HOVEDPROSJEKT:

IBM

Business Process Modelling prototype engine using Web Services and Flow language

FORFATTERE:

Frode Mangseth
Stein Tore Tøsse

Dato: 23/5-2002

SAMMENDRAG AV HOVEDPROSJEKT

Tittel:	Norsk: IBM - Prototyp for Web Services	Nr. :
	Engelsk: Business Process Modelling prototype engine using WebServices and Flow Language	Dato : 23/5-2002
Deltakere:	Frode Mangseth Stein Tore Tøsse	
Veileder:	Frode Haug	
Oppdragsgiver:	IBM Hamar	
Kontaktperson:	Jørn B Nordlund	
Stikkord	Innovativ, forretningsmodeller, tjenester, rammeverk, Internett	

Antall sider: 51	Antall bilag: 4	Tilgjengelighet (åpen/konfidensiell): Åpen

Kort beskrivelse av hovedprosjektet:

I dagens samfunn blir det stadig viktigere å kunne tilby tjenester på Internett, og ikke minst det å kunne bruke dem. E-handel, handel over Internett, blir for mange mer og mer vanlig. Dette gjør at det stilles strenge krav til metodene som benyttes, og standardiserte måter å gjøre dette på. Flere firmaer blant annet Microsoft, HP, Sun og IBM har laget sine forslag som har endt opp i en felles W3C standard som kalles Web Services. Rammeverket er utarbeidet med tanke på å være enkelt og fleksibelt, men samtidig robust og plattformuavhengig.

En del av vårt prosjekt var å sette oss inn i dette rammeverket og prøve det ut i praksis, altså lage en del Web Services.Vi ville også finne ut om det var mulig å bruke ”gammelt” verktøy til å modellere interaksjonen mellom ulike tjenester. Modellen kjøres igjennom vår prototyp som representerer tjenester som Web Services. En selger skal for eksempel kunne dra ut til en kunde og modellere en skisse for hvordan forretningsprosessen vil være. Deretter kan han bruke vår prototyp til å vise hvordan de ulike Web Servicene vil samhandle i praksis. Denne samhandlingen krever også en standard, men har enda ikke blitt en W3C standard. En del av vår oppgave har derfor vært å se på IBM sitt forslag til samhandling, WSFL, og bruke dette som utgangspunkt for deler av vårt prosjekt. Løsningen vi har laget er laget i den hensikt å sjekke om ting fungerer. Dersom videre utvikling er ønskelig er det gode muligheter for dette.

FORORD

Det siste halvåret vårt har vært veldig lærerikt og spennende. Vi har jobbet mye for å nå de mål vi hadde satt oss. Vi føler at vi kom i mål, selv om det ikke ble så mye programmering som vi hadde trodd. Gjennom hele prosjektet har vi vært nødt til å lære oss nye ting. Disse tingene håper vi vil være nyttige å ha med seg i fremtiden når vi skal søke jobb. Vi synes det har vært spennende å løse noe ingen har gjort før oss. Derfor håper vi at løsningen vi har kommet frem til blir arbeidet videre med. Vi har lært mye om prosjekt arbeid og planlegging, noe vi forhåpentligvis får bruk for videre. Vi føler at hovedprosjektet har vært en god avslutning på 3 år på høgskolen.

Vi vil takke veileder Frode Haug for hjelp med rapporten. I tillegg vil vi takke Jørn Nordlund for hans oppriktige interesse og velvilje til å hjelpe oss. Jørn har under hele prosjektet holdt motivasjonen vår oppe. Gustav Bilben fortjener takk for hjelp til tekniske spørsmål. Og en takk til Anett Bigsett fordi hun var behjelpelig med å lese gjennom rapporten vår underveis og med plakaten.

Gjøvik den 23/5-2002

_____ _____

Frode Mangseth Stein Tore Tøsse

1 Innledning

1.1 Problemområde

Den siste tiden har flere selskaper gått sammen om å lage en standard som skal brukes til å definere koreografien mellom de forskjellige tjenestene, her representert ved Web Services, i en forretningsprosess. Kort forklart er en Web Service en applikasjon som ligger tilgjengelig på internett. Web Service kan være en applikasjon som ved et gitt personnummer returnerer data om en person. En mer nøyaktig forklaring er gitt i kapittel 2. En forretningsprosess vil her si for eksempel en beskrivelse av hvordan man kan registrere en bil, i hovedtrekk hvilken rekkefølge ting skal skje i. En forretningsprosess kan bruke en eller flere tjenester, i denne oppgaven Web Services. Det finnes i dag få verktøy som genererer den nye typen beskrivelsesspråk (WSFL - som er standard hos IBM). Inntil større verktøy blir laget trengs et verktøy som lager en prototyp av en forretningsmodell, denne prototypen parser FDL og mapper til WSFL og lager en presentasjon av dette, slik at selgere kan gå ut og enkelt vise hvordan en forretningsprosess blir representert. Teknologien for dette finnes, men det trengs en prototyp som løse dette ved bruk av Web Services. Dette har ført til at man ønsker å enkelt kunne lage en representasjon av en forretningsprosess, slik at selgeren kan enklere vise fram hvordan en tjeneste faktisk vil fungere. Slik det er nå bruker selgere altfor mye tid til på å lage en representasjon av forrentingsmodeller til en kunde. En slik prototyp kan også fungere som brukseksempel når WSFL skal standardiseres.

1.2 Avgrensing

Oppgaven går ut på at vi skal lage en prototypmotor, det vil si at dette prosjektet er et testprosjekt for å se hvilke teknologier som kan brukes og hvordan de kan brukes. Det finnes utrolig mye interessant og lærerikt innenfor Web Services, vi avgrenser dette med å lage en prototyp som skal i all hovedsak fungere på de WSFL vi generer ut fra de regler vi har satt opp. For å unngå at oppgaven blir altfor stor har vi satt oss som mål å lage 2 scenarioer som viser hvordan det hele fungerer. Det er ikke satt noen krav til sikkerhet eller robusthet for prototypen, det kravet som er satt er at den skal fungere under fremføringen av prosjektet. Prototypen skal fungere slik at den kan jobbes videre med senere og/eller være en veiledning for videre arbeid innen fagområdet. De skjermbilder som genereres av prototypen skal være intuitive slik at en eventuell kunde skjønner hva som foregår og

hvordan dette i all hovedsak fungerer. Parseren som skal lages skal fungere utifra de regler vi setter opp.

1.3 Oppgavebeskrivelse

Etter å ha blitt forelagt de aktuelle hovedprosjektene her på Høgskolen i Gjøvik for 2002, valgte vi det prosjektet som virket som en bra utfordring for oss, hvor vi kunne lære mye nytt. Det at en av gruppe medlemmene allerede kjente litt til vår oppdragsgiver var også et viktig moment. Vi visste at de var engasjert og ville hjelpe oss så mye som mulig. Oppgaven går ut på å lage en prototypmotor for forretningsmodeller ved bruk av Web Services som kan videreutvikles og brukes av bedrifter (i hovedsak IBM) til presentasjon hos kunder. Prototypen skal ved hjelp av en parser og et program som modellerer forretningsprosesser generere skjermbilder som viser gangen i bruken av flere Web Services sammen. Oppgaven har så blitt delt opp i noen hovedpunkter:

- Sette oss inn i Web Services og verktøyene vi skulle bruke fra IBM, WebSphere Application Developer[21] og BPM Workbench. Lage to Web Services som bruker database og teste ut hvordan disse fungerer.
- Lage en parser fra ”Flow Definition Language” – FDL til ”Web Services Flow Language” – WSFL og definere kravene som stilles til modellering i BPM Workbench for at vi skal få en gyldig WSFL
- Lage en applikasjon som kommuniserer og kjører Web Services basert på dens ”Web Service Description Language” – WSDL fil.
- Lage en tilstandsmaskin som får alt dette til å fungerere sammen som en prototyp.

1.4 Formål

Teknologien som blir brukt i prosjektet er veldig ny og er under utvikling hele tiden, det er enda ikke definert standarder for alle de forretningsspråk vi bruker og derfor kan dette prosjektet være med på å påvirke hvordan disse standardene i fremtiden blir definert. Ved hjelp av en prototypmotor som dette kan det bli mye enklere for en bedrift å lage en presentasjon av en eller flere forretningsprosesser som de kan bruke ved salg av tjenester til sine kunder. Prototypen skal forenkle arbeidet i forbedringsfasen til en kundepresentasjon. Grunnen til at vi kun utvikler en prototyp er fordi det til tider er usikkert hva som lar seg gjøre og hva som ikke lar seg gjøre. Dette prosjektet er starten på noe som kan og bør utvikles videre for å bli utnyttet best mulig. Veldig mange bedrifter satser nå på teknologien rundt Web Services, det er derfor en fordel for oss å kunne en del om dette når vi skal

ut å søke jobb. Siden teknologien er plattformuavhengig vil alle i fremtiden kunne lage, bruke og publisere Web Services.

1.5 Målgruppen for prosjektrapporten

Målgruppene for rapporten er:

- Oppdragsgiver
- Personer som skal jobbe med fremtidige prosjekter om og med Web Services
- Sensor
- Andre som har interesser innen fagområdet

Det er i all hovedsak fagpersoner som i fremtiden vil kunne bruke og å ha nytte av denne rapporten. Noe av det vi skriver om er lett forståelig for alle mens en del vil være veldig spesifikt og vanskelig å forstå uten en del forkunnskaper. Vi har skrevet rapporten slik for at den skal være mulig å bruke for personer med kjennskap til fagområdet. Selv om IBM er vår oppdragsgiver, og deres verktøy er brukt i prosjekt arbeidet, vil rapporten forhåpentligvis ha nesten like stor nytte for personer som jobber med Web Services også utenfor IBM.

1.6 Vår bakgrunn og kompetanse

Prosjektgruppa består av Frode Mangseth(23 år) og Stein Tore Tøsse(22 år). Begge studerer dataingeniør på HiG. Vi har lært Java og databaser på skolen, noe vi har fått brukt mye i prosjektet. I tillegg har vi hatt bruk for systemutvikling. Vi valgte oppgaven fordi IBM er en spennende og utfordrende bedrift å jobbe for. Vi har fått muligheten til å se hvordan de jobber ute i bedriften. I løpet av arbeidet med oppgaven har vi lært svært mye om et felt som vi tidligere ikke har noen erfaring innen, samtidig som vi har brukt en del av det vi har lært her på skolen i løpet av de 2.5 første årene.

1.7 Arbeidsformer i gruppa

Fordi gruppen består av kun to personer har vi arbeidet mye hver for oss, det har stort sett ikke vært tid til at vi begge sitter med de samme oppgavene. Vi føler at dette ikke har hatt noen betydning for resultatene vi har oppnådd. Vi har begge hjulpet hverandre når vi har trengt dette. Under hele prosjektet har vi hatt et nært samarbeid med oppdragsgiver og har så ofte som det har vært mulig dratt for å jobbe på IBM Hamar. Der har vi fått mye faglig hjelp og tanker rundt det vi har utviklet. Fagområdet var ikke veldig kjent for vår veileder på HiG derfor har hans rolle vært som veileder i rapportskrivingen.

Veldig mye av tiden har blitt brukt til å lese og lære oss nye ting. I løpet av tiden vi har jobbet med prosjektet har vi lest veldig mange bøker og artikler som ligger på internett. For å øke kunnskapsnivået har vi også hatt for vane og diskutere de ting vi har lest for å se om begge har forstått tingene likt.

Prosjektet har vi delt opp i fire deler. Først har vi hatt en planleggingsfase, som startet allerede før jul. Så har det vært en fase med lesing og forståelse av teknologi. Denne fasen har pågått gjennom hele prosjektet. Under utviklingen av prototypen har vi fulgt IBMs Global Services Method mal som vist i avsnitt 8.1.2

Vi har avtalt statusmøte med veileder på HiG hver onsdag kl 0900. Disse timene har vært brukt når vi har trengt hjelp til rapportskriving, når vi har trengt hjelp på faglige områder har vi henvendt oss til vår oppdragsgiver på IBM. Under hele prosjektarbeidet har vi hatt kontakt med oppdragsgiver både over telefon, gjennom mail og ved hyppige besøk.

Under hele prosjektet har vi hatt faste arbeidstider, men når det har vært nødvendig med ekstra jobbing har ikke dette vært noe problem. Vi har kontor på høgskolens sivilingeniør lab og er veldig fornøyd med arbeidsforholdene der. Vi har underveis måttet finne ut og erfare hvordan vi oppnår ett best mulig resultat.

Gantt skjemaet vi lagde i planleggingsfasen kan sees i vedlegg B.

1.8 Øvrige Roller

Frode Haug er veileder og sensor. Han er først og fremst veileder når det gjelder å skrive rapporten best mulig. Men vi stiller ham faglige spørsmål hvis dette er nødvendig. Som prosjektveileder har vi hatt Jørn B. Nordlund ved IBM Hamar. I tillegg til at det er han som har gitt oppgaven blir han en veileder i å løse prosjektet best mulig faglig. I tillegg har vi fått hjelp av Gustav Bilben(også IBM Hamar), på en del tekniske spørsmål som har dukket opp underveis. Vi har brukt IBMs ressurspersoner aktivt fordi det ikke finnes mange personer på Høgskolen som har spesialisert seg innen Web Services. Jørn B. Nordlund har også tatt seg tid til å lese korrektur på rapporten og komme med innspill der han vil ha forandringer.

1.9 Organisering av selve rapporten

1.9.1 Beskrivelse

Rapporten er delt i 10 kapitler. Først kommer en innledning, med problemområde, avgrensning, formål, prosjektmål og organisering av rapporten. I kapittel 2 har vi skrevet en innføring i Web Services, her finnes både deler som er veldig enkle å forstå samtidig som mer detaljerte sider er beskrevet. Kapittel 3 er et eksempel på bruk av prototypen. I kapittel 4 har vi skrevet en fullstendig kravspesifikasjon. Kapittel 5 omhandler analysedelen av rapporten. I kapittel 6 forklares prototypens design. Det sjuende kapitlet inneholder implementeringen av prototypen. Kapittel 8 er en evaluering av kvalitetssikringen. I kapittel 9 diskuterer vi de resultater vi har kommet frem til. Mens i kapittel 10 har vi kommet frem til en konklusjon. Så kommer en liste over litteratur vi har benyttet i prosjektet og helt til slutt kommer alle vedlegg til rapporten.

1.9.2 Terminologi

Vi bruker Web-tjeneste og tjeneste i samme betydning som Web Services for å variere språket litt.

2 Innføring i Web Services

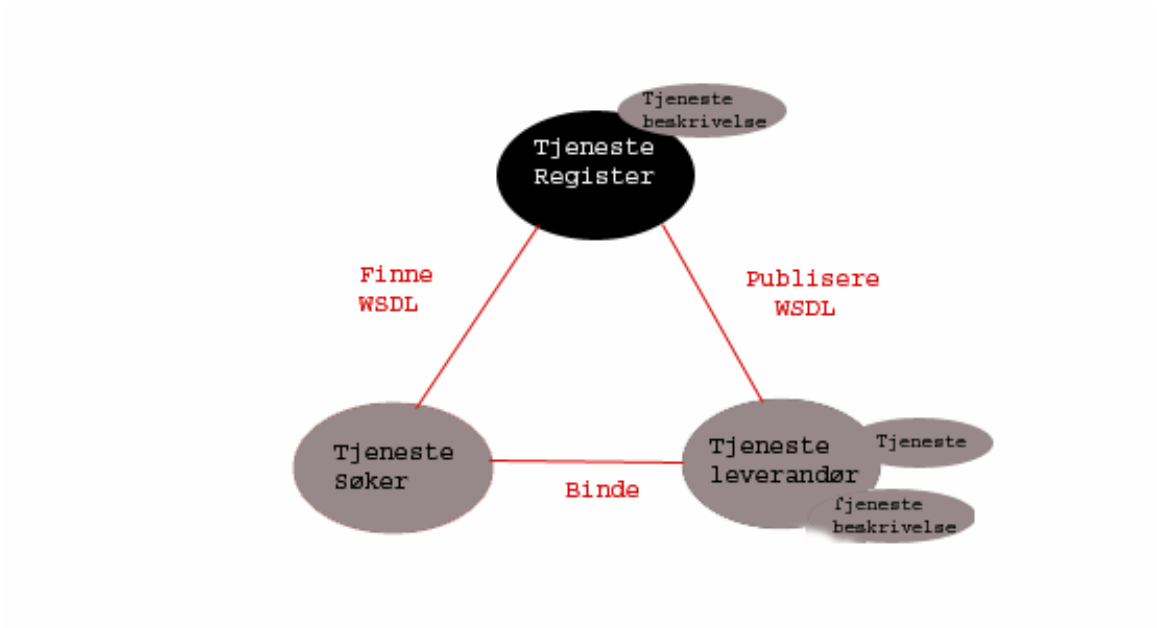
Oversetter vi Web Service[10] til norsk vil vi få noe vi kan kalle en Web-tjeneste, dette er egentlig en god beskrivelse av hva en Web Service egentlig er. En slik Web-tjeneste beskriver en gruppe operasjoner som er tilgjenglige over internett. Oppmelding til eksamen kan være en Web Service, hvor studenten taster inn sitt studentnummer og fagkode. Studenten kan så få en retur som forteller om når eksamen finner sted. Det som gjør dette til en Web Service er måten tjenesten beskrives på. Den beskrives ved bruk av de standarder som er satt for Web Services. Det finnes mange leverandører av slike Web-tjenester, men i denne rapporten forholder vi oss hovedsakelig til tjenestene som er bygd opp ved hjelp av IBMs standarder. På internett har det den siste tiden kommet mange Web-tjenester som brukes til testing[26]. Til hver Web-tjeneste finnes det en beskrivelse som lar brukeren vite alt som trengs for å påvirke tjenesten. Slike beskrivelser er basert på XML[22] og heter Web Services Description Language – WSDL[9]. Beskrivelsene inneholder all informasjon om tjenesten og dens inn og ut verdier. Denne beskrivelsen, som egentlig er et interface, skjuler altså all informasjonen om implementasjonen av selve tjenesten, slik at den kan bli brukt uavhengig av både hardware og software. Ett eksempel på en slik tjeneste kan være kredittsjekk av kunder for et selgerfirma. Tjenesten fungerer slik at leverandøren publiserer sin tjeneste til et register. Så må firmaet som trenger en tjeneste kontakte dette registeret. Leverandør og tjeneste søkeren blir så ”bundet” sammen, som vist i Figur 2.1. I dag finnes det mange gratis tjenester som for det meste er ment til testing, i fremtiden vil de aller fleste tjenester koste penger. Bruken av slike tjenester på nettet er voksende og det utvikles stadig nye programmer for å enklest mulig kunne lage Web Services. I dag er det for det meste snakk om tre teknologier når vi snakker om Web Services, disse er:

UDDI (Universal Description, Discovery, and Integration) – UDDI er et fellesskap mellom IBM, Microsoft og Ariba for å lage en internett standard for beskrivelse, registrering og oppdagelse av Web Services. Dette er et punkt vi ikke kommer mer inn på i denne oppgaven.

WSDL (Web Services Description Language) – WSDL-filer som inneholder kontakt informasjon, beskrivelse av Web Servicen, dens lokasjon og spesifikasjon for hvordan å bruke servicen.

SOAP (Simple Object Access Protocol)[16] – SOAP er en protokoll for utveksling av strukturert data i et distribuert miljø. Det er denne protokollen som brukes når en Web Service skal kjøres. SOAP er basert på RPC (remote procedure call), og hele poenget med SOAP er at når et program sender en melding med et SOAP metodekall eller en annen instruksjon over internett, så forventes mottagende applikasjon å sende et svar som tilfredsstiller forespørselen[15].

I en større sammenheng trengs ofte flere slike Web Services å fungere sammen, avhengig av hverandres resultater, derfor er det blitt introdusert enda en teknologi, Web Services Flow Language – WSFL. WSFL definerer en arbeidsflyt bestående av Web Services. Poenget med denne teknologien er å få flere Web-tjenester til å fungere sammen, en WSFL kan bestå av en eller flere andre WSFL'er og/eller en eller flere WSDL'er.



Figur 2.1 Web Service Arkitektur

- **Tjeneste leverandøren** (service provider) er eier av tjenesten sett fra et virksomhetsperspektiv. Sett fra et arkitekturperspektiv er dette plattformen som tilbyr tilgang til tjenesten.
- **Tjeneste søkeren** (service requester) er virksomheten som krever en viss funksjonalitet. Fra arkitekturperspektivet er dette applikasjonen/komponenten som leter etter og interagerer med en tjeneste.
- **Tjenesteregisteret** (service registry) er et søkbart register over tjenester hvor tjenestetilbydere kan publisere sine tjenester og tjenestebrukere kan finne tjenester og bindingsinformasjon for disse via tjenestebeskrivelser.

Publisere
Innebærer å reklamere for en tjeneste til et register. Tjeneste leverandøren kontakter registeret for å publisere en Web Service.
Finne
Denne operasjonen blir utført av tjeneste søkeren og tjeneste registeret sammen. Tjeneste søkeren beskriver den Web Servicen de leter etter, og registeret leverer resultatet som best passer til beskrivelsen. I fremtiden vil det også være mulig å finne WSFL-dokumenter som beskriver flere tjenester og deres samhørighet.

Binde
Operasjonen foregår mellom tjeneste søkeren og tjeneste leverandøren. De to partene kommer frem til en avtale slik at søkeren kan aksessere og bruke leverandørens Web Service.

2.1 Derfor er Web Services fremtiden

Ett av målene i utviklingen av en standard til Web Services er plattformuavhengighet. Dette vil si at brukeren ikke trenger å tenke på hvordan tjenesten er laget, brukeren kan bruke sin telefon med WAP og koble opp mot akkurat den samme tjenesten som ved bruk av en vanlig pc. Når en standard for å vise flyten mellom forretningsprosesser blir presentert vil dette være et stort skritt videre i Web Service teknologien, da får vi en standard som binder flere Web-tjenester sammen og kontrollerer flyten mellom dem. Brukeren trenger en applikasjon som tolker WSDL-dokumentet og påvirker Web-tjenesten på grunnlag av denne tolkningen. Web Services gjør forretningsprosessene veldig mye enklere for bedrifter, i stedet for å komme ut med helt nye applikasjoner og versjoner av software når små endringer er gjort kan bedriften vedlikeholde sin tjeneste og forandre kun på denne. Hvis en Web Service forandres, slik at den for eksempel får en ny adresse, trenger eieren av denne tjenesten kun å forandre adresse i WSDL-filen. Bedrifter kan skreddersy sine egne forretningsprosesser for å få et best mulig resultat, de kan selv bestemme hvilke Web-tjenester de vil bruke, i den rekkefølge de ønsker og basert på de regler bedrifter setter. I fremtiden vil Web-tjenestene koste penger, noen gjør også dette allerede. For i fremtiden å få utnyttet Web Services fullt ut trengs et verktøy som fra en modellert forretningsprosess generer WSFL etter de standarder som vil bli definert. Se også [8].

2.2 WSDL

Beskrivelsen av en Web Service må nødvendigvis gjøres av et standardisert og felles språk, til denne beskrivelsen brukes språket WSDL. Dette språket var designet med tanke på at leverandøren skal klare å beskrive tjenesten til verden, samtidig som verden skal skjønne hvordan tjenesten fungerer og hvordan de skal kunne bruke den.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PersonInfoRemoteInterface"
   targetNamespace="http://www.personinfo.com/definitions/PersonInfoRemoteInterface"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.personinfo.com/definitions/PersonInfoRemoteInterface"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsd1="http://www.personinfo.com/schemas/PersonInfoRemoteInterface">
   <message name="getPersInfoRequest">
      <part name="persnr" type="xsd:string"/>
```

```
  </message>
  <message name="getPersInfoResponse">
    <part name="result" type="xsd1:anyElement"/>
  </message>
  <portType name="PersonInfo">
    <operation name="getPersInfo" parameterOrder="persnr">
      <input message="tns:getPersInfoRequest" name="getPersInfoRequest"/>
      <output message="tns:getPersInfoResponse" name="getPersInfoResponse"/>
    </operation>
  </portType>
  <binding name="PersonInfoBinding" type="tns:PersonInfo">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getPersInfo">
      <soap:operation soapAction="" style="rpc"/>
      <input name="getPersInfoRequest">
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://tempuri.org/PersonInfo" use="encoded"/>
      </input>
      <output name="getPersInfoResponse">
        <soap:body
          encodingStyle="http://xml.apache.org/xml-soap/literalxml"
          namespace="http://tempuri.org/PersonInfo" use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="PersonInfoService">
    <port binding="tns:PersonInfoBinding" name="PersonInfoPort">
      <soap:address location="http://localhost:8080/WS/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```

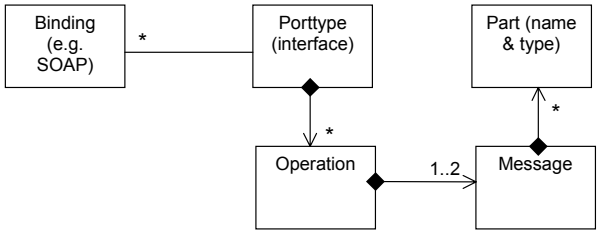
Kode 2.1

Kode 2.1 viser WSDL-filen til en Web Service vi har laget. For innføring i XML og namespaces se referanse[12] og [3]. Dette er en veldig enkelt beskrivelse av en tjeneste, men den viser de elementære og viktige delene av en WSDL. Først, rot noden i et WSDL-dokument heter alltid ”definitions”. Denne tjenesten inneholder kun en mulig metode som heter ”getPersInfo”. Hvilke metoder en WSDL inneholder kan du lese av elementene ”operation” under ”portType”, hvert slik ”operation” element er beskrivelsen av en metode. De forskjellige elementene som er viktige i dette dokumentet er:

- **Message**, denne delen inneholder her to elementer, de to viser hvilke parametere som skal bli sendt til tjenesten og hvilke som returnerer. Vi ser her at ”getPersInfoRequest” sender en parameter kalt ”persnr”, av typen string. Mens retur verdien står i ”getPersInfoResponse” hvor retur verdien av tjenesten er et anyElement, dvs. i form av XML.
- **portType**, et sett av operasjoner. Har en attributt som heter ”name” denne attributtens verdi er navnet på Web Servicen dette elementet tilhører.

- **Operation**, en beskrivelse av de metoder som er støttet av tjenesten. Vi ser her at input til metoden ”getPersInfo” heter ”getPersInfoRequest”. Og output heter ”getPersInfoResponse” som vi har vært inne på tidligere. Output og input viser til ”message” delen. Under ”operation” finner vi navnene på de ”message” elementene som definerer datatypene som skal sendes og mottas.
- **Service**, en beskrivelse av blant annet adressen til SOAP serveren som tjenesten skal bruke.

Vi har nå kun beskrevet de elementer i WSDL-dokumentet vi ser på som de viktigste, for mer informasjon og en fullstendig definisjon av WSDL se [4]. Figur 2.2 viser sammenhengen mellom hovedkonseptene i WSDL.



Figur 2.2

WSDL-dokumentet inneholder altså informasjon om hvor Web Servicen ligger, hva metodene som skal kalles heter og hvilke type returverdier som man får returnert fra tjenesten. WSDL er bygd opp av XML, uansett hvordan selve tjenesten fungerer, dette gjør Web Services til noe som kan brukes av alle.
Hver enkelt tjeneste har et unikt navn på den serveren den tilhører. Vi ser av WSDL-dokumentet at Web-tjenesten ”PersonInfo” har det unike navnet ”http://tempuri.org/PersonInfo”, kun en tjeneste på denne serveren kan ha dette navnet.

2.3 SOAP

SOAP definerer hvordan meldinger skal struktureres og hvordan data skal kodes på XML-format og består av tre deler: En konvolutt (envelope) som definerer et rammeverk for innholdet og prosesseringsregler for meldinger, kode-regler (encoding-rules) for å uttrykke instanser av datatyper og en konvensjon for å representere fjernprosedyrekall og svar. SOAP er en protokoll for å sende meldinger og utveksle informasjon i en Internett-omgivelse [2]. Den definerer et sett regler for strukturert meldingsutveksling som kan bli brukt til enkle enveis utveksling, men som viser sin styrke når den utfører RPC request-response dialoger. SOAP er ikke bundet til noen spesiell kommunikasjonsprotokoll. SOAP kan brukes i kombinasjon med en

rekke andre protokoller, selv om spesifikasjonen kun definerer bindinger til http [15]. Dette gjør SOAP til en viktig byggeblokk for utviklingen av distribuerte systemer.

2.4 WSFL

Web Services Flow Language er et XML-basert beskrivelse som beskriver flyten i en forretningsprosess bestående av en eller flere Web Services. WSFL[13] ble designet av IBM for å bli en del av Web Service teknologien på lik linje med SOAP, WSDL og UDDI. En slik løsning er enda ikke standardisert, og Microsoft har et språk, XLANG, som skal gjøre det samme som WSFL. Et WSFL-dokument kan, som nevnt tidligere, bestå av flere Web-tjenester eller flere WSFL-dokumenter, dette gjør at kompliserte og store forretningsprosesser kan beskrives i et dokument.

Innenfor Web Services er språk som XLANG (fra Microsoft) og Web Services Flow Language (WSFL – fra IBM) utviklet for å beskrive rekkefølgen av og regelverket for aktivitetene og dataflyten i en forretningsprosess. La oss se litt på WSFL:
WSFL beskriver to typer modeller:

- **Flow Model**: Her beskrives – koreograferes – en forretningsprosess som en sekvensiert sammensetning av Web Services. Innenfor Web Services-teknologien anvendes gjerne begrepet orkestrering istedenfor koreografi – de to termene har imidlertid i denne sammenheng nøyaktig samme betydning.
- **Global Model**: Dette er en slags samhandlingsmodell – den beskriver et sett interaksjoner eller lenker (*plugLinks)* – en lenke for hver aktivitet i flow-modellen som er implementert som en web service.

I en Flow Model vil hver web service være en activity – her kan vi bl.a. angi den operasjonen som webtjenesten utfører og den tjenestetilbyderen (serviceProvider) som implementerer tjenesten. Hver tjenestetilbyder er videre definert. Kontrollenker (controlLink) beskriver overganger fra en aktivitet til den neste, mens egne datalenker (dataLink) beskriver hvordan data flyter i modellen.

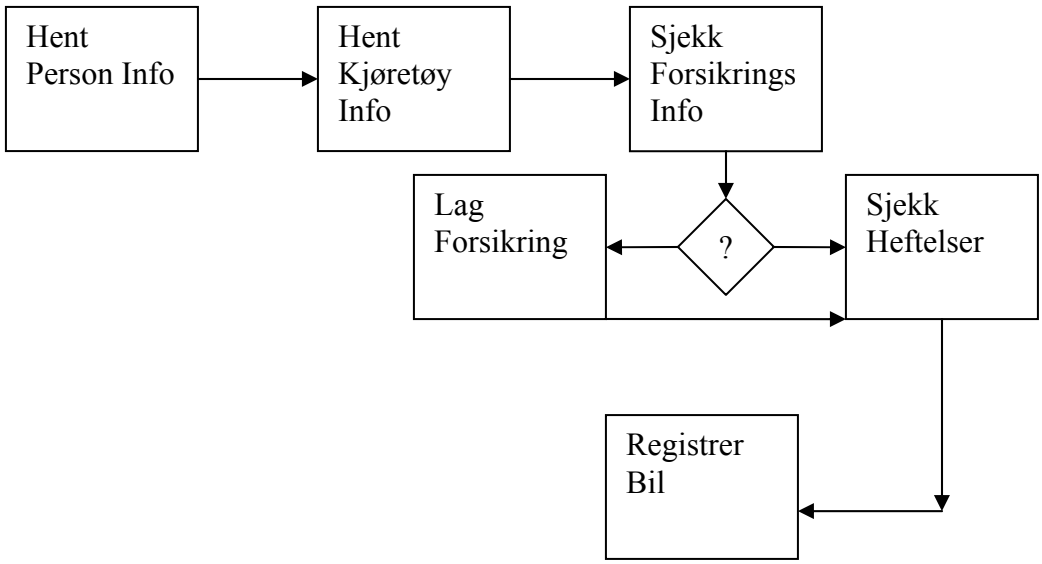
WSFLs Flow Model er primært en beskrivelse av en forretningsprosess - slik den ses internt i en organisasjon – der de enkelte operasjoner i prosessen realiseres som Web Services (som da kan være interaksjoner med samhandlingsparter) [15].

3 Eksempel på bruk av prototypen

Forretningsprosesser og Web Services er ikke alltid like lett å forstå. Det tar ofte tid før en skjønner hvordan dette fungerer. Vi vil derfor med et lite eksempel vise hvordan prototypen vil fungere i praksis. Eksempelet vi beskriver her har vi brukt til testing av prototypen.

3.1 Problemstilling

Et fiktivt firma som vi kaller ”Gamle Biler” har annonsert at de trenger et nytt system som behandler deres krav til omregistrering. ”Gamle Biler” har sammen med annonsen gitt ut hvilke krav de stiller til et slikt system. Kravene består av hva som skal sjekkes og hvilke funksjoner en slik registrering skal ha. Firmaet har planer om at systemet skal påbegynnes så fort så mulig, dette vil si at den bedriften som først kommer med et system ”Gamle Biler” har tro på vil få kontrakten med å utvikle systemet. Forretningsprosessen som dette systemet skal bruke er enkelt skissert i Figur 3.1.



Figur 3.1Forretningsprosessmodell

Firmaet vil ha et system der brukeren kun trenger å taste inn personnummeret til den nye eieren og bilens registreringsnummer. Alle data om personen og bilen blir hentet ut og brukt videre i prosessen. Når dette er gjort skal forsikringen sjekkes, hvis ingen forsikring finnes fra før av lages en ny forsikring, så sjekkes antall heftelser som finnes på bilen. Kunden får så se hvor mange heftelser som finnes på bilen, for så å bestemme seg for

om han vil fortsette eller ikke. Hvis kunden velger å fortsette registreres bilen på den nye eieren.

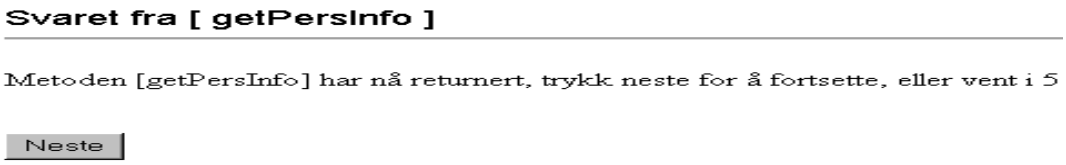
3.2 Løsning

Vårt firma har akkurat fått inn problembeskrivelsen fra ”Gamle Biler” og denne leses nøye, dette er en bra kontrakt og vi vil gjerne sette vår signatur på den. Det står klart i annonsen at de som har en løsning på problemet først får kontrakten, dette er en glimrende mulighet til å teste ut vår nye prototyp. En av våre selgere setter seg ned for å modellere forretningsprosessen i BPM Workbench.(Her kunne vi brukt mange andre modelleringsprogrammer, kravet er at de eksporterer FDL med syntaks lik den BPM Workbench gir.) Vårt firma har kontakt med et register av Web Services som bedrifter kan kjøpe. I dette registeret henter vi så ut Web Servicer som kan brukes i sammenheng med oppdraget. Her ville vi brukt tre Web Servicer som henter ut den informasjon vi trenger fra nasjonale register. Prototypen gjør det ikke lettere eller raskere å modellere en forretningsprosess, så det er denne delen som vil ta mest tid. Når modelleringen er ferdig og FDL har blitt generert kan vi kjøre FDL-filen gjennom prototypens parser. Deretter genereres det WSFL. Dokumentet som blir generert er ikke helt fullstendig i forhold til de krav vi setter til WSFL for at prototypen skal fungere. I WSFL-dokumentet må det manuelt legges inn informasjon slik som adresser til Web Service beskrivelsene (WSDL-filene). Når den informasjonen som trengs har blitt lagt inn skal prototypen testes. Det første skjermbildet som kommer opp under kjøring av prototypen er en liten velkomstmelding som forteller litt om tjenesten. For hvert skritt som skjer i forretningsprosessen må brukeren selv trykke en knapp, dette er for å vise alle de skritt som skjer i prototypen. Når et produkt er ute på markedet er et av målene å få dette til å gå automatisk. Nå skal prototypen vises til en kunde som skal forstå hva som skjer og hvordan det skjer. Figur 3.2 viser hvordan en metode i en Web Service blir representert i klienten. Navnet på metoden vises som overskrift mens alle parameterne som må sendes med blir listet opp nedover med navn og en tom tekstboks.

Figur 3.2

Før vi kan dra ut til kunden å vise frem hva vår prototyp har generert må vi selv teste om løsningen fungerer slik den skal. Vi tester en metode som tar

imot et personnummer og returnerer informasjon om personen som XML. Vi tester med personnummeret 1, som vi vet er en testperson i databasen.



Figur 3.3 Returbilde

Vi får opp bildet som vist i Figur 3.3. Hvis det skjer feil i uthenting av data kommer det opp beskjed om dette. Det ligger et skjult felt som heter ”data” som inneholder de data Web Servicen har returnert. Returen fra denne Web-tjenesten er som vist i Kode 3.1.

```
<personer>
        <person>
                <personnummer>1</personnummer>
                <etternavn>Andresen</etternavn>
                <fornavn>Anders</fornavn>
                <adresse>Akersgata 1</adresse>
                <postnr>1111</postnr>
                <poststed>Andeby</poststed>
        <person>
</personer>
```

Kode 3.1 Retur fra Web Servicen

Akkurat denne Web Servicen returnerer XML. Tjenesten vi bruker som sjekker om det finnes en forsikring på bilen returnerer et polisenummer. All data som blir returnert fra de forskjellige Web-tjenestene lagres og skrives ut når prosessen er ferdig. Når et fullverdig system blir laget vil disse dataene enkelt kunne bli behandlet. Når vi har sjekket at prototypen fungerer slik den skal, kan selgeren vår ta med seg sin bærbare pc med prototypen installert. Demonstrasjonen som her er blitt laget er lagret i dokumentet Prototyp.ppt på cd-rom.

4 Kravspesifikasjon

Vi har underveis i prosjektet utarbeidet en kravspesifikasjon for prototypen. Denne kravspesifikasjonen har fått små forandringer underveis når vi har sett at deler i prosjektet burde gjøres annerledes, men i hovedtrekk er den veldig lik førsteutkastet. Vi har gjennom hele prosjektet hatt møter om hvordan prototypen skal fungere. Kravspesifikasjonen er laget etter en mal som vi fikk i systemutviklingsfaget. Vi kom tilslutt frem til den endelige kravspesifikasjonen etter samarbeid med oppdragsgiver.

4.1 Brukerbeskrivelse

4.1.1 Omgivelser

System skal kjøres hos en oppdragsgiver som en demonstrasjon for en kunde, system trenger kun å være tilgjengelig fra en oppdragsgivers pc. Systemet skal være helt uavhengig av plattform og brukerne bruker sine nettlesere til å se på prototypen, for å bruke parseren trenger brukerne å ha Java installert. For at systemet skal fungere skikkelig kreves det at brukerne har tilknytning til Internett, systemet kan også kjøres lokalt på en pc.

4.1.2 Systemets Brukere

Systemet består kun av en gruppe brukere. Denne ene gruppen av brukere har all tilgang til systemet, prototypen er ikke et lukket system, og brukerne har tilgang til all kildekode de måtte ønske. Brukeren regnes som en kvalifisert person som kan gjøre endringer på systemet.

4.1.3 Funksjon

Systemet har tre moduler, den første modulen er **parseren**. Denne modulen brukes til å mappe fra ett språk til ett annet. Brukeren kan velge hvilken fil som skal mappes, mappingen skal baseres på ett sett med regler. Den andre modulen er **tilstandsmaskinen** som holder orden på hvor i prosessen en bruker befinner seg. Her velger brukeren en WSFL-fil som skal brukes i prosessen, denne tolkes og tilstandsmaskinen setter i gang en sesjon basert på de data som hentes fra WSFL-dokumentet. Den siste modulen er en **generisk Web Service klient** som enten kan fungere **alene**, eller som en del av hele systemet. Brukes modulen alene kan brukeren taste inn den WSDL som ønskes brukt, og hvilken metode i denne som skal brukes. Brukes

denne sammen med tilstandsmaskinen vil disse jobbe sammen og output til skjerm blir avhengig av input både fra brukeren og fra tilstandsmaskinen.

4.1.4 Operasjon

Systemet trenger kun å være tilgjengelig når brukeren skal vise en demonstrasjon for en kunde, men da bør systemet være stabilt og fungere bra. Ved feil i systemet bør brukeren være såpass kvalifisert at disse kan fikses. Det trengs ingen sikkerhetskopi av data som tastes inn i systemet.

4.1.5 Aspekter omkring livssyklus

Dataene i systemet skal vedlikeholdes og videreutvikles av brukeren, det er han som avgjør når systemet ikke skal brukes mer og når det trengs videreutvikling. Systemet skal kunne legges over på enhver pc som har installert en Apache Tomcat 4.0 server og skal være plattformuavhengig. Vi skal lage en prototyp som må videreutvikles for at den skal kunne brukes i en større sammenheng. Alt vi gjør skal dokumenteres for at fremtidig videreutvikling skal kunne gå lett.

4.1.6 Ytelse

Ytelsen til prototypen vil være begrenset av den pc som det kjøres på. Ytelsen vil også være avhengig av de Web Services som påvirkes over internett, her har da internettforbindelsen og pc-ens prosessor mye påvirkning.

4.1.7 Begrensninger

Prototypen skal ikke føre til noen nye investeringer, den skal utvikles ved bruk av språk og verktøy som IBM bruker. **Parseren** skal fungere ut fra de regler og krav som gruppa setter opp i løpet av prosjektet.

4.1.8 Antagelser

Vi antar at brukerne har en Apache Tomcat server installert, eller en liknende server som støtter JSP[18] og SOAP (slik som WebSphere Application Server). For at prototypen skal fungere best mulig antar vi at brukeren har en internett tilknytning.

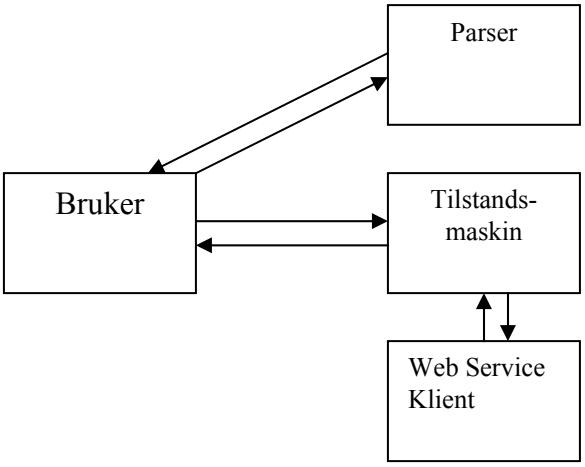
4.1.9 Hjelpefunksjon

Det finnes ingen hjelpefunksjon i denne prototypen, den hjelp som trengs kommer frem av denne rapporten. Det er foreløpig unødvendig med noen form for egen manual.

4.2 Funksjonell Kravspesifikasjon

4.2.1 Funksjonell Struktur og Tverrelasjoner

Figur 4.1 viser dataflyt mellom brukeren og systemet i de forskjellige moduler, dataflyten fra brukeren til systemet avhenger veldig av hvordan WSFL-dokumentet er oppgitt og hvordan tilstandsmaskinen skal ta imot og sende data mellom forskjellige Web Services.



Figur 4.1 Prinsipiell Arkitektur

Brukeren kan bestemme hvilke data som skal sendes med til den enkelte metode i en Web Service. Hvis det ikke er definert hva som skal sendes med og bearbeides i WSFL-dokumentet.

4.2.2 Data Spesifikasjon og Data Ordliste

4.2.2.1 Data Input

Brukeren taster inn adressen til FDL-dokumentet som skal parses. Tilstandsmaskinen tar imot en adresse til et WSFL-dokument som skal tolkes. Web Service klienten tar imot adressen til en WSDL-fil og hvilken metode i denne filen som skal brukes.

4.2.2.2 Data Output

Brukeren får adressen til et WSFL-dokument i retur fra parseren, i tillegg vises hele WSFL-dokumentet i en tekstboks. Tilstandsmaskinen skriver ut en velkomstmelding hvis denne ligger i WSFL-dokumentet. Helt til slutt

skriver den ut alle data som er hentet inn gjennom hele prosessen. Web Service klienten gir i output de argumenter som må fylles inn av brukeren i den valgte metoden. Når Web Servicen har returnert skrives det ut en melding om dette.

4.2.2.3 Tverrfunksjonelle datadefinisjoner

Modulene jobber mot forskjellige sett med data, en del av dataene i de forskjellige modulene blir sendt frem og tilbake.

4.2.3 Overordnede operasjonelle systemkrav

4.2.3.1 Normal Operasjon

4.2.3.1.1 Modus og kontroll

Systemet starter når brukeren velger å starte tilstandsmaskinen. Parseren starter og brukes kun når en bruker manuelt går inn og utfører en parsing.

4.2.3.1.2 Ytelse

Ytelsen til prototypen vil være begrenset av internettforbindelsen og brukerens pc. Det er ellers ingen krav til respons tider.

4.2.3.1.3 Sikkerhet

Det er ikke satt noen krav til sikkerhet i systemet.

4.2.3.1.4 Oppstart og nedtagning

Prototypen skal legges inn på en Apache Tomcat server. Det er ikke satt noen krav til å lage et installasjonsprogram, men en forklaring for hvordan å få systemet i gang skal beskrives i rapporten. Systemet er oppe når serveren er på, det er brukeren selv som tar systemet ned.

4.2.3.1.5 Tilgjengelighet

Denne avgjøres av brukeren, systemet har kun oppetid da brukeren trenger systemet.

4.2.3.1.6 Innebygde tester

Det er ikke lagt inn noen feil-testing, hvis en feil oppstår vil denne skrives ut til skjerm, og brukeren vil kunne rette på dette selv. Hvis en Web Service systemet kaller på ikke er oppe vil ikke systemet fungere. Ved en videreutvikling vil man da kunne finne tilsvarende Web-tjenester andre steder og bruke disse i stedet.

4.2.3.2 Operasjon i feilsituasjoner

4.2.3.2.1 Feilrapportering

Feil i systemet sendes til skjerm, slik at brukeren selv kan tolke og rette feilene. Brukeren får melding for eksempel dersom en fil ikke eksisterer.

4.2.3.2.2 Gjenervervelse etter feil

Brukeren kan prøve å rette en feil som eventuelt oppstår, så prøve å få systemet til å fungere på nytt. Etter en feil må brukeren begynne på nytt igjen, alle data vil bli borte hvis en feil oppstår midt i en sesjon.

4.2.3.2.3 Sikkerhet

Det er ikke satt noen krav til sikkerhet, derfor vil ikke feil ha noen form for innvirkning på sikkerheten.

4.2.3.2.4 Ytelse

Hvis det er problemer med å finne en fil, feil i internettforbindelsen vil gjøre at systemet går saktere.

4.2.4 Funksjonelle krav

4.2.4.1 Funksjonelle krav til Parsermodulen

4.2.4.1.1 Input

Her må brukeren gi et FDL-dokument som input til applikasjonen. Dette kan gjøres på to måter, enten ved å gi filnavn med kataloghenvisning som argument på kommandolinja eller ved å velge fil fra et fildialogvindu som dukker opp hvis det ikke gis med noe argument.

4.2.4.1.2 Prosessering

FDL-dokumentet blir lest inn og parset sekvensielt etter en de regler vi bestemmer og bygger samtidig opp et WSFL-dokument. Altså søkes dokumentet etter kjente elementer.

4.2.4.1.3 Output

Det resulterende WSLF-dokumentet dukker opp i et tekstredigeringsvindu. Her kan brukerne tilføye eller gjøre andre endringer direkte, deretter kan det ferdige dokumentet enten lagres til en fil eller det kan markeres og kopieres.

4.2.4.1.4 Feilrapportering

Det skal ikke legges så mye vekt på robusthet og stabilitet, men det bør gis melding hvis filen ikke finnes og lignende. Oppstår det en kritisk feil rapporteres dette ved hjelp av feilmeldinger fra Java. Det forusettes at brukeren er kyndig og ikke gir totalt feilaktig input.

4.2.4.1.5 Gjenervervelse etter feil

Hvis det oppstår en feil vil programmet bare avbryte og det må startes på nytt.

4.2.4.1.6 Sikkerhet

Sikkerhet er ikke vektlagt i det hele tatt. Det er lite eller ingenting i denne applikasjonen som kan være noe sikkerhetsrisiko.

4.2.4.2 Funksjonelle krav til Tilstandsmaskinmodulen

4.2.4.2.1 Input

Denne tilstandsmaskinen skal ta et WSFL-dokument som input. Måten dette gjøres på er at det sendes med en URL som enten kan henvise til en fil lokalt på maskinen som tilstandsmaskinen kjøres på eller en fil som ligger på en WWW-tjener på Internet. ette kan enten være generert fra parsemodulen (som generer WSFL fra FDL), satt opp manuelt eller et annet dokument som tilfredsstiller de reglene vi har satt opp for et ”gyldig” WSFL-dokument.

4.2.4.2.2 Prosessering

Først parses dokumentet som et XML-dokument, og bygger opp de ulike tilstandene med nødvendig informasjon. De ulike tilstandene vil her være de aktivitetene som beskrives i WSFL-dokumentet og hver aktivitet beskriver en Web Service. Deretter bygges en transisjonstabell opp. Denne tabellen beskriver hvilken tilstand man skal gå til nesten gang gitt nåværende tilstand og en eventuell betingelse. Med andre ord så er tabellen koreografien mellom de ulike Web Servicene. Det bør også være muligheter for å ta vare på dataene hele veien slik at disse kan vises i en oppsummering til slutt.

4.2.4.2.3 Output

Brukeren vil følge de ulike tilstandene via et HTML grensesnitt og blir bedt om å fylle inn nødvendig informasjon og brukeren trykker en knapp for å gå til nesten tilstand. Brukeren vil ikke ha noe mulighet til å påvirke koreografien eller dataflyten mellom de ulike aktivitetene, men kun bestemme hastigheten det går i. Og til slutt vil alle data listes opp på en siste side, dette er strengt tatt ikke nødvendig, men kan være greit for å oppsummere og sjekke at alt gikk greit.

4.2.4.2.4 Gjenervervelse etter feil

Hvis det oppstår en feil vil tilstandsmaskinen avbryte og den må startes på nytt. Grunnen til dette er at dette kun skal være en prototyp og skal kun behandles av kyndige mennesker og i all hovedsak oppdragsgiver.

4.2.4.2.5 Sikkerhet

Sikkerheten er lite vektlagt også, da dette kun er en enkel prototyp for å undersøke om det er mulig å gjennomføre de mål vi satte oss.

4.2.4.3 Funksjonelle krav til Generisk Web Service Klient modulen

4.2.4.3.1 Input

Brukeren må taste inn de data som en metode i den valgte Web Servicen trenger for å kunne returnere korrekte verdier. Brukeren skal kunne velge hvilken metode i en Web Service som skal brukes. Dersom denne delen brukes alene taster brukeren inn hvilken WSDL-fil som skal brukes. Her tastes filnavnet inn som en adresse, de fleste slike filer som skal brukes ligger på internett og derfor blir http protokollen brukt her.
Input til hver enkelt Web Service er beskrevet i den tilhørende WSDL-filen. Web-tjenestene kan ta imot de fleste typer input. Det vanligste er at en Web Service tar imot strenger. Hvor mange verdier som blir sendt med når Web Servicen skal brukes er totalt avhengig av Web-tjenestens tilhørende WSDL-dokument.

4.2.4.3.2 Prosessering

Filen som ligger på adressen som blir tastet inn eller sendt med, skal tolkes og de data brukeren trenger skal skrives til skjerm. Hvis adressen er ugyldig eller filen ikke er spesifisert korrekt vil ingen data skrives til skjerm.

4.2.4.3.3 Output

Brukeren skal få en oversikt over de metoder som kan brukes og hvilke parametere som brukeren må fylle inn. Brukeren vil til slutt få en melding om at Web-tjenesten er fullført og at en verdi er returnert. Denne verdien vil så bli sendt videre til tilstandsmaskinen.

4.2.4.3.4 Feilrapportering

Det skrives feilmelding til skjerm dersom noe har gått galt. Går noe galt må prosessen startes på nytt neste gang man skal prøve.

4.2.4.3.5 Gjenervervelse etter feil

Dersom systemet går ned må brukeren selv ordne dette og starte systemet på nytt. Dataene som brukeren arbeidet med vil ikke bli lagret.

4.2.4.3.6 Sikkerhet

Det er ikke satt noen krav til sikkerhet i systemet.

4.3 Begrensninger

4.3.1 Software design begrensninger

4.3.1.1 Software standarder og språk

System skal utvikles i Java, JSP, XSL[20], XSLT[23] og HTML.

4.3.1.2 Software grensesnitt

Nettleser, HTML. Systemet kommuniserer internt ved bruk av internettstandard og til Web Servicene ved bruk av SOAP.

4.3.1.3 Software pakker/verktøy

Serveren må ha et verktøy som støtter JSP og SOAP, slik som Apache Tomcat 4.0. Det er ønskelig at systemet i det minste skal fungere på Internet Explorer.

4.3.1.4 Software kommunikasjonsstandarder og grensesnitt

Kommunikasjon mot brukeren foregår enten lokalt på en pc, eller over internett. Brukeren skal kun trenge en vanlig internettforbindelse og en nettleser som støtter Java og XML.

4.3.1.5 Operativsystem

Systemet kan ligge på en hvilken som helst plattform, det skal være plattformuavhengig. Brukeren vil med fordel bruke Internet Explorer 6.0 eller nyere som støtter XML.

4.3.2 Hardware design og begrensninger

4.3.2.1 Hardware krav og omgivelser

Det er ikke satt noen krav til hardware sett bort fra at serveren på fungere på den brukte hardware.

4.3.2.2 Hardware grensesnitt

Dette begrenses av brukerens internettforbindelse og operativsystem.

4.3.3 Aspekter omkring livssyklus

4.3.3.1 Dokumentasjon

Koden må kommenteres. Vi må også lage en oversikt over alle filer og kataloger som brukes.

4.3.3.2 Modul og integrasjonstesting

Modulene testes etter at de er ferdig utviklet hver for seg, men noe må testes sammen. Vi vil også teste modulene underveis, og jobbe med disse til vi er fornøyde og de fungerer bra. Vi kommer til å gi ut en prototyp, denne bør videreutvikles. Det gis ikke ut noen endelig versjon av systemet enda.

4.3.3.3 Konfigurasjon og Versjonsstyring

Vi lager kun en prototyp versjon, som kun vil lanseres for vår oppdragsgiver. Det er ingen versjonsnummer på dette systemet, dersom det skulle hatt ett nummer vill det vært 0.5.

4.3.3.4 Krav til support, service og vedlikehold

Brukeren selv har ansvar for support. Brukeren vil være kvalifisert til å modifisere og vedlikeholde systemet. Systemet skal ikke behøve noen kontinuerlig vedlikehold, det trenger heller en kontinuerlig videreutvikling.

4.3.3.5 Krav til utvidelser

Systemet skal kunne utvides og viderearbeides, eller være grunnlaget for en ny prototyp eller ett nytt fullverdig system.

4.3.4 Aspekter omkring Installasjon

4.3.4.1 Overgang og omlegging

Prototypen er helt ny, det finne ikke lignende systemer fra før. Brukeren følger de installasjonsrutiner som er satt opp og installerer på lokal pc.

4.3.4.2 Opplæring

Brukeren som her er vår oppdragsgiver på IBM Hamar har fulgt oss under hele prosessen og vil kun trenge en liten innføring i hvordan bruke systemet. Brukeren skal kunne forklare og lære bort det som skjer.

4.3.4.3 Utgivelser underveis

Vi skal i løpet av prosjektet levere tre statusrapporter. De tre status rapportene skal leveres når vi har nådd forskjellige milepæler i prosjektet. Disse statusrapportene skal inneholde hvordan vi ligger i forhold til tidsskjemaet, hva vi har gjort, litt om samarbeidet, motivasjon og hvilke problemer / muligheter vi ser.

4.3.4.4 Akseptansekrav

Prototypen skal tilfredsstille de krav som er stilt i denne kravspesifikasjonen. Oppdragsgiver kan komme med forandringer og tillegg underveis. Vi retter feil underveis, så lenge vi har tid til dette.

5 Analyse

5.1 Hva systemet skal løse

Gjennom arbeid med kravspesifikasjon og møter med oppdragsgiver fikk vi frem hva vi skulle lage. Da vi startet var det ikke sikkert at alt det vi skulle prøve å lage faktisk var mulig å lage, og en stor del av oppgaven var å undersøke om det var praktisk mulig. Mye av det vi har lest om og de løsninger vi har sett på er enda ikke standardisert og spesifisert. Det første vi gjorde var å sette oss inn i Web Services og lære hvordan dette fungerte. Så begynte vi med parseren og med Web Service klienten, helt til slutt lagde vi tilstandsmaskinen. Det hele skal resultere i en prototyp som kan brukes som et demonstrasjonsverktøy. Prototypen er starten på en løsning av problemområdet vårt, det blir ingen fullverdig løsning, men starten på noe som må videreutvikles. Vi har hele veien vurdert fordeler og ulemper med det vi har gjort. Flere ganger har vi måttet gjøre ting på andre måter eller tenke annerledes enn det vi trodde i utgangspunktet. Parseren skal ta for seg en et FDL-dokument, parse dette og plukke ut de nødvendige elementene. Disse elementene skal så mappes til WSFL-elementer. Til slutt settes elementene sammen til et WSFL-dokument. Det stilles en del krav til FDL-dokumentet, derfor må også forretningsmodellene designes etter regler vi setter opp. Dette betyr at parseren ikke godtar alle FDLer, dessuten er det mulig at de WSFLene som genereres ikke er ”gyldige” i fremtiden da WSFL ikke er standardisert enda. Så vi har hatt som mål at WSFL-dokumentet skal kunne parses av tilstandsmaskinen. Tilstandsmaskinen skal som sagt parse et WSFL-dokument og gjenskape forretningsmodellen som beskrevet i FDLen i form av tilstander og transisjoner. Målet med tilstandsmaskinen er at brukerne skal kunne kjøre igjennom forretningsmodellen i praksis. Dette er en enkel prototyp, derfor skal ikke stabilitet og generiskhet prioriteres.

5.2 Eksisterende Løsninger

Når vi startet prosjektet fantes det ingen eksisterende løsninger på vårt problem. Disse er under utvikling, og vi har hørt at det har kommet ut en liknende løsning fra SilverStream[24]. Vi har ikke hatt noen mulighet til verken å se på eller teste denne løsningen, så vi har ingen kommentarer til den.

Når det gjelder den Generiske Web Service Klienten finnes det noen løsninger ute på internett, vi har kikket litt på en av dem.

- SoapClient, dette er en veldig bra generisk WSDL-tolker, som fungerer på de fleste WSDL-dokumenter vi har testet. Men denne er

bygd ved hjelp av et programmerings bibliotek som koster penger. Denne klienten har ingen mulighet for å samarbeide med de resterende delene av prosjektet vårt, men vi har brukt tid på å studere hvordan denne fungerer for å få til et best mulig resultat.
Hjemmeside: http://www.soapclient.com/soaptest.html

5.3 Totalvurdering

En av grunnene til at prosjektet skulle gjennomføres var at det ikke fantes noen løsninger på problemet. I løpet av det siste halvåret har det kommet en løsning som visstnok skal fungere. Vi har ikke hatt noen mulighet til å teste dette. Prototypen skal lages for å teste ut ny teknologi.

6 Design

Prototypens hoveddeler vises i Figur 4.1. De tre modulene prototypen består av har først blitt laget hver for seg, for så å ha blitt satt sammen. Hver enkelt av modulene beskrives videre i dette kapittelet.

6.1 Hvordan vi laget det

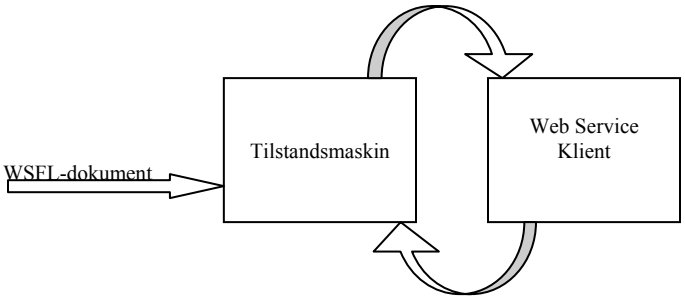
6.1.1 Parseren

Vi trodde først at denne skulle være enkelt å lage, da mappingen fra FDL til WSFL virket ganske triviell. Men etter å ha begynt med utviklingen støtte vi på en rekke problemer. Vi hadde ingen forutsetninger for å bruke programmet Holosofx BPM Workbench da ingen av oss har noen erfaring med forretningsprosesser og alle de ord og uttrykk som hører til. Så det vi prøvde på først var å bruke en del av de eksemplene som følger med Holosofx og bare eksporterte de rett til FDL. Vi satte oss deretter og sammenlignet FDL med nåværende forslaget til WSFL for å finne noen regler med tanke på mappingen. Og det virket greit i starten, men etter hvert viste det seg FDL ikke alltid var så enkelt å forstå som vi trodde. Men vi har allikevel lagd en prototyp på en FDL til WSFL mapper. Den er ikke veldig generisk, men fungerer i de tilfeller hvor forretningsmodellen settes opp etter de regler vi har kommet fram til.

6.1.2 Tilstandsmaskinen

Tilstandsmaskinen[5] eller WSFL-motoren skal ”lede” brukeren igjen om ett sett med ulike Web Services, denne koreografien er beskrevet i et WSFL-dokument. Figur 6.1 viser en overordnet arkitektur for tilstandsmaskinen. Denne valgte vi å implementere som en Java Servlet og bruker sesjoner for å ta vare på informasjon og data. Vi ble enig med oppdragsgiver at denne motoren ikke skal være veldig generisk, men være en pekepinn på om dette faktisk lar seg gjøre. Tilstandsmaskinen får WSFL-dokumentet som input. Dette parses som et XML-dokument. Først bygges en oversikt over alle tjenestetilbydere med nødvendig informasjon, deretter settes alle de ulike aktivitetene eller tilstandene opp. Hver av disse er en Web Service. Så settes koreografien opp, altså flyten og eventuelle betingelser mellom de ulike tilstandene. Til slutt settes eventuell dataflyt mellom de ulike aktivitetene opp. Når dokumentet er lest ferdig vises det første skjermbildet. Hva som blir første bildet kommer an på om det er lagt inn ett dokumentasjons element i WSFL-dokumentet eller ikke. Er dette lagt inn vises denne teksten

som en velkomst tekst hvor brukeren må trykke på en knapp for å starte tjenesten. Om dette elementet ikke er lagt inn blir brukeren ledet rett til den første aktiviteten. Som nevnt før så er det ikke tilstandsmaskinen som binder seg til de forskjellige Web Servicene, dette er det Modul 3 som gjør. Motoren kaller bare opp de ulike tjenestene og tar imot eventuelle returdata og om nødvendig sender dette videre til neste tjeneste som skal utføres. Når hele forretningsprosessen er gjennomført listes alle data som er returnert fra de ulike aktivitetene på en siste side. Etter dette er sesjonen avsluttet.



Figur 6.1 Tilstandsmaskinens Overordnede Arkitektur

6.1.3 Generisk Web Service Klient

Når vi startet med designet av Web Service klienten, fant vi fram noen WSDL-dokumenter som vi i første omgang skulle få klienten til å fungere sammen med. Det viktige var å få klienten til å fungere sammen med de WSDL-filer og Web Services som WSAD [10][6] genererer og støtter. Underveis visste vi at vi kunne forandre og forbedre løsningen hvis dette trengtes. Denne modulen har blitt forandret flere ganger underveis og vi kunne godt ha skrevet om mange versjoner av modulen, men vi har kommet frem til at vi forteller om de to produksjonene som fungerer best. Først laget vi et XSL-dokument som tolket WSDL-dokumentet og skrev dens tilhørende metoder og parametere til skjerm. Alt fungerte bra og det så ut som om dette var den endelige løsningen, den kommuniserte først med en JSP-fil som kommuniserte videre med en Java-klasse. For å optimalisere dette ble funksjonaliteten fra Java-klassen lagt over i JSP-filen. Vi merket med en gang at dette økte ytelsen til systemet og det var enklere å søke etter feil. Når denne modulen var slik vi tenkte den skulle være og vi skulle koble den sammen med tilstandsmaskinen oppdaget vi at det var en mangel. Tilstandsmaskinen skal nemlig kunne bestemme hvilken metode fra WSDL-dokumentet som skal skrives til skjerm, og eventuelt sende med data til noen av parameterne. XSL[7] er ikke et veldig komplisert programmeringsspråk. Det finnes mange operasjoner som ikke er mulig å gjennomføre. En av de tingene vi ikke fikk gjort var nemlig å sende med data til XSL-dokumentet, for så å hente disse dataene ut og jobbe videre på bakgrunn av disse data. Derfor begynte vi å utvikle en versjon til av Web Service klienten. Den nyeste klienten jobber mot den samme JSP-filen som den første, men her

kan tilstandsmaskinen bestemme hvilken metode som skal vises på skjerm. Denne klienten er laget i JSP. Begge løsningene plukker ut de samme data fra WSDL-filen, slik at den videre prosesseringen blir lik. I XSL-løsningen vil et inputfelt skrives ut til skjerm dersom ingen adresse til et WSDL-dokument er gitt. Her kan brukeren manuelt taste inn adressen til den WSDL han ønsker å benytte seg av.

6.2 Hvorfor laget vi det slik?

Løsningen er en prototyp og alle de skjermbilder som blir generert er der for at en eventuell kunde skal se hvordan det hele fungerer. Ideen bak dette er at det skal fungere automatisk, og at kun sluttevalueringen skal skrives til skjerm. En slik løsning vil nødvendigvis være en del av en større applikasjon og samhandle med denne. Noe av det vi har laget ser veldig enkelt ut for de som kun ser på demonstrasjonen og ikke på det som ligger bak. Vårt mål er ikke å lage noe som ser best mulig ut. Vi har konsentrert oss om det som ligger bak og de skjermbilder som kommer opp skal være enkle, kortfattede og informative slik at en bruker skal forstå og enkelt kunne forklare hva som skjer. Parseren er laget med et skjermbilde der brukeren kan se på den WSFL-filen som er generert, her bør og må brukeren gå inn for å se om alle de elementer som han vil ha med i sin WSFL er slik de skal være. Noen av elementene, som det ikke finnes noen måte å representere i FDL, vil brukeren nødvendigvis gå inn å legge til i WSFL-dokumentet.

7 Implementering

7.1 Valg av programmeringsspråk

Prosjektet vi skulle løse for IBM måtte løses på IBMs egne premisser og vi valgte å bruke de språk og verktøy som IBM bruker. Parseren og tilstandsmaskinen er derfor begge løst ved å bruke Java, et språk som IBM har veldig stor tro på og som de vil satse videre på i fremtiden. JSP valgte vi å bruke fordi det er enkelt å lage dynamiske Web-sider og fordi servere som støtter SOAP også støtter JSP. Dette gjør at brukeren slipper å installere mange programmer på sin pc for å få prototypen til å fungere. Alle moduler i prototypen skal programmeres i samme språk for at utvidelser og integrasjoner skal skje enkelt. Vi valgte først å bruke XSL og XSLT til å tolke og hente ut de elementer i WSDL-dokumentene som vi trengte. Dette viste seg å være en grei løsning, men en del av den funksjonaliteten vi ønsket var ikke mulig å få laget ved bruk av XSL. Derfor laget vi også en løsning i JSP. Vi har valgt å forklare hvordan begge de to klientløsningene er laget, for så å veie de opp mot hverandre. XSL var og burde være det naturligste valget siden WSDL-dokumentene bygger på XML og XSL er StyleSheets for XML slik CSS er StyleSheets for HTML. JSP er en veldig fleksibel løsning.

7.1.1 Parseren

Denne applikasjonen skal som tidligere nevnt mappe fra et flytspråk til et annet. I dette tilfellet fra FDL til WSFL[13]. Det ble bestemt at robusthet og fleksibilitet ikke skulle være noe primærmål i utviklingen av prototypen. Dette har selvfølgelig påvirket utviklingen en god del. Først satte vi opp regler for mappingen. Det første som skjer i Java-klassen er at den forsøker å åpne et FDL-dokument, som brukeren enten angir filbanen til som argument eller plukker fra et filvelgervindu. Deretter leses linje for linje nedover i FDL-dokumentet og sjekkes etter kjente elementer som vi har laget regler for. Hver gang et ”kjent” FDL-element gjenkjennes skrives det tilhørende WSFL-elementet til et buffer. Alle elementer som ikke er relevante for et WSFL-dokument hoppes bare over og tolkes ikke på noen måte av parseren. Når vi når slutten av FDL-dokumentet vises den oppbygde WSFL’en i et tekstediteringsvindu. Her kan brukeren editere, tilføye eller fjerne deler av WSFL-dokumentet og kan deretter velge å lagre dokumentet til en fil eller markere det og kopiere det til utklippstavlen.

7.1.1.1 Regler for mapping fra FDL til WSFL

Her presenter vi de reglene som brukes når et FDL-dokument skal mappes til et WSFL-dokument. Disse reglene er ikke helt generelle da ulike produsenter av forretningsmodelleringsverktøy ofte har sine egne tillegg og variasjoner til FDL, men de er gjeldene i mange tilfeller. Dessuten er det muligheter for at det vil bli små endringer i WSFL da dette språket ikke er standardisert enda. Målet med denne mappingen er å kunne modellere en forretningsprosess i et verktøy som kan generere FDL, for deretter å kunne gi en gjennomkjøring i praksis.

7.1.1.1.1 Aktivitetene

Dette er de ulike aktivitetene i forretningsprosessen, disse blir representert som tilstander i tilstandsmaskinen. I FDL leter vi først etter ”PROGRAM_ACTIVITY”-element, som vist i Kode 7.1. Dette elementet mappes til et <activity>-element i WSFL, som vist i Kode 7.2. Attributter som navn, tjenestetilbyder og metodenavn plukkes også ut.

```
PROGRAM_ACTIVITY  Approvet'('SWOrder','SWOrder')
        LAYOUT   XPOS -8935   YPOS 9224
        START AUTOMATIC WHEN ALL CONNECTORS TRUE
        EXIT MANUAL
        PRIORITY 0
        DONE_BY PROCESS_STARTER
        PROGRAM 'ApproveOrder'
        SYNCHRONIZATION NESTED
        END 'What'
```

Kode 7.1 Aktiviteter i FDL

```
<activity name="Approve">
 <performedBy serviceProvider="PROCESS_STARTER"/>
 <implement>
  <export>
   <target portType="totalSupplyPT" operation=" ApproveOrder "/>
  </export>
 </implement>
</activity>
```

Kode 7.2Aktiviteter i WSFL

7.1.1.1.2 Koreografi/Transisjoner

Her skal samhandlingen mellom de ulike aktivitetene beskrives. Dette betegnes som koreografien mellom de ulike aktivitetene, som beskriver regler for hvilken aktivitet man går til neste gang. I tilstandsmaskinen vil dette tilsvare de ulike transisjonene mellom tilstandene. FDL definerer dette som et ”CONTROL”-element, med attributtene ”FROM” og ”TO” som sier hvilken aktivitet man skal gå til når man er ferdig med en annen. Vi mapper

dette til et <controlLink>-element i WSFL, men henholdsvis ”source” og ”target” som attributter. I tillegg kan det også være betingelser, ”CONTROL”-elementet har da et ”WHEN”-attributt som inneholder et boolsk uttrykk, vist i Kode 7.3. Vi mapper bare dette til et ”transitionCondition”-attributt til <controlLink>-elementet, som vist i Kode 7.4.

```
CONTROL FROM 'Approve' TO 'Notify'
        WHEN 'Accept= "No" '
```

Kode 7.3Control element i FDL

```
<controlLink source=”Approve” target=”Notify” conditionTransition=”Accept='No'”/>
```

Kode 7.4 Control element i WSFL

7.1.1.1.3 Dataflyt

Det er også muligheter for å sende data fra en aktivitet videre til en annen, men det er ikke påkrevd å ha dataflyt i forbindelse med hver transisjon. Dataflyt beskrives i FDL med et ”DATA”-element med ”FROM”-og ”TO”-attributt, vist i Kode 7.5. Dette mappes til et <dataLink>-element i WSFL, med ”source” og ”target”-elementer, som vist i Kode 7.6. I tillegg er det muligheter for å mappe fra en datatype til en annen, men dette har ikke vi benyttet oss av.

```
DATA FROM 'What' TO 'Approve'
        MAP '_STRUCT' TO '_STRUCT'
```

Kode 7.5 Data Element i FDL

```
<dataLink source=”What” target=”Approve”>
  <map sourceMessage="_STRUCT " targetMessage="_STRUCT "/>
</dataLink>
```

Kode 7.6 DataLink i WSFL

7.1.2 Tilstandsmaskinen

Vi ble enige om at denne maskinen skulle være tilgjengelig gjennom en nettleser da dette stiller lite krav til programvare hos brukeren. Og siden det andre vi har laget er utviklet i Java valgte vi å bruke Java Servlets til denne tilstandsmaskinen. For å holde orden på de dataene som for eksempel nåværende tilstand, returdata og lignende valgte vi å bruke sesjoner. Det stiller kun krav til at nettleseren støtter standard HTML og cookies, ikke noe annet. Siden tilstandsmaskinen trenger ett WSFL-dokument som input, må

den ha en URL som argument. Dette kan for eksempel gjøres i URL'en til tilstandsmaskinen, altså på som vist i Kode 7.7

```
http://katalog/FSM.java?wsfl=<URL til WSFL>
```

Kode 7.7

WSFL-dokumentet må deretter parses for å bygge opp tilstander og transisjoner. Vi tenkte først å parse WSFL-dokumentet omtrent på samme måte som vi gjorde med FDL-dokumentet i Modul 1. Men vi mente at det burde finnes enklere og bedre måter å lese ut informasjonen da et WSFL-dokument er et gyldig XML-dokument. Hos Apache Group fant vi en pakke som heter Xerces, denne inneholder forskjellige verktøy for å lese et XML-dokument. Dette gjorde ting mye lettere da vi kunne søke etter de elementene vi trengte og plukke ut de nødvendige attributtene. For å representere de ulike tjenestetilbyderne, aktivitetene og transisjonene lagde vi en klasse for hver av de. Hvor for eksempel en instans av tjenestetilbyder-klassen inneholder alle de ulike tjenesttilbyderene som en forretningsprosess innehold, det samme gjelder aktiviteter og transisjoner. Når alt dette er satt opp er det klart for å starte forretningsprosessen. Hva som skjer først avhenger av om det finnes en <documentation>-tag i WSFL-dokumentet. Hvis det gjør dette vises det en tekst, brukeren klikker "start" og kommer til første oppgave i forretningsprosessen. Hvis <document.>-tagen ikke er tatt med sendes brukeren rett til første oppgave. Det som skjer når bruker er i en aktivitet eller tilstand er at Modul 3 kalles opp for å utføre oppgaven. Så i de tilfellene brukeren skal taste inn noe så snakker han ikke direkte med tilstandsmaskinen, men med den Generiske Web Service Klienten. Denne klienten tar imot eventuelle parametere, utfører tjenesten og sender eventuelle returdata tilbake til tilstandsmaskinen. Da kan tilstandsmaskinen utifra nåværende tilstand og returdata hoppe til neste tilstand i forretningsprosessen. Dette skjer helt til forretningsprosessen er ferdig. Helt til slutt listes alle data som tilstandsmaskinen har mottatt på en avslutningsside. Disse dataene ligger lagret i en Vector-klasse. Helt til slutt avsluttes sesjonen slik at brukeren kan starte på nytt igjen.

7.1.3 Generisk Web Service Klient

7.1.3.1 Tolkning av WSDL

7.1.3.1.1 Hva som måtte gjøres

Kommunikasjonen med Web Service delen trenger en del bestemte elementer og parametere fra WSDL-dokumentet som vi må hente ut, i

tillegg trenger den de parametere og de verdier som skal bli sendt med til Web Servicen. Uthenting av informasjon fra filen er gjort på følgende måter

- Først finner vi det elementet i WSDL-filen som heter <portType>. Under dette elementet ligger andre noder som inneholder navn på alle de metoder som finnes i den Web Service WSDL-dokumentet beskriver.
- Så lagres informasjon i skjulte inputfelter slik at de enkelt kan hentes ut igjen senere.
- For hvert argument som skal sendes med til metoden lages et skjult input felt som får navn stigende fra nr 1 og oppover, verdien som legges i disse feltene er navnene på argumentene, dette er gjort slik at det er enkelt å sjekke hvor mange argumenter som finnes og hva deres navn er.
- De inputfeltene som synes i nettleseren får navn som er lik tilhørende argument, derfor må vi bruke de feltene som står i forrige punkt for å hente ut de riktige verdiene av de forskjellige argumentene.

7.1.3.1.2 Hvordan vi gjorde det i XSL

For å starte med denne delen trengte vi en del bakgrunnsinformasjon, ingen av oss kunne noe om XSL og XSLT før vi startet med prosjektet. Vi startet her med et par WSDL-dokumenter som klienten skulle klare å tolke. For å kunne gjøre en transformasjon av et XSL-dokument måtte vi kjøre et script, et slik script tar imot WSDL-dokumentet og tolker det ved bruk av XSL-dokumentet, denne transformasjonen er definert som XSLT eXstensible Stylesheet Language Transformation. For å forklare litt bedre hva som egentlig skjer, har vi limt inn deler av XSL-koden og skrevet forklaringer til.

```
<xsl:template match="wsdl:portType">
<xsl:for-each select="wsdl:operation">
    <table border="0" width="50%">
        <xsl:variable name="method" select="substring-after(wsdl:input/@message,':')"/>
```

Kode 7.8

Det som skjer i Kode 7.8 er at XSL-filen først finner elementet i WSDL-dokumentet som heter ”portType”, grunnen til at vi ikke bare skriver ”portType” her, men faktisk bruker en namespace som kalles wsdl, er at denne namespacen er definert både i WSDL-dokumentet og i XSL-filen. Dette gjør at vi vet at det er et WSDL-element vi skal hente ut. Så startes en for løkke, denne skal gå for hvert element under portType som heter operation. Den siste linjen i koden viser hvordan vi kan hente ut tekst fra en streng og lagre denne som en variabel. I WSDL skrives ofte navnene på metodene som skal begynnes med f.eks. ”tns:metodenavn”, mens andre steder i dokumentet brukes kun metodenavnet. Derfor brukes substring-after til å hente ut de tegn som er gitt etter ’:’.
Denne WSDL-tolkningen er testet ut i Internet Explorer 6 og Opera 6.0, men siden Opera ikke har noen slik støtte for XML som IE6 har fungerer dette kun i IE.

7.1.3.1.3 Hvordan vi gjorde det i JSP

Denne løsningen ble gjort helt på slutten av prosjektet fordi vi fant ut at XSL-løsningen ikke klarte alle de krav vi hadde ønsket. Ved bruk av node behandling i Java kunne vi rimelig enkelt plukke ut akkurat de verdiene og de nodene vi ønsket, denne løsningen tar imot et filnavn og et metodenavn. Løsningen skriver kun ut en metode, hvis feil metodenavn er gitt med kommer en beskjed om at den medfølgende metode ikke finnes i den gitte WSDL.

7.1.3.2 Kommunikasjon med Web Service

Vi lagde først en klient som kommuniserte med en gitt Web Service, altså ikke en dynamisk som kan kommunisere med de fleste. Denne klienten laget vi først som en Java-klasse. Så konverterte vi denne over til JSP. Dette gjorde testing og feilretting enklere og raskere. For å lage en slik klient i Java må vi vite hvordan Java bruker SOAP til å kommunisere[25]. Vi vil her forklare hvordan vi har gjort en slik klient generisk. For at klienten skulle bli generisk må den ikke være avhengig av ett visst antall parametere eller at retur verdien fra Web-tjenesten må for eksempel være XML. XSL / JSP tolkeren av WSDL-filen måtte derfor hente ut hvilken type som skulle returneres. Dette fordi vi må behandle de forskjellige typene på ulike måter, ikke bare ved å caste de om til korrekte typer, men også SOAP kommunikasjonen må behandles forskjellig. Hvis returen fra en Web Service er av typen XML må vi sette encodingen til NS_URI_LITERAL_XML. Mens hvis returen er en streng, en integer eller liknende må den samme encodingen settes lik NS_URI_SOAP_ENC. Tjenesten blir kalt på med alle de argumenter og parametere som trengs, retur verdien blir så castet om til sin opprinnelige type. Deretter legges retur dataene i et skjult tekstfelt, slik at de kan sendes videre til tilstandsmaskinen som styrer hele prosessen. Her skrives det til skjerm at Web-tjenesten har returnert og at brukeren kan trykke neste for å fortsette til neste steg. Hvis brukeren ikke trykker innen 5 sekunder vil knappen automatisk bli trykket. Når en Web-tjenesten returnerer XML, må JSP-filen skrive om resultatet og legge det i en streng.

7.2 Bakgrunn for verktøyvalg

Når vi skulle velge hvilken server vi skulle bruke under prosjektet hadde vi to stykker å velge mellom, IBMs WebSphere Application Server (WAS) og

Apache Tomcat 4.0. Vi tenkte først at vi skulle velge å bruke mest mulig av IBMs verktøy og begynte derfor med WAS, etter mye testing, prøving og feiling kom vi frem til at denne løsningen ikke var slik vi hadde tenk. Vi hadde allerede brukt mye tid på å få serveren opp og få lagt ut de Web Servicene vi trengte til testing av prototypen, noe vi ikke fikk til å fungere på IBMs server. Vi valgte derfor å prøve å bruke Tomcat, og dette var en mye enklere og bedre løsning for oss. Tomcat er gratis og kan lastes ned av alle samtidig som den er veldig enkel å installere og sette seg inn i. Under utviklingen og testingen har vi også brukt IBMs server som er integrert i WSAD. Som utviklingsverktøy brukte vi WSAD som hovedverktøy og testverktøy, men fordi dette verktøyet krever veldig mye CPU kraft og minne synes vi det var enklere å programmere i små tekstbehandlingsverktøy som EditPlus, dette er en god editor som inneholder syntaks til kjente programmeringsspråk som Java og JavaScript. Til XSL-transformasjonen brukte vi en editor som heter XML Spy. WSAD støtter også XML, XSL og XSLT men allikevel valgte vi å bruke lettere verktøy for å spare tid og for å kunne gjøre flere ting samtidig uten å bruke opp alle maskinressursene. Vi valgte å bruke Sun sin JDKs Java kompilator fordi dette er en kompilator vi vet fungerer godt. Ved programmering av JSP filene var det beste å bruke WSAD sin editor, siden EditPlus ikke inneholder syntaks til JSP.

7.3 Prinsipper vi har fulgt

Vi har hele tiden fordelt oppgavene tydelig slik at vi begge har visst hva vi skal gjøre. Det har derfor blitt slik at vi for det meste har sittet og kodet hver for oss og veldig lite sammen. Når vi har kodet noe ferdig har vi så testet dette. Under hele prosessen har vi vært i god kontakt med oppdragsgiver, han har gitt oss mye tilbakemeldinger om hvordan han vil ting skal være.

7.4 Beskrivelse av forløp

Det meste av tiden har vi brukt på å lære oss nye ting, vi planla å bruke ca en måned på å sette oss inn i Web Services. Det viste seg at dette ikke på noen måte var så trivielt som vi trodde, vi har under hele prosjektet sittet og lest og lært oss nye ting, selv de siste ukene har vi måttet lære oss en del nytt. Når vi så følte at vi hadde kommet opp på et slikt kunnskapsnivå at vi forstod det grunnleggende begynte vi med kodingen og designet av prototypen. Vi lagde først noen Web Services som vi har brukt til testing av de andre delene av prototypen. Så begynte vi å lage den generiske Web Service klienten for å teste ut om de Web Servicene vi hadde laget virkelig fungerte. Samtidig som vi begynte med dette startet vi med å utforske hvordan vi kunne mappe FDL til WSFL. Begge disse arbeidene var mer tidkrevende og utfordrende enn vi hadde planlagt, og arbeidene har under

hele prosessen blitt forandret litt i forhold til de krav resten av prototypen stilte. Når vi var kommet så langt med arbeidet med de to første delene av prototypen begynte vi på tilstandsmaskinen. Også i dette arbeidet ble det mye lesing før vi kunne begynne, vi har lært en del om tilstandsmaskiner fra før, men ikke hvordan disse skal implementeres i for eksempel Java.

7.5 Endelig løsning

Den første versjonen av prototypen ligger på CD-romen som følger med rapporten. Det ligger også en gyldig WSFL basert på de regler vi har definert på CD-romen. Prototypen vil ikke fungere med dette WSFL-dokumentet da de Web-tjenestene som WSFL-dokumentet bruker ikke eksisterer. Web-tjenestene har kun eksistert i 3 måneder under prosjektet, helt til prøvetiden på DB2 gikk ut. For å få testet prototypen må en WSFL lages etter våre regler. Dette WSFL-dokumentet må da bruke Web Servicer som eksisterer og fungerer. Vi legger allikevel med en liten installasjonsguide til prototypen, vist i vedlegg **Feil! Fant ikke referansekilden.**.

8 Kvalitetssikring, testing og realisering

8.1 Organisering av kvalitetssikring

Kvalitetsskiring er studentenes ansvar. IBM kan gi veiledning slik at denne blir gjort i henhold til IBM sitt kvalitetssikringssystem og metodeverk. IBM baserer seg på IBM sitt kvalitetssikringssystem etter krav fra ISO 9001 og GSMethod. Prosjektet skal følge retningslinjene som kvalitetssikringssystemet og metodeverket gir. Ved å levere statusrapporter til veileder og kontaktperson sikrer vi at gruppa må jobbe kontinuerlig med oppgaven. Vår kontakt person på IBM er veldig engasjert og interessert i prosjektet vi skal jobbe med, så han vil til stadighet undersøke hva vi gjør og hvordan vi gjør ting.

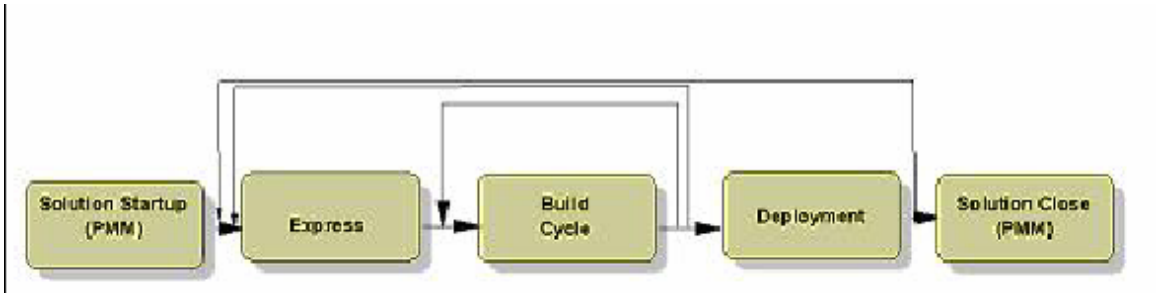
8.1.1 Kvalitetssikrings mal

- Vi har satt opp tidspunkter for å jobbe med prosjektet.
  Tirsdag – Fredag kl. 08:00 – 16:00 Felles jobbing med prosjektet
  Onsdag kl. 09:00 Møte med Frode Haug hvis nødvendig.
- Vi skal jobbe på IBM Hamar minimum to ganger i måneden.
- Vi skal levere tre statusrapporter til veileder.
- Gruppa skal hele tiden sjekke med oppdragsgiver når noe er uklart og når noe er ferdig utviklet for å få innspill og hjelp hvis det er noe vi ikke har forstått.
- Alle filer vi lager skal sikkerhetskopieres. Vi har sikkerhetskopier liggende på pc-ene samt at vi har sikkerhets kopi på en server vi har stående hjemme.
- All kode vi lager skal kommenteres slik at både vi og utenforstående får innsikt i hva koden gjør.
- Vi må regne med å jobbe også utover de tider som er satt opp for å rekke tidsfristene vi har satt opp.
- Vi leser gjennom det vi har skrevet for så å rette feil og mangler.
- Når vi har skrevet et kapittel sendes dette til oppdragsgiver slik at han kan se gjennom det vi skriver.
- Vi skal sammen med oppdragsgiver komme frem til den beste løsningen på prototypen.
- Gruppa skal følge IBMs GSMethods for kvalitetssikringen.

8.1.2 IBM Global Services Method Release 3.0 (GSMethod)

IBM sitt metodeverk for gjennomføring av utviklingsprosjekter er dynamisk og kan tilpasses kundens krav til metode. GSMethod skal brukes i

gjennomføring av prosjekter som ledes av IBM! GSMethod er en strukturert iterativ metode som omfatter og støtter prosessen for leveranse av ferdige løsninger. Metoden dekker i tillegg til utvikling og videreutvikling av eksisterende applikasjoner også implementasjon av "pakkeløsninger". Metoden er integrert med teknikker og verktøy for å skape et komplett miljø. IBMs utviklingsprosjekter og implementasjonsprosjekter baserer seg alle på denne utviklingsmetoden, som inneholder metode og prosesser for etablering av kundespesifikke løsninger. GSMethod fokuserer på prosjektleveranser (Work Products) og deres innbyrdes avhengigheter - det betyr at prosjektgruppen kan fokusere på selve utviklingen av løsninger. Det finnes forskjellige modeller i forhold til type prosjekter som skal gjennomføres. For små prosjekter med fokus på E-business er " e-business Express" en god modell. GSMethod e-business Express Engagement Model består av de faser som er vist i Figur 8.1 GSMethod e-business Express Engagement.



Figur 8.1 GSMethod e-business Express Engagement

- Solution Startup Omhandler nødvendige oppstartsaktiviteteter fra Prosjektledelse.
- Express Klargjør for build, ved utvikling av krav, analyse modell, arkitektur og fysisk design for release.
- Build Cycle Utvikling og testing av løsning.
- Deployment av systemet.
- Solution Close Inneholder krav til avslutningsprosedyrer fra PMM inkludert uttak av intellektuell kapital.

8.1.3 Kvalitetssikring av produkt

Vi kvalitetssikrer produktet ved å teste det vi har laget, etter hvert som det lages. Når deler av prosjektet er ferdig vises dette til oppdragsgiver for vurdering, hvis noe ikke er laget godt nok har vi diskutert dette med oppdragsgiver for å få nye ideer om hvordan problemet kunne løses bedre. Vi har brukt oppdragsgiver aktivt under hele prosessen for at resultatet skal bli mest mulig slik han ønsket. Testingen av prototypen vil bestå i å teste på de grunnlag vi vil at prototypen skal fungere. Det er prosjektgruppa som har

gjort all testingen av prototypen, vi har ikke hatt behov for å la utenforstående være med å teste. Oppdragsgiver har flere ganger underveis kommet med gode og nye ideer om hvordan vi kunne løse prosjektet. De fleste ideene har vi rukket å løse, men noen av de ideene som kom den siste tiden har vi ikke hatt tid til å se på.

8.1.4 Kvalitetssikring av prosess

Vi har holdt veldig få møter med veilederen vår på HiG, dette fordi den hjelp vi har trengt har han ikke vært i stand til å hjelpe oss med. Vi har heller brukt de ressurspersonene vi kjenner på IBM til hjelp, da de sitter inne med stor ekspertise på området. De møter vi har hatt med veileder har vært diskusjoner og veiledning til rapportskriving. Vi har fulgt malen vi har satt opp for kvalitetssikringen og følte at dette fungerte bra. Sikkerhetskopieringen har vi til tider ikke vært veldig nøye med, men siden vi har de fleste filene liggende på minst to pc-er har vi ikke sett noen nødvendighet i å ta sikkerhetskopiering veldig ofte. Vi har ofte jobbet mer enn de timene vi hadde satt opp, dette fordi det var veldig mye mer å sette seg inn i og lære seg før vi kunne begynne med selve produktet. Under hele prosessen har vi sittet og jobbet på den plassen vi fikk tildelt av skolen. Når vi har jobbet i helgene har vi ofte sittet separat, vi har da holdt møter etterpå for å oppdatere hverandre om de ting vi har gjort. Vi fant ganske tidlig ut at vi kom til å få dårlig tid og at vi måtte jobbe hardere enn først antatt. Under hele prosessen har vi holdt oppdragsgiver oppdatert om det vi har gjort, dette har vi byttet på å gjøre, både ved å sende mail og ved å vise hva vi har laget når vi har vært på Hamar. Fordi det var veldig mye å gjøre har vi hele tiden sittet ved hver vår pc, dette gjorde at vi fikk gjort mer enn hvis vi hadde sittet ved samme pc. Vi fikk litt problemer underveis da vi fant ut at ikke alle de løsningene vi hadde planlagt kunne fungere slik vi ønsket, alt vi har drevet med har vært nytt og det var veldig mye å lese.

8.1.5 Evaluering av kvalitetssikringsmal

Det var veldig vanskelig å sette opp en mal for hvordan vi skulle arbeide da det i starten var veldig usikkert hvordan prosjektet faktisk kom til å bli. Den malen vi satt opp er vi fornøyd med og har fulgt den ganske bra hele tiden. Statusrapportene skulle leveres til veileder i de tidsrom vi skulle ha vært ferdig med deler av prosjektet, de to siste statusrapportene leverte vi senere enn det som var planlagt. Statusrapportene ble levert da vi følte at vi hadde nådd milepæler, disse milepælene kom litt senere enn vi hadde antatt. Vi hadde ingen planer for hvordan vi skulle løse en eventuell sprekk i tidsrammen, dette er noe vi vil arbeide med i fremtidige prosjekt. Det at vi leverte den siste statusrapporten kort tid før vi skulle levere prosjektet var

egentlig ganske bra, for da fikk vi virkelig se hvordan vi i forhold til fristen for innlevering.

9 Diskusjoner av resultater

9.1 Har vi nådd målet?

Prototypen ble kanskje ikke så generell som oppdragsgiver hadde ønsket. Vi har allikevel gjennomført alle de punkter som vi beskrev i oppgavebeskrivelsen, men punktene kan utvides og utbedres senere. Oppgaven har forandret seg litt underveis ettersom det ikke var klart om alt vi skulle lage gikk an å lage, samt at standarden for WSFL egentlig skulle være ferdig før jul 2001. Ved prosjektslutt har standarden enda ikke kommet. Når en slik standard kommer vil denne bli et bra grunnlag for videre arbeid. Etter hvert som vi har jobbet med prosjektet har vi skjønt hvorfor akkurat en slik prototyp skal lages, den forkorter en arbeidsprosess med mange timer. Det største problemet for oss var tiden, det tok veldig mye lengre tid å sette seg inn i de forskjellige delene enn det vi hadde trodd. Vi har ikke skrevet veldig mye kode, dette fordi vi har måttet lære oss all teknologien bak Web Services, FDL, WSFL og hvordan lage tilstandsmaskiner i Java. Løsningen fungerer på de WSFL-dokumenter vi har laget og testet. For å få løsningen helt slik oppdragsgiver tenkte seg må dette prosjektet viderearbeides. Det kom innspill om mange mulige ting å legge til i løsningen, men dette har vi ikke hatt tid til. Løsningen skal fungere i alle nettlesere, men den delen av prototypen som bruker XSL fungerer foreløpig kun i Internet Explorer 6. Det er lagt veldig liten vekt på utseende av prototypen, det er funksjonaliteten som er viktig. Vi skal få frem hvordan prototypen fungerer på en forretningsprosess, dette mener vi at vi har gjort ganske bra. Det har vært veldig utfordrende å lage en slik parser vi har laget, selv om de to språkene som brukes kan ligne litt på hverandre. Det finnes svært lite informasjon og dokumentasjon angående definisjon og standarder i FDL. Dette har gjort at vi har måttet tolke FDL-dokumentene og selv finne ut hvilke kodelinjer som tilsvarer de forskjellige kommandoene vi trengte til WSFL-filen. Ingen av oss har tidligere prøvd noe modellering av forretningsprosesser. Det har vært vanskelig å finne noen som virkelig kan bruke BPM Workbench og FDL som vi har brukt som kilde for parseren.

9.2 Hva har vi gjort bedre enn forventet?

Den siste tiden av prosjektet har vi fått gjort mye mer enn forventet. Vi har fått laget mange Web Services for å teste ut løsningen vi har laget. De kunnskapene og det kunnskapsnivået vi nå ligger på ville vært veldig bra for en videre jobbing med temaet, da vi har måttet tilegne oss mye mer kunnskap og informasjon om temaet enn først antatt. Kommunikasjonen i gruppa har vært veldig bra under hele prosjektet. Vi har hatt god kontakt

med arbeidsgiver og veileder under hele prosjektet. Gruppearbeidet har gått bedre enn vi forventet.

9.3 Forslag til endringer

Underveis har vi hele tiden prøvd å løse problemet på best mulig måte. Det at vi faktisk ikke var helt sikre på om alle de mål vi hadde satt oss var mulige å gjennomføre, og at kunnskapsnivået har steget under hele prosjektet har gjort at vi vet hva som bør jobbes med videre. Dette har også gjort at vi har sett ting vi kanskje kunne gjort annerledes, men at det fort ville blitt veldig tidkrevende og gjøre om. De forskjellige delene av prototypen er alle mulige å utvide og forbedre.

9.3.1 Forslag til endringer i parseren

Parseren kunne vært laget mye mer fleksibel, slik at den kunne akseptert større variasjoner i FDL-dokumentene. Og hadde nok ikke vært så veldige vanskelig, hvis vi i starten av prosjektet hadde visst det vi vet i dag om forretningsmodeller og FDL. Vi kunne også lagt inn muligheter for at det resulterende WSFL-dokumentet bare blir sendt til standard utskrift slik at man kunne brukt applikasjonen i en automatisert prosess.

9.3.2 Forslag til endringer i tilstandsmaskinen

Med tanke på hastighet og stabilitet, så er det mye som kan gjøres bedre. Teoretisk sett skal det ikke være noe i veien for å ha mange samtidige brukere, men da dette ikke er testet noe videre vet vi ikke sikkert. Og ting viser seg ofte å være mindre stabilt enn antatt.
Med tanke på at WSFL enda ikke er standardisert, vil det være en god del endringer som må gjøres i fremtiden. Måten de ulike elementene plukkes ut av dokumentet og navnene deres må med stor sannsynlighet endres litt.

9.3.3 Forslag til endringer i Web Service klienten

Klienten ble til slutt laget med to løsninger som bruker to forskjellige måter å løse ett problem på, begge klientene har fordeler og ulemper.

9.3.3.1 Hvilken løsning er den beste?

Grunnen til at det ble laget to løsninger var at vi i den første løsningen ikke klarte å plukke ut bare en bestemt metode, dette fordi vi ikke fant ut hvordan

vi skulle sende med variable inn i XSL-transformasjonen. XSL-løsningen er rask og viser alle metodene i en tjeneste. Fordelen er kanskje at denne løsningen er vesentlig raskere enn JSP løsningen. I JSP løsningen kan man velge en metode å skrive til skjermen, dessuten er den mye mer fleksibel og man kan gjøre veldig mange flere ting enn man kan i XSL. JSP løsningen er den som passer best til akkurat vårt bruk i dette prosjektet for at brukeren ikke skal taste inn data i feil metode. Dersom brukeren skal prøve en tjeneste uten å vite hvilke metoder som hører til Web Servicen er det nok best å bruke XSL-løsningen. Hadde vi fått sendt med parametere til XSL-løsningen slik at kun en metode ble skrevet til skjerm ville dette ha vært den beste og raskeste løsningen.

9.3.3.2 Hvilke endringer kan gjøres?

Begge løsningene kan nok forbedres og gjøres bedre. Men dette er egentlig ikke veldig nødvendig for en slik prototyp. Begge løsningene vi har kommet frem til klarer å vise det vi mener er viktig. Den delen som kan endres mest er JSP-filen som kaller på Web Servicen. En ting som bør gjøres for at klienten skal bli mer generisk, er at den kan håndtere flere returtyper. I dag håndterer JSP-filen XML, Strenger og tall. Å utvide denne delen slik at den takler for eksempel arrayer ville vært en nyttig endring. Det å gjøre klienten slik at den viser mer informasjon ut på skjerm er en mulig utvidelse som vi ikke har hatt noe behov for i utviklingen av dette prosjektet.

10 Konklusjon

10.1 Hva har vi kommet frem til?

Prototypen vi har kommet frem til fungerer etter de premisser vi har satt. Vi synes den viser hvordan en forretningsprosess med Web Services vil fungere. Oppdragsgiver har under hele prosjektet kommet med forslag og hjelp når vi har trengt dette. Det er ikke alt vi har hatt tid til å utføre. Vi har lært oss veldig mye omkring Web Services. Dette gjør at vi nå ser hvorfor Web Services er en teknologi som kommer til å bli brukt mye i fremtiden. Vi regnet ikke med at vi kom til å bruke så mye tid på å lære oss teorien og teknologien bak prosjektet som vi gjorde. Brukergrensesnittet i nettleseren har vi brukt veldig liten tid på. Det er kun satt krav om at brukergrensesnittet skal være enkelt å forstå og oversiktlig, slik at en eventuell kunde forstår det grunnleggende. Vi har stor tro på at prototypen kan videreutvikles. I første omgang satt vi oss mål om at løsningen skulle fungere i Internet Explorer, den fungerer også i andre nettlesere som for eksempel Opera. Java er noe vi har jobbet med tidligere. Vi synes det er veldig bra å jobbe med Java og at dokumentasjonen er veldig god. JSP er også veldig bra å jobbe med. Vi er fornøyd med å få laget en parser som fungerer rimelig bra selv om vi ikke har funnet noen spesifikasjoner på FDL. Vi har kommet frem til at det er mulig, ved å følge de regler vi har satt opp, å lage en prototyp som enkelt viser flyten i en forretningsprosess. Det er mulig å parse FDL til WSFL, selv om det er en del ulikheter i språkene. Vi har stor tro på at prototypen kan videreutvikles til et produkt som IBM kan bruke som demonstrasjonsverktøy for sine kunder.

10.2 Hva har vi lært?

Vi har i løpet av prosjektet lært veldig mye. Om Web Services har vi lært hvordan de lages, brukes og beskrives. Vi har lært å bruke en del nye verktøy slik som WebSphere Studio Application Developer, DB2[14][17] og Apache Tomcat 4.0. I tillegg har vi prøvd å lære oss å bruke WebSphere Application Server, noe vi ga opp etter en stund. I stedet brukte vi Apache Tomcat 4.0. Vi har også sett litt på IBMs Web Services Toolkit[11], men vi fant ut at dette ikke var det riktige verktøyet å bruke for oss. Språk som XML, XSL, FDL, WSDL og WSFL har vi også lært oss. Programmering i JSP var noe vi ikke hadde gjort før. Vi har lært at en prototyp bør lages slik at den er enkel å videreutvikle. Ingen av oss hadde vært med å skrive en så stor rapport før. Nå har vi lært mye om å skrive rapport, arbeide i grupper og forholde oss til en oppdragsgiver. Samtidig har vi lært å implementere en tilstandsmaskin i Java Servlets ved bruk av sesjoner.

10.3 Hva kunne vært gjort annerledes?

Vi kunne laget en prototyp som viste mer til skjerm. Hadde vi funnet mer informasjon om FDL underveis kunne vi laget parseren slik at den ville fungere på alle FDL-dokumenter. Vi kunne lagt mer vekt på parseren, men da hadde vi ikke hatt tid til alt det andre vi har gjort. Vi ville også ha hatt bedre tid til prosjektet hvis vi hadde fått plass på grupperommet når vi ble lovet. Det tok nesten en måned over tiden før alt fungerte slik det skulle.

10.4 Bruk av resultatet og fremtidig arbeid

Prototypen bør videreutvikles og gjøres ferdig. Dette hadde vært en oppgave for en ny prosjektgruppe. Utgangspunktet til en videre jobbing med prototypen kan være dette prosjektet. Vi vil anbefale at de som videreutvikler denne prototypen har litt kunnskaper innenfor fagområdet før de går i gang. Koden kan sikkert gjøres mer robust og bedre. Det viktigste er å få videreutviklet parseren, dette bør gjøres av personer med kjennskap til forretningsprosesser og FDL. Prototypen vi har kommet frem til er et slags første utkast av noe som etter en videreutvikling kan brukes til demonstrasjon for kunder. Det er sikkert mer som kan gjøres med prototypen. Det er nok av oppgaver som er tenkt gjennomført til minst ett hovedprosjekt til. Det er også mulighet for noen ved IBM å jobbe videre med prototypen for å gjøre den akkurat slik de ønsker den skal være.

10.5 Hvordan arbeidet har vært

Det har, som forventet, vært veldig mye å gjøre. Vi har jobbet fast 4 dager i uken fra kl 0900 til kl 1600. I tillegg har vi brukt mye tid utenom til å lære oss teknologier. Den siste måneden har vi jobbet mer enn resten av halvåret. Vi er begge fornøyde med samarbeidet i gruppen. Kommunikasjon med veileder og oppdragsgiver har vært bra. Vi har begge to stått på for å gjøre prosjektet best mulig og ingen av oss synes den andre har gjort for lite. Vi har for det meste sittet og jobbet ved hver vår pc. Jobbingen har gått veldig bra også når vi har sittet sammen med oppgaver. Det har vært viktig at vi har gitt tilbakemeldinger på hverandres jobbing. Veldig viktig er det også å gi positive tilbakemeldinger og ikke bare gi kritikk. Gruppa søkte om å få arbeidsplass på skolen for å ha ett fast sted å jobbe med prosjektet. Grupperommet vi har jobbet på har vært veldig bra, og det er har vært et bra miljø under hele prosjektet. For å få bedre tid i prosjektet skulle vi kanskje ha vært en person til på gruppa, da kunne vi fordelt oppgavene slik at flere ting ble jobbet på samtidig. Slik vi har fordelt oppgaver har vi hele tiden hatt to oppgaver å jobbe med, dette for å få litt variasjon i arbeidet. Med flere personer i gruppa vil det oftere komme inn nye ideer om hvordan ting skal løses. Men vi har jobbet veldig bra sammen og angrer ikke på at vi kun var

to stykker. Det gjør det letter å kommunisere og fordele oppgavene. Samtidig er det enklere å avtale tider å jobbe på som passer for hele gruppa. Vi har lært hvor viktig det er med god planlegging når et prosjekt skal gjennomføres. Gruppen hadde ikke veldig mye erfaring med å planlegge et så stort prosjekt, noe som vises av forskjellene i de to Gantt skjemaene som vises i vedlegg B. Grunnene til de store forskjellene i skjemaene er at vi brukte mye lenger tid enn det vi trodde. Teknologien og systemene vi skulle sette oss inn i var mer kompliserte enn først antatt. Når vi skulle undersøke krav for FDL til WSFL brukte vi lang tid fordi det var veldig vanskelig å finne nyttig informasjon om FDL, derfor har også utviklingen av parseren tatt lenger tid en forventet. Dette har medført at deler av tiden vi skulle bruke til å teste prototypen isteden har blitt brukt til utvikling. Det tok nesten en måned før vi fikk det grupperommet vi var lovet av skolen, noe som også førte til at vi ikke kom skikkelig i gang med prosjektet da vi hadde planlagt. Vi har lært å takle motgang i prosjektet. Hvis en av oss har vært sliten eller lei har vi måttet passe på å ikke la dette gå utover den andre på gruppa.

Litteraturliste

1. Apache, XML Standard API, URL, 2002 http://xml.apache.org/xerces2-j/javadocs/api/index.html
2. Box, Simple Object Access Protocol (SOAP) 1.1, URL, 2000 http://www.w3.org/TR/SOAP/
3. Bray, Namespaces In XML, URL, 1999 http://www.w3.org/TR/1999/REC-xml-names-19990114/
4. Christensen, Web Services Description Language(WSDL) 1.1, URL, 2001 http://www.w3.org/TR/wsdl
5. Eckel, Thinking in Patterns- Problem-Solving Techniques using Java, 2001.
6. Endrei, IBM WebSphere V4.0 Advanced Edition Handbook, PDF, 2001
7. Grosso, XSL Concepts And Practical Use, URL, 2000 http://www.nwalsh.com/docs/tutorials/xsl/xsl/frames.html
8. IBM, Web Services – The Web's next revolution, URL, 2000 http://www6.software.ibm.com/developerworks/education/wsbasics/index.html
9. IBM, Web Service Introduction, PDF, 2002
10. IBM Web Services Architecture Team, Web Services Architecture Overview, PDF, 2002
11. IBM, Web Service Toolkit, URL, 2002 http://www.alphaworks.ibm.com/tech/webservicestoolkit
12. Jesse Liberty, XML Web Documents from scratch, QUE, 2000
13. Leymann, Web Services Flow Language (WSFL 1.0), PDF, 2002
14. Michel, Connecting WebSphere to DB2 UDB Server, PDF, 2001
15. Norsk EDIPRO, Infrastruktur for elektronisk Handel, 2002
16. Siddiqui, Deploying Web services with WSDL, Part 2: Simple Object Access Protocol (SOAP), URL, 2002 http://www-106.ibm.com/developerworks/webservices/library/ws-intwsdl2/?loc=dwmain
17. Shiral, DB2 UDB V7.1 Performance Tuning Guide, PDF, 2000
18. Sun Microsystems, Java Servlet Pages, URL, 2002 http://java.sun.com/products/jsp/
19. Sun Microsystems, Java Serlvets API Dokumentasjon, URL, 2002 http://java.sun.com/products/servlet/2.1/api/packages.html
20. Uche Ogbuji,WSDL Processing with XSLT,URL, 2000 http://www-106.ibm.com/developerworks/library/ws-trans/index.html
21. Ueli Wahli, Web Services Wizardry with WebSphere Studio Application Developer, PDF, November 2001
22. W3C, Exstensible Markup Language (XML), URL, 2001 http://www.w3.org/XML/
23. W3C, XSL Transformations, URL, 1999

http://www.w3.org/TR/xslt
24. Woodie, SilverStream Tackles New WSFL Spec, URL, 2002 http://www.midrangeserver.com/mso/mso040202-story06.html
25. XMethods, A Quick-Start Guide for Installing Apache SOAP, URL 2001, http://www.xmethods.com/gettingstarted/apache.html
26. XMethods, www.XMethods.net, URL, 2002 http://www.xmethods.net

Definisjoner

Bindig	Interaksjon som innebærer at en tjenestebru en lokaliseringsoperasjon å binde seg til en tjenestetilbyder og interagere med denne [15].
BPM	Business Process Modelling er en betegnelse på modellering av forretningsprosesser. Holosofx, et produkt som IBM markedsfører, gir en visuell presentasjon av forretningsprosesser. Den visuelle presentasjonen kan eksporteres til et tekstlig format (FDL) som kan tolkes av applikasjoner for videre bruk.
Classpath	Definerer en eller flere kataloger som en Java kompilator skal søke i for å finne klasser utover standardmiljøet.
FDL	Flow Definition Language er et proprietært språk som beskriver flyten i en forretningsprosess.
Forretningsprosess-modell	Detaljerte beskrivelser av de enkelte forretningsprosessene i en virksomhet[15]
Forretningsprosesser	En kjede av logisk sammensatte, repetitive aktiviteter som utnytter virksomhetens ressurser til å foredle et objekt (fysisk eller mentalt) med det mål å oppnå spesifiserte og målbare resultater/produkter til interne eller eksterne kunder[15].
JSP	Java Servlet Pages, ett tillegg til Java Servlets, her kan utviklere raskt og enkelt utvikle dynamiske og plattformuavhengige Web-sider ved bruk av Java. Må kjøres på en server som støtter JSP, sånn som Apaches Tomcat 4.0
Koreografi	Lovlig rekkefølge på de meldinger som sendes mellom ulike roller (aktører) i en samhandling[15].
Mappe/mapping	”Oversettelse” fra et dataelement i en samhandlingsmodell til tilsvarende dataelementer i en annen samhandlingsmodell.[15]
Parser	Bryter en tekst opp i kjente elementer for å kunne foreta en tolkning.

Prototyp	Utkast eller grunnform av noe som videre.
RPC	Remote Procedure Call. Fjernmetodekall – blant annet benyttet i SOAP – som kjennetegnes ved at mottagende applikasjon forventes å sende et svar som tilfredsstiller den sendte forespørselen[15].
SOAP	Meldingsprotokoll basert på XML og RPC (Remote Procedure Call) som spesifiserer hvordan tjenester kan snakke sammen i en Web Service-infrastruktur ved å definere regler for hvordan en melding ser ut og hvordan data skal kodes[15].
StyleSheets	Beskriver regler for et stilsett som skal brukes f.eks. i et dokument, og dokumentet henviser man bare til de ulike reglene. Hvis man vil forandre på noe i stilsettet, så gjøres dette kun i stylesheetet og alle steder blir riktig.
UDDI	Spesifikasjon for Web-baserte informasjonsregistere for Web Services som gjør det mulig for utviklere og administratorer på en enkel måte å dele informasjon om interne og offentlige ”Web-tjenester” hhv. på tvers av foretaket og over Internett.
Web Service	Web Services er selvbeskrivende applikasjoner som kan bli publisert, lokalisert og påvirket over internett. Når en Web Service er lagt ut på nettet kan andre applikasjoner (og andre Web Services) lokalisere og bruke denne tjenesten. Et eksempel på Web Services er kredittsjekkings system som returnerer kreditt informasjon når et personnummer er tastet inn.
WSAD	WebSphere Studio Application Developer, IBM utviklingsverktøy som brukes til utvikling av bl.a. Java, XSL, XML, JSP og litt til Web Services.
WSDL	Web Services Description Language er et XML-basert språk for å beskrive grensesnitt til Web Services grensesnitt på et XML-format. Brukes til å spesifisere abstrakte operasjoner som deretter bindes til konkrete protokoller, f.eks. SOAP[15].

WSFL	Web Services Flow Language beskriver flyten i en forretningsprosess med "koreografien" mellom et sett med Web Services.
XML	eXstensible Stylesheet Language er et språk for å beskrive innholdet av dokumenter. XML-dokumentet beskrives syntaktisk som et sett informasjonsobjekter (elementer i XML-terminologi), der hvert objekt identifiseres med en start element tag og en tilhørende avslutningselement tag[15].
XSL	eXstensible Stylesheet Language bygger på Cascading Style Sheets CSS som html bruker.