden for å ha kvar sine bibliotek i prosjektet.

Kandidaten merka ei betydleg forenkling i utrullinga av prosjekt etter overgangen til Maven. Det var eit klart løft i utrullingsprosessen som gjekk i frå eit tidkrevande arbeid til arbied som kun innehaldt heilt nødvendige konfigurasjonar.

m2e - Maven intigrering i Eclipse

Eclipsepluginen "m2e" gjer det mogleg å styre heile Maven utrullinga direkte frå Eclipse. Dette er noko som gjer det enklare for utviklarar å testkjøre den siste funksjonaliteten som er implementert. Du kan kjøre dette med din egen valfrie server akkurat som om du kjører det fra kommandolinja. Kandidaten har kjørt det gjennom Tomcat [40] og Jetty [28] og anbefaler sistnemnde til testkjøring og debugging. Jetty er kjappare å starte og meint som ein testkjøringserver for midlertidig utrulling av systemet.

6.2.3 Google Maps API

Google Maps API var kandidatens valg av kart implementasjon, sjølv om det finst andre gode karttenester som pustar Google Maps i nakken, er det den mest utbredte karttenesten som finst for øyeblikket.

Google Maps API-et er veldig godt dokumentert samt at det eksisterer veldig stort utval eksempel på forskjellige løysingar på implementasjon. Kandidaten blei overraska over at det finst veldig gode konkurrentar til Google Maps, men er fornøgd med sitt val av karttjeneste. Kandidaten meiner det vil være ein fordel for vidare utvikling at den best dokumenterte og mest tilgjengelege karttenesten er valt.

6.2.4 Andre teknologiar

Det er brukt fleire teknologiar enn dei overnemnde i denne oppgåva. Dette delkapittelet vil gje ein enkel gjennomgang av dei resterande teknologiane;

- Svn
  - Eit heilt nødvendig versjoneringsverktøy, brukt både under utvikling og under skriveprosessen.

- Tomcat
  - Web Server for å kjøre Java-kode, brukt for utrulling av DPG.
- Firebug
  - Firefox utviding for feilsøking i blandt anna JavaScript i nettsider, nyttig i samanheng med utviklinga av JavaScript-kartet.
- LaTeX
  - Eit typesettingssystem for dokumentproduksjon med ei rekke pakker som handterer innhaldliste og liknande heile denne oppgaven er skrive i LaTeX.

6.3 Vidare arbeid

6.3.1 Utvide interaksjonen i Kart-plugin

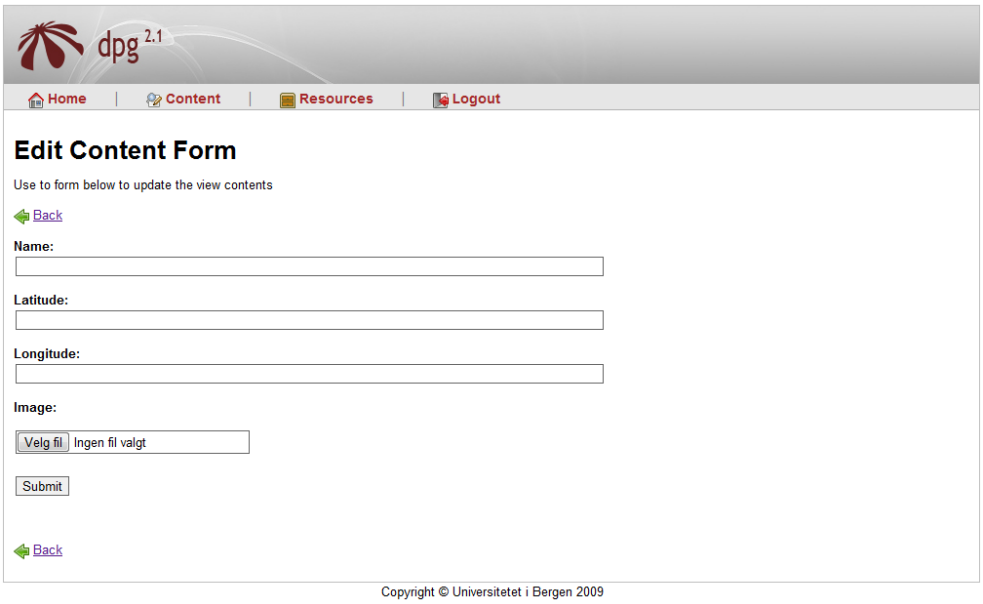
Kartpluginen har no funksjonar som å gje markørar karakter og anmeldelse. I vidare utvikling kunne det vore svært nyttig å la brukaren også definere sine eigne markørar/geodata. Eksempel på dette kan være brukeren sine:

- Favoritt spisestader
- Reisemål
- Families bostader
- Treningsturar

Dette bør vere mogleg å implementere i form av et enkelt klikk i kartet som resulterte i at geodata blir lagt inn i filsystemet, på lik linje med karaktersettinga og kommentarfeltet i den noverande pluginen. Dette burde ikkje være ein veldig tidkrevande jobb å implementere ettersom det er lagt tilrette for brukarinteraksjon allereie, og vil gje DPG og pluginen ytterligare funksjonalitet og bruksområde.

6.3.2 Utvide pluginkontrakten ytterligare

Pluginkontrakten blei utvida med ein ny metode for å sjølv bestemme strukturen på innhaldet pluginen skal ha inn. Vidare kunne kontrakten og utvidast med ein ny metode som definerer korleis PCE skal presentere innleggingen av data. Som figur



Figur 6.1: Eksempel på korleis ein PCE presenterar innlegging av geodata til kart-pluginen i dag.

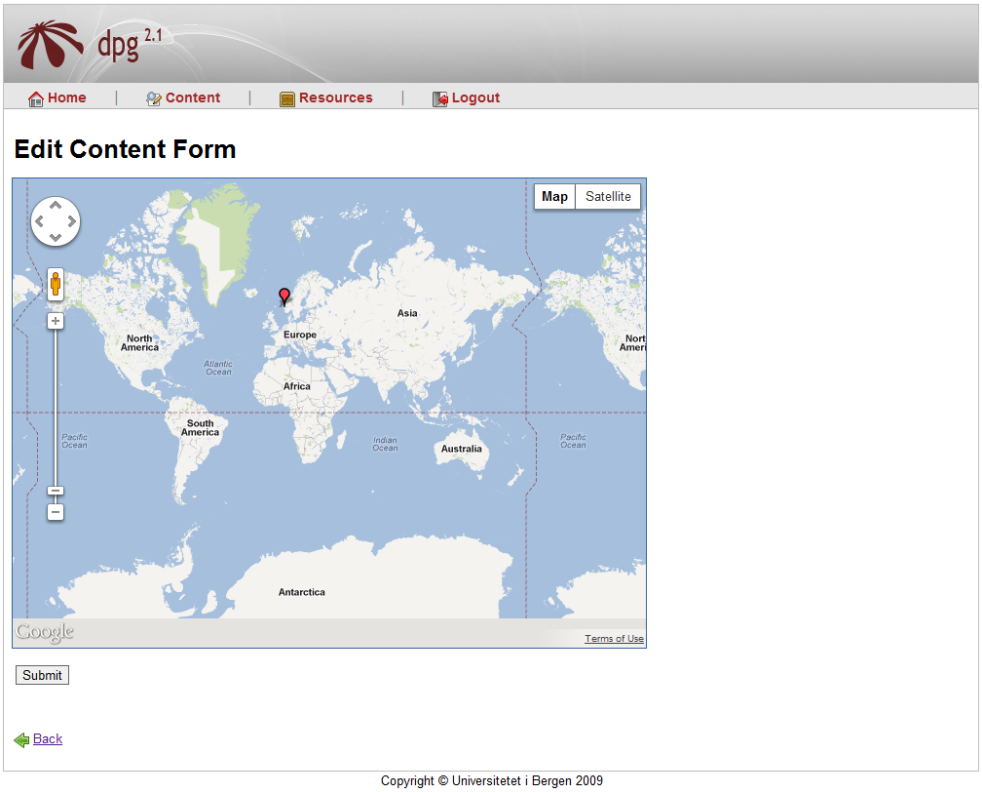
6.1 viser er det berre eit skjema som blir vist, og du må sjølv legge inn lengde og breiddegrad manuelt.

Eit meir brukarvenleg alternativ er presentert i figur 6.2, der kan ein berre klikke på kartet i det punktet ein vil ha markøren, og så kan det eventuelt komme opp eit vindu der ein fyller inn informasjon i tillegg til lengde og breiddegrad som blir plukka ut i frå brukarens klikk.

6.3.3 Oppdatere "gamle" plugins

No som den nye metoden `generatePatternStructure()` er lagt til i DPG bør ein oppdatere dei gamle pluginene slik at alle bruker denne metoden. Dette vil automatisk gjere mønsteret enklare og meir brukarvenleg for ein mønsterutviklar.

Det er klart at plugins som handterer felt som `string` og liknande ikkje nødvendigvis vil gjere mønsteret enklare, men det vil invitere nye pluginutviklare til å følgje same metodar for å lage ny plugin og ikkje minst gjere kommunikasjonen mellom plugin og resten av systemet meir strømlinjeforma.



Figur 6.2: Illustrasjon på korleis ein PCE kunne presentert innlegging av geodata til kart pluginen i framtida.

6.3.4 Utvide funksjonaliteten i DPG

Kandidaten meiner det kan være fordelar ved å utvide funksjonaliteten til også å handtere andre typer felt enn `string`. DPG bør og kunne handtere felt som for eksempel mail-adresse og telefonnummer o.l.

Om ein allereie hadde gjort dei nødvendige implementeringane i plugin-kontrakten slik delkapittel 6.3.2 diskuterer, kunne ein og i presentasjonen av desse typane evaluert dei med eit skript før dei blei godkjend for å lagras i PCE.

6.3.5 Vidareutvikle plugin sin ressurshandtering

I masteroppgåva til Kelly Alexander Teigland Whiteley [50] blir probematikken rundt plugin sin ressurshandtering grundig gjennomgått. Oppgava hans omhandlar ein implementasjon av Java Persistence API (JPA) [55] som handterer alle ressursar som handteres i plugins. Dette vil gjere det enklare å utvikle ein plugin som bruker ressursar. Både i sjølve utviklinga av pluginen samt og når pluginen skal hente/lagra til og frå fil.

Med filstruktur som einaste alternativ for lagring av ressursar i dag er det tungt for ein plugin å både hente og lagre informasjon. Kandidatens erfaring med ressurshandteringa for plugins i dag er at når det kjem til større kart med mange markørar går det veldig seint å laste kartet. Kandidaten er overbevist om at Kelly Alexander Teigland Whiteley si løysing er eit mykje bedre alternativ enn dagens løysing. Kandidaten er heilt sikker på at kart-pluginen vil bli mykje raskare om den hadde brukt denne løysinga.

6.4 Konklusjon

Denne masteroppgåva presenterer utvikling av DPG, eit stort og komplekst system. Den omhandlar både individuelt arbeid og arbeid i saman med andre utvkilare på same system.

Kandidaten har hatt store utfordingar ved å setje seg inn i eit så stort system. Det har vore veldig interessant å utvikle på kjernefunksjonaliteten til DPG då dette krevde forståelse av heile systemet. Det er ei stor utfordring å setje seg inn i så store mengder kjeldekode og dokumentasjon over ein relativt kort periode. Det har blitt tatt i bruk fleire metodar og teknikker som kandidaten har fått god erfaring med. Metodar som feilsøking ved bruk av Log4j [14] og kartlegging av dataflyt ved bruk av av UML [23], har vist seg å være gode teknikker og metoder for kandidaten.

I sammanheng med utviklinga av kartpluginen har det blitt vurdert fleire kart API-er. Google Maps API-et som blei valgt har vist seg å vere eit rett valg for ei slik implementering, samt eit godt alternativ til det formålet kandidaten skulle bruke kartet til. Kandidaten har fått god erfaring med vurdering av slike API-er og kva ein skal legge vekt på i slike sammenlikningar.

Eit stort poeng for kandidaten var å gjere pluginane sine enkle for nye utviklarar å bruke/vidareutvikle. I denne samanhengen har kandidaten sete seg inn i ein del kodestankar, samt deltatt på ein workshop for refaktorering. Kandidaten har refaktorert pluginane slik at dei skal vere lette å setje seg inn i, samt forandre på. Kandidaten lærte mykje om kodestank og kodekvalitet som han vil ta med seg vidare.

Kandidaten har i samarbeid med Kelly Alexander Teigland Whiteley [50] parprogrammert i store delar av fellesdelen. Dette har gitt kandidaten ei god erfaring i forhold til å jobbe så tett med andre utviklarar.

Kandidaten, som er relativt ferske i utviklarrolla, har laga nokon forskningspørsmål som nemd tidlegare i oppgåva. Spørsmåla har gitt kandidaten ein peikepinn på korleis ein skal tenkje for å komme fram til dei gode løysingane i eit så omfattande system som DPG.

Kandidaten har sett stor verdi av arbeidet på oppgåva. Det har blitt brukt mange populære teknologiar og rammeverk, noko som kjem til å styrke kandidatens kvalifikasjonar i hans vidare karriere, samt gi kandidaten ein enklare inngong til arbeidslivet.

Litteratur

[1] Apache. Welcome to apache maven. http://maven.apache.org/. Accessed 2011.10.14.

[2] Drupal Association. Drupal. http://drupal.org/. Accessed 2011.12.16.

[3] Karianne Berg. Strukturert refaktorering. Workshop på Smidig 2011 konferansen.

[4] Karianne Berg. Persistensproblematikk i Dynamic Presentation Generator. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2008.

[5] Microsoft Corporation. Bing Maps. http://www.bing.com. Accessed 2012.01.02.

[6] Microsoft Corporation. Microsoft visio. http://office.microsoft.com/en-us/visio. Accessed 2011.10.14.

[7] Satellite Imaging Corporation. Geoeye 2. http://www.satimagingcorp.com. Accessed 2011.12.06.

[8] Satellite Imaging Corporation. Satellite imaging corporation. http://www.satimagingcorp.com/satellite-sensors/geoeye-2.html. Accessed 2011.12.06.

[9] Kevin Cruickshanks. Verktøy for generering av XML-baserte presentasjonar: JPGen - Java presentasjons generator. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2004.

[10] Yngve Espelid. Dynamic Presentation Generator. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2004.

[11] Mozilla Foundation. Firebug. http://getfirebug.com. Accessed 2011.11.14.

[12] Mozilla Foundation. Javascript. https://developer.mozilla.org/en/JavaScript. Accessed 2011.07.25.

[13] Mozilla Foundation. Mozilla firefox. http://www.mozilla.com. Accessed 2011.12.14.

[14] The Apache Software Foundation. log4j. http://logging.apache.org/log4j/. Accessed 2011.12.15.

[15] The Apache Software Foundation. Subversion. http://subversion.apache.org/. Accessed 2011.12.15.

[16] The Apache Software Foundation. The Apache Velocity Project. http://velocity.apache.org. Accessed 2011.12.06.

[17] The Eclipse Foundation. Eclipse. http://eclipse.org. Accessed 2011.12.10.

[18] Martin Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional, 1999.

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

[20] Google. Google chrome. http://www.google.com/chrome. Accessed 2011.10.14.

[21] Google. Google drawings. http://www.google.com/google-d-s/drawings/. Accessed 2011.08.09.

[22] Google. Google maps api dokumentasjon. http://code.google.com/intl/nn/apis/maps/index.html. Accessed 2011.12.06.

[23] Object Management Group. Unified Modeling Language (UML). http://www.omg.org/spec/UML/. Accessed 2011.12.15.

[24] Jason Hunter. Jdom. http://www.jdom.org/. Accessed 2011.11.28.

[25] Morten Høyland. Datainnsamling med XForms i Dynamic Presenation Generator. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2010.

[26] Bjørn Ove Ingvaldsen. Multimedia i dynamisk presentasjons generator 2.0. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, September 2008.

[27] JAFU. JAva for FjernUndervisning. http://nettkurs.uib.no/. Accessed 2010.07.06.

[28] Jetty. Maven jetty plugin. http://docs.codehaus.org/display/JETTY/Maven+Jetty+Plugin. Accessed 2011.12.12.

[29] Robert C. Martin. The Open-Closed Principle. *C++ Report*, 1996. Accessed 2011.12.12.

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

[31] Robert C. Martin. *Clean Code - A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009.

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

[33] Nokia. Nokia Maps. `http://maps.nokia.com`. Accessed 2012.01.02.

[34] Tobias Rusås Olsen. Interaksjon og søk i Dynamic Presentation Generator. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2010.

[35] The Apache Ant Project. Apache. `http://ant.apache.org/`. Accessed 2011.12.16.

[36] LaTeX3 project team. Latex project. `http://www.latex-project.org/`. Accessed 2011.09.12.

[37] Alex Rodriguez. Restful web services: The basics. `https://www.ibm.com/developerworks/webservices/library/ws-restful/`, 2008. Accessed 2011.12.06.

[38] Øystein Lund Rolland. Integrasjon av Orbeon Forms Designer i Dynamic Presenation Generator. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2010.

[39] Bjørn Christian Sebak. Dynamic Presentation Generator 2.0 – Utvikling av ny dynamisk presentasjonsgenerator og presentasjonsmønsterspesifikasjon. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2008.

[40] Yoav Shapira. Apache tomcat 6.0. `http://tomcat.apache.org/tomcat-6.0-doc/developers.html`, 2011. Accessed 2011.12.11.

[41] Peder Lång Skeidsvoll. Støtte for rike klienter i DPG. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2010.

[42] Edgewall Software. Trac. `http://trac.edgewall.org/`. Accessed 2011.12.15.

[43] Spring. Spring - web mvc framwork. http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html. Accessed 2011.12.06.

[44] Spring. Spring framework. http://www.springsource.org/.

[45] Spring. Spring security. http://static.springsource.org/spring-security/site/. Accessed 2011.09.18.

[46] Joomla Leadership Team. Joomla. http://www.joomla.org/. Accessed 2011.12.16.

[47] UiB. Uib webpage. http://www.uib.no. Accessed 2011.12.06.

[48] W3C. HTML 4.01 Specification. http://www.w3.org/TR/REC-html40/. Accessed 2011.09.21.

[49] World Wide Web Consortium (W3C). XSL Transformations (XSLT) - Version 1.0. http://www.w3.org/TR/xslt, 1999. Accessed 2011.12.06.

[50] Kelly Alexander Teigland Whiteley. Resource management for plugins in the Dynamic Presentation Generator. Masteroppgave, Institutt for Informatikk, Universitetet i Bergen, 2011.

[51] Wikipedia. Application programming interface. http://en.wikipedia.org/wiki/Application_programming_interface. Accessed 2011.12.06.

[52] Wikipedia. Cascading style sheets. http://en.wikipedia.org/wiki/Cascading_Style_Sheets. Accessed 2011.09.13.

[53] Wikipedia. Data access object. http://en.wikipedia.org/wiki/Data_access_object. Accessed 2011.12.06.

[54] Wikipedia. Jar (file format). http://en.wikipedia.org/wiki/JAR_(file_format). Accessed 2011.12.06.

[55] Wikipedia. Java persistence api. http://en.wikipedia.org/wiki/Java_Persistence_API. Accessed 2011.12.16.

[56] Wikipedia. Uniform resource identifier. http://no.wikipedia.org/wiki/Uniform_Resource_Identifier. Accessed 2011.12.06.

[57] Wordpress. Wordpress. http://wordpress.org/. Accessed 2011.08.12.