Universitetet i Oslo
Institutt for informatikk

Cand.Scient Thesis

Arkitekturer innen Internett-programmering for e-handel innen turisme

Tom-Erik Valsø

09.02.2004

FORORD

Jeg vil takke Almira Karabeg og Dino Karabeg for entusiastisk og klargjørende veiledning gjennom hele denne oppgaven. Av deres arbeid har jeg lært mye om Internetts viktighet og konsekvenser både i og utenfor forretningsverdenen, samt fått inspirasjon til å lære mer om dette. Jeg ønsker også å takke Naci Akkøk for ha gitt klare retningslinjer for bruk av de teknologiene jeg benyttet, og klargjort i de tilfellene der jeg ikke uttrykte meg tydelig nok i forhold til teknologivalgene.

Fra SoftwareAG vil jeg takke Jacob Bakstad og Lars Jelgren for meget god oppfølging rundt Tamino XML Server og for raske og utfyllende svar om bruk av både XML, XML databaser og Web Services teknologi.

Jeg vil også takke mine foreldre som har støttet meg økonomisk mens jeg avsluttet denne oppgaven, og ikke minst for interessen de har vist for det jeg har drevet med. Jeg vil også takke Heidi Kjølsrød for å ha korrekturlest denne oppgaven uttallige ganger.
Takk også til Elin Iversen for hjelp til design av illustrasjoner, og Andreas Bade og Tor Erling Nygård for viktige tilbakespill i bruken av AXIS, HTML, XML, XSL og Java.

Innhold

Kapittel 1

Innledning

1.1 Beskrivelse av problemet

De siste fem-seks årene har det skjedd voldsom utvikling innen bruken av Internett, og mange teknologier og programmeringsspråk er utviklet for å gjøre Internett enklere for sluttbrukeren, billigere for de som vil kjøpe systemene og lettere for utviklerne av Internettløsningene. Man kan i dag velge mellom titalls programmeringsspråk som muliggjør å tilgjengeliggjøre data på Internett, og de fleste aktørene i næringslivet bruker nå denne teknologien i utstrakt grad.

To teknologier som har kommet de siste årene og som har vært mye omtalt i Internettsammenheng er XML og den XML-baserte teknologien Web Services, og jeg ønsket å undersøke hva disse teknologiene kunne bidra med i arbeidet med å lage Internettapplikasjoner.

Det stilles i dag høye krav til kvaliteten på Internettapplikasjoner fra brukerne, og firmaer og organisasjoner må strekke seg stadig lenger for å oppfylle dette. Kravene til personlig tilpassede Internettapplikasjoner er stigende, for eksempel med språktilpasning, og dette er dyrt og omfattende å utvikle. Jeg ville derfor undersøke om XML kunne forenkle utviklingen av slike personifiserte Internettapplikasjoner, og dette sto sentralt i denne oppgaven.

En annen side av Internettapplikasjoner jeg ville undersøke, hadde utgangspunkt i at mesteparten av Internettsidene og applikasjonene som utvikles på Internett i dag er bygget på de samme prinsippene med basis i sentraliserte systemer som ikke er tilgjengeliggjort utover det man kan

se når man går inn på den aktuelle URL-adressen.

Mye tyder på at det i Internettsammenheng er på tide å utvide perspektivene og å tenke mer nytt. Fra å betrakte Internett som en samling isolerte Internettsider med informasjon, på linje med bøker eller brosjyrer, bør man gå over til å tenke at Internett kan være en samling dynamiske data som er kilde for distribuert informasjon, og der man som bruker av Internett ikke skal måtte lete fra Internettside til Internettside for å finne fram. I stedet burde datamaskiner kunne hjelpe deg med denne jobben. Det er lagt ned et stort arbeid i dette de siste årene, og XML og Web Services var spådd å bringe oss et skritt nærmere denne visjonen om et Internett der programmer søker automatisk på Internett for å samle informasjon dynamisk.

Det er viktig å fremheve at dagens Internett har uttallige gode nettsteder med høy brukervennlighet og bra design, men at denne oppgaven handler om å se et skritt videre, for å se hva annet som kan legges til dagens Internett for å supplementere det med ytterligere funksjonalitet og kvalitet, samt forenklinger, i håp om å peke på områder der den nye teknologien, slik XML og Web Services er, skal forenkle og forbedre mulighetene innen Internettprogrammering.

Ved å flytte fokus vekk fra den standardiserte tankegangen, som er meget mye brukt, ville jeg prøve å se på måter å bruke denne nye teknologien på.

Internett har fortsatt stort potensial som er langt hevet over fragmenterte enheter av data, men fram til i dag har den distribuerte teknologien ikke blitt så populær som man kunne ønske. I dag er Internett fortsatt slik at mye informasjon som burde vært integrert, ligger på forskjellige Internettsider, og jeg ønsket å se på muligheter for å integrere informasjon, og som et eksempel for å lage noen prototyper, brukte jeg en gruppe mindre bedrifter innen autentisk turisme.
Motivasjonen for å få til dette i min oppgave, var å undersøke muligheter den nye teknologien kunne bidra med i utviklingen av Internettapplikasjoner.

Mine tre fokusområder i denne oppgaven var derfor å oppnå:

- Lavere kompleksitetsnivå for utviklerne av Internettapplikasjoner
- Økt bruk av personifisert informasjon for sluttbrukerne

- Lettere tilgang til informasjon for sluttbrukerne

Som eksempel i denne oppgaven brukte jeg et prosjekt som heter Authentic Norway Project.

Denne oppgaven gikk ut på å se på de overnevnte teknologiene, og jeg lagde prototyper som omhandlet forretningsdrift på Internett, og arbeidet kan derfor ses i relasjon med e-handel, men det er viktig å fremheve at denne teknologien ikke er bundet til å måtte gjelde e-handel eller turisme, som er det Authentic Norway arbeider med, men også liknende problemstillinger generelt for Internettutvikling.

Jeg vil i slutten av oppgaven komme med en anbefaling til den eller de teknologiene jeg undersøkte som jeg mente hadde mest potensial til å kunne bidra med nyskapende og bra funksjonalitet til Internettapplikasjoner i dag.

1.2 Hva implementerte jeg

Jeg ønsket å lage en prototyp for hvordan en virtuell bedrift, bestående av flere mindre bedrifter, kunne samarbeide ved å samkjøre og integrere informasjon og tjenestene de tilbød , slik at det skulle være mindre jobb for kundene å kunne sette sammen aktiviteter og overnatting for et ferieopphold. På denne måten ville ikke bare tilbudet bli bedre for kunden, men reisedestinasjonene kunne samkjøre informasjon om perioder der kundeantallet var høyt eller lavt, og på den måten dele erfaringer og planlegge kampanjer og aktiviteter.

I to av prototypene forutsatte jeg at alle deltagerne brukte samme Internettapplikasjon og delte database til reservasjon, mens i min tredje prototyp ville man kunne bruke selvstendige systemer som kommuniserte med hverandre over Internett via Web Services.

Det var også mulig å lage en hybridløsning der noen brukte en sentral database, mens noen av bedriftene hadde egen bestillingsløsning i tillegg, men at de tilgjengeligjorde utvalgte data ved hjelp av Web Services, men jeg implementerte ikke en slik prototyp. Dette hadde blitt en kombinasjon av prototyp 1 og 3.

Jeg ønsket i denne oppgaven å sammenligne tre arkitekturer der èn bru-

kes mye på Internett i dag og to er mer uvanlige, for å undersøke deres styrker og svakheter.

Det er viktig å fremheve at det på Internett finnes svært mange teknologier å velge mellom som utvikles av store programvareleverandører, både innen distribuerte og ikke-distribuerte Internettapplikasjoner. Det ville være umulig å lage et sammendrag om disse, men IBMs WebSphere, Suns J2EE, CORBA og Microsofts .Net er eksempler på viktige teknologier i Internettapplikasjoner.

Årsaken til at jeg har valgt teknologiene som er brukt i de tre prototypene, er at jeg mente at disse kunne illustrere 3 prinsipper innen Internettprogrammering som var fundamentalt forskjellige, og at de kunne implementeres så enkelt at de viktige prinsippene ville komme frem.

Den første prototypen var den desidert mest utbredte teknologien av de tre jeg hadde valgt.

Med prototyp 2 ville jeg undersøke mulighetene for å basere en Internettapplikasjon på XML og å bruke XML og XSL-transformasjoner for personifiserte visninger av dataene.

Den siste prototypen var basert på at i de siste årene, fra XML ble lansert i 1999, har man hørt mye om at XML og Web Services kan brukes til forenkling av systemintegrasjon. Jeg ønsket med denne prototypen å sammenligne hvordan teknologien "Web Services" egnet seg i praksis til å integrere distribuerte data. Web Services muliggjør integrasjon og kommunikasjon mellom IT-systemer på en ny måte, fordi man på en mye lettere måte kan sende data mellom systemer som ligger på forskjellige steder på Internett. Jeg ville derfor implementere disse tre prototypene, der den første var svært veldokumentert og uttestet, mens de to andre var nyere og i raskere utvikling.

- Prototyp 1: En sentralisert 2-lags Web applikasjon med relasjonsdatabasen MySQL, implementert i HTML og PHP

- Prototyp 2: En sentralisert 2-lags Web applikasjon med XML databasen Tamino, implementert i HTML og PHP

- Prototyp 3: En distribuert 2-lags Web applikasjon som kommuni-

serte via Web Services på Internett over SOAP, implementert med AXIS, HTML, PHP, Java og relasjonsdatabasen MySQL.

Jeg ønsket å sammenligne kompleksiteten og gevinsten ved disse to prototypene (2 og 3) mot prototyp 1, som er en mer tradisjonell måte å utvikle Internettsider på, der man har applikasjonen og en relasjonsdatabase i samme lokalnettverk(LAN).

1.3 Formål med oppgaven og oppgavens problemstilling

Målet med denne oppgaven var å undersøke Internett-teknologier, og ved å gjøre dette ville jeg prøve å finne den beste løsningen for en Internettportal for Authentic Norway Project. Denne løsningen skulle inneholde funksjonalitet basert på CRM-tankegang, samt ha en fleksibel arkitektur for å kunne bygges videre ut. Jeg ønsket å se på hvordan de nye teknologiene kunne støtte opp, og forbedre de forretningsprosessene som de potensielle deltagende bedriftene hadde.

Jeg implementerte samme Internettapplikasjon med tre forskjellige arkitekturer, for å se på fordelene og ulempene som var relatert til hver av disse.

I Authentic Norway Project var de potensielt deltagende bedriftene små firmaer, som ikke hadde økonomiske forutsetninger til utvikling av dyre IT-løsninger. I dette prosjektet ønsket jeg å finne en arkitektur som kunne være utgangspunkt for en IT-basert forretningsløsning som styrket deres situasjon innen e-handel.

Dersom jeg klarte å finne arkitektturen for en e-handelsløsning for disse selskapene, slik at de kunne nå ut med og samkjøre sitt budskap i tillegg til å kutte vekk unødvendige arbeidsrutiner, ville disse bedriftene få en sterkere konkurransesituasjon.

Å velge riktig arkitektur for å utvikle en Internettapplikasjon er svært essensielt, fordi et riktig valg her vil være avgjørende for videre muligheter innen samarbeid mellom bedriftene, samt at det vil legge rammene for videre utvikling. Dette kan bety "å være eller ikke være" for de bedriftene som opplever vanskeligst konkurranse mot de store internasjonale selskapene.

Dette var en oppgave som ikke kun fokuserte på teknologien, fordi jeg i mitt eksempel ønsket å se oppgaven i forretningsmessig sammenheng for en liten til mellomstor bedrift som ville satse på Internetteknologi, og jeg prøvde å beskrive kompleksiteten i det tre foreslåtte arkitekturene.

Problemstilling:

Tre prototyper til en ny forretningsløsning på Internett for en liten til mellomstor virtuell bedrift sammensatt av mindre bedrifter eller enheter har blitt implementert og evaluert. Oppgavens problemstilling er å avgjøre hvilken av de tre prototypene, eventuelt en kombinasjon av de eller ingen av de, er det som er best egnet til applikasjonen som er valgt.

1.4 Kapitteloversikt

I kapittel 2 skriver jeg om bakgrunnen for prosjektet, samt den teorien fra turisme og kundebehandling som ble vektlagt som viktig for dette prosjektet.

I kapittel 3 skriver jeg om viktigheten av riktig arkitekturvalg, og hva jeg ønsket å programmere i mine prototyper.

I kapittel 4 gjennomgår jeg den arkitekturen med HTML, PHP og MySQL, samt presenterer prototyp 1 implementert i denne teknologien.

I kapittel 5 gjennomgår jeg prototyp 2, som er basert på en XML datastruktur, der dataene ligger lagret i en XML database. Jeg skriver også om XML generelt og Tamino XML Server.

I kapittel 6 gjennomgår jeg prototyp 3, som er basert på Web Services. Her skriver jeg om hva Web Services er, samt kort om valget av AXIS og Java.

I kapittel 7 beskriver jeg fordelene og ulempene i prototypene som representerer de tre arkitekturene. Her omtaler jeg blant annet kompleksitet.

I kapittel 8 er konklusjonen, samt forslag til arbeid videre.

1.5 Begrensninger og forutsetninger

- I denne oppgaven arbeidet jeg med arkitekturer og funksjonalitet, og prototypene er derfor ikke komplett implementert designmessig slik de er tenkt å se ut.

  Jeg har med vilje tatt bort de fleste designelementer, fordi jeg så på dette som lite relevant i forhold til formålet med prototypene. Det som ble gjort av design var å sette bakgrunnsfarger på HTML-sidene og HTML-tabellene, slik at man lett kunne se i hvilken HTML-tabell funksjonaliteten lå.

  Design er svært viktig i alle Internettapplikasjoner, og mine arkitekturforslag skal underbygge en arkitektur der de som designer applikasjonen har stor fleksibilitet til å lage ønsket design.

- Det er også viktig å fremheve at problemstillingen ikke er gjensidig ekskluderende. De tre prototypene er laget slik at de kan integreres sammen, men i min oppgave vil jeg evaluere hver implementasjon hver for seg.

- I denne oppgaven refererer jeg også til prototypene som en som Internettportal og portal. Hvorvidt applikasjonen vil være så stor at den med rette kan kalles en portal er et definisjonsspørsmål, men slik jeg tenkte er den ment som en portal, fordi den er en samling av ulike bedrifters forretning på Internett.

- I prototypene jeg lagde forutsatte jeg at kunden hadde valgt reisedestinasjon.

- Når jeg skriver om kompleksitet og prøver å bedømme om den er lav, middels eller høy, er dette kun relativt til prototypene innbyrdes, for å skille på disse. Kommentarene om vanskelighetsgrad er ikke ment å gjelde utover sammenligningen mellom disse prototypene.

- Det er også viktig å fremheve at blandingen av språk mellom norsk og engelsk, som dessverre kan virke slurvete, er bevisst. Etter å ha prøvd litt frem og tilbake med oversettelser bestemte jeg meg for å bruke det engelske ordet Web Services, fordi jeg tror det er

et ord med klarere betydning enn for eksempel Internett Tjeneste. Jeg bruker også noen andre engelske ord som Web og Server fordi den engelske terminologien også ofte brukes i norsk sammenheng.

- Jeg vil i denne oppgaven tidvis forkorte Authentic Norway Project til ANP for at det skal bli lettere for leseren.
- Prototypene jeg lagde var tenkt å ligge som en del av en større Internettportal.

Kapittel 2

Bakgrunn om turisme, salg og CRM

"Nå som det er blitt vanlig å drive forretninger på Internett, er det ikke lenger snakk om e-business eller e-commerce, det er bare business og commerce"[28, Finger og Arnonica,2001].

Med dette mener forfatterne av boken sitatet er hentet fra, at e-handel og handel er så sterkt knyttet at det er unaturlig å skille på de, og dette er et viktig fokus å ha i min oppgave også.

2.1 Authentic Norway Project

Authentic Norway Project er et samarbeidsprosjekt mellom Kunst og Håndverkskolen, Institutt for Informatikk og aktører innen autentisk turisme, og det var dette prosjektet jeg brukte som eksempel når jeg studerte ulike Internetteknologier.

Jeg ville bruke arbeidet jeg gjorde til å se på hvordan dette kan brukes i en reell sammenheng, og framover i oppgaven vil jeg vise resultatene av det jeg kom fra til i en tenkt Internettapplikasjon brukt av Authentic Norway Project. Mitt utgangspunkt for oppgaven var at ved å forenkle bestillingen og planleggingen av et reiseopphold tiltrekker man seg trolig en større gruppe av potensielle kunder til disse reisedestinasjonene. Dersom man får til dette vil etter all sannsynlighet antallet turister stige til denne delen av bransjen.

De 3 prototypene var således ment i passe inn i en større helhet der man brukte erfaringer fra e-handel, markedsføring og CRM for å forenkle og forbedre arbeidet til de deltagende bedriftene, og i resten av dette

kapittelet følger bakgrunnsstoff som har motivert og påvirket prototypene, og jeg ønsket å finne den arkitekturen som i høyest grad kunne understøtte disse prinsippene.

Dersom man integrerte informasjon på en internettside som ellers ville ligget på flere, så kunne man gjøre bestillingene lettere for kunden. Jeg ønsket også å undersøke kompleksiteten ved å samle alle bestillinger på èn transaksjon, og det har blitt gjort et stort arbeid med dette av store reiseoperatører. For eksempel på Internettsiden www.opentravel.org kan man lese om reiseoperatører som har lagt ned mye arbeid i å få med alle bestillinger på èn transaksjon med èn id. De mindre reiselivsselskapene som ikke har budsjetter til IT-utvikling har sjelden deltatt i dette arbeidet.

Innen turisme, som denne oppgaven omhandler, opplever de minste aktørene i markedet en hardere og hardere kamp mot de største selskapene som har et teknologisk forsprang.

2.2 Bakgrunn om turisme

Det er først de siste 50 årene, og spesielt de siste 20 at å reise over lange distanser har blitt mulig for flertallet av befolkningen. Det er derfor oppsiktvekkende at reising og turisme(eng:Travel and Tourism) er verdens nest største industri (dessverre etter våpenindustrien). Dette sier mye om ønsket mennesket har til å reise og se nye steder og den eventyrtrangen som bor i oss. Tall tyder på at denne bransjen fortsatt er i sterk vekst, men det er viktig å være klar over at dette varierer sterkt fra land til land, og at Norge er et av de landene som ikke kommer optimalt ut i analyser om reising og turisme i 2004, samt årene framover.

I denne oppgaven ville jeg se nærmere på en liten del av denne bransjen, autentisk turisme, og prøve å integrere denne med elektronisk handel. E-handel oppsto teoretisk sett på slutten av 1960-tallet med EDI systemer (Electronic Data Interchange), men det var først på midten av 1990-tallet at e-handelen fikk sin store oppsving med utbredelsen av Internett. E-handelen må derfor anses å være en svært fersk bransje. I tillegg har den vært kostbar og lite tilgjengelig for de fleste bedrifter fordi investeringer i slike løsninger dessverre ofte har vært dyre.

The World Travel and Tourism Council (WTTC) anslår at den totale et-

terspørselen (demand) til reising og turisme (fra nå forkortet RT) vil nå USD 4.5 trillioner i slutten av 2001 og stige til USD 9.3 trillioner i 2011 [21, WTTC, 2003]. Dette er 36 tusen milliarder norske kroner i 2001, og den er altså spådd til å nå 74 tusen milliarder i 2011. Etterspørselen og omsetningen steg 5-9 prosent i 2000, og den er spådd å stige 4 prosent per år det neste tiåret. I tillegg skaper reising 207 millioner jobber direkte eller indirekte (i 2001). Denne bransjen er overveldende stor, og "The World Travel and Tourism Council "skriver: "Reising og Turisme er verdens viktigste skaper av velstand"[20, WTTC, 2003].

Fordi reising er verdens nest største industri har det utviklet seg multinasjonale selskaper i denne industrien også, men samtidig er reising en bransje som har et potensial til variasjon som er helt unikt. I Europa består 99 prosent av alle firmaer av under 10 personer, og dette skaper stort mangfold og variasjon, også innen hva reiseselskapene kan tilby.

Mye av problemet er likevel at det ikke er like lett for en vanlig forbruker å finne en komplett løsning med et bra tilrettelagt tilbud på steder der antallet besøkende ikke er like stort som på masseturismedestinasjonene. Det er slik at når det er mange turister og besøkende, er det lettere for bedriftene som arrangerer reiser å lage et tilbud som er allsidig og variert, fordi det er så mange turister at det vil lønne seg økonomisk selv om for eksempel bare 5 prosent av turistene deltar. Slike reisedestinasjoner har derfor ofte å ha brede tilbud til folk som ikke ønsker å bruke mye tid på å planlegge reisen. Dette gir de et stort konkurransefortrinn. Typiske ferieparadiser eller solrike øyer som Kanariøyene er steder der turoperatørene har stor omsetning fordi tilbudet er lett tilgjengelig for turistene.

Dette gjør at en del mindre bedrifter innen turisme har problemer med å tiltrekke seg kunder, fordi det er vanskelig å økonomisk organisere så mange aktiviteter for en mindre gruppe mennesker, samt å reklamere for seg på samme måte som store kjeder eller selskaper kan. Store reiseselskaper har kommet langt i sine prosesser med å lage et effektivt apparat som skal ta seg av turistene, for eksempel "siste rest" plasser der man kommer til et hotell som er uspesifisert og likevel fungerer det. Dette er økonomisk gunstig og effektivt.

Allikevel er det mange mennesker som synes slike pakketurer er en for lite kreativ måte å feriere på. Masseturismedestinasjonene kan ikke tilby det disse turistene ønsker seg dersom de ønsker å se steder uten mye

turisme. Da ønsker en del mennesker i stedet en ferie som er mer rettet mot å se steder slik de egentlig er, og kanskje å oppleve mer "ekte" stedsopplevelser og naturopplevelser enn det de store feriestedene kan tilby. Slike reiser mot mindre destinasjoner sier man at er mer “informasjonsintensive”, og slik er bedriftene i Authentic Norway Project.

Noe av problemet er at man ikke like lett kan finne en slik reise ved å dra til nærmeste reisebyrå . Man har kanskje heller ikke sett mange reklamer for de mindre hotellene eller reisedestinasjonene, fordi de ikke har reklamert i de store kommunikasjonskanalene som radio, aviser og tv. Det kreves en større innsatsvilje for den som skal reise å finne et slikt sted, og man er ofte mer usikker på hva stedet egentlig har å tilby, fordi disse stedene sannsynligvis ikke er like tilrettelagt for turister som de store stedene er. Å skape basisen for en IT-løsning for "Authentic Norway"-bedriftene som underbygger samkjørt informasjon med felles markedsføring var et mål i denne oppgaven.

Jeg tror det er en utfordring for de mindre destinasjonene som selger opplevelsesreiser at folk skal kunne se og oppleve ting på disse stedene uten å bruke for mye energi på planlegging, og dette var noe av det viktigste i prototypene jeg lagde. Mange turister ønsker ikke å bruke mye tid på å planlegge tilbudene og aktivitetene til en ferie. De ønsker å oppleve mye i feriene uten å måte gjøre så mye selv, og derfor kan det være lett å velge en pakketur til for eksempel til Syden. Derfor var det et mål at Authentic Norway Project skulle forenkle bestillingsprosessen for kundene.

Turismens betydning for lokalsamfunn og økonomi

Det er på alle måter ønskelig å få økt turismen til Norge. Denne turismen er en viktig inntektskilde både i hovedstaden og utenfor, og i år med dårlig turisme er det mange som har dette som sitt levebrød som merker det sterkt negativt økonomisk. Norge er ikke et tettbebygd land, og arealmessig og naturmessig har vi mye å tilby turistene. Det er derfor med et sterkt ønske man ønsker flere turister, men dette er ikke noe som kommer av seg selv. Sterk langsiktig merkevarebygging og markedsføring må til i dager da media og reklame styrer mye av de ideene folk får. Derfor ser jeg på det som riktig at man starter et arbeid med å lage og distribuere informasjon og tilrettelegge for turister i enda større grad enn det som gjøres i Norge i dag.

Agrawal [2, Agrawal, 2002] refererer til tall fra Verdensbanken når han

skriver: “En pekepinn til viktigheten og betydningen av turisme er at på verdensbasis står turisme for mer enn ti prosent av det globale brutto nasjonalprodukt”. Internett er en del av denne forretningen, men potensialet for Internetthandelen til å øke er stort fordi løsningene som brukes i dag ofte har mange misfornøyde brukere.

Det er en lang liste med problemer som må løses før Internett er i nærheten av å ha nådd sitt potensial. Selv om elektronisk handel og CRM fortsatt ikke er optimalt kundevennlig, må Internetteknologi nøye tas med i betraktningen når man ser på reising og turisme. 30 prosent av alle e-handelstransaksjoner er reisning[3, Pollock, 2002], og potensialet for å vokse er fortsatt stort.

Å selge en ferie er annerledes enn å selge et “håndfast” produkt, fordi i en reise inngår mange selskaper for å fullføre ferien for kunden, og dette er et sentralt tema i min løsning, der bedrifter på reisedestinasjonene skal samarbeide og koordinere for at kunden skal få et best mulig tilbud.

Salg krever en grundig dialog mellom selger og kjøper for å gi kunden en opplevelse som er så bra at hun eller han vil komme tilbake for å kjøpe mer. Å etablere en slik dialog har blitt praktisert og forsøkt forbedret i alle år, men Internett kan hjelpe til å gjøre dette lettere. Et eksempel på dette er 1:1 kunderelasjoner og 1:1 markedsføring. Internett med sine kanaler som e-post, SMS og potensial til å forandre stil og innhold raskt på firmaers hjemmesider muliggjør en effektiv oppfølging av kundenes ønsker og krav.

Et viktig poeng er at ANPs IT-løsning skal ha en arkitektur som støtter CRM på en effektiv måte.

2.3 Tall fra World Travel and Tourism Council

Motivasjon for Authentic Norway Project er tildels basert på økonomiske rapporter og prognoser. I dette avsnittet følger noen fakta om turisme i Norge basert på The World Travel and Tourism Councils (WTTC) rapport om Norge fra 2003.

WTTC og Oxford Economic Forecasting har laget en rapport om Norge som heter: “Norway - Travel and Tourism - A world of opportunity”, der de så på omfanget av RT. Her blir det lagt fram tall over investerin-

2003:	2004-2013:
227,7 milliarder NOK (inkludert 1,2 prosent vekst i 2003).	3,5 prosent vekst per år som vil bli 400,6 milliarder NOK i 2013.

Tabell 2.1: Etterspørsel/demand (kilde: WTTCs rapport om Norge fra 2002).

RT-industri 2003	RT-industri 2004-2013	RT-økonomi 2003	RT-økonomi 2004 -2013
41,9 milliarder NOK	2,8 prosent økning per år som blir 69 milliarder NOK i 2013	163,9 milliarder NOK	2,8 prosent økning per år som blir 264,9 milliarder i 2013

Tabell 2.2: Omsetning (kilde: WTTCs rapport om Norge fra 2002)

ger, inntekter, sysselsetting de siste årene, samt prognoser for årene fra 2003 til 2013.

WTTC bruker to forskjellige termer når de skriver om Reising -og Turismens omfang i økonomi samt for samfunnet ellers. De bruker termene "Travel and Tourism Industry" og "Travel and Tourism Economy", som jeg fra nå referer til som "RT-industri" og "RT-økonomi", der den første er industrien og arbeidsplassene som er direkte involvert i RT, det vil ansikt til ansikt med turister og reisende, mens "RT Økonomi" inkluderer alle de som indirekte er involvert , både når det gjelder investeringer og arbeidsplasser for eksempel matproduksjon, bygningsarbeid og regnskapsfirmaer.

Viktige konklusjoner i rapporten var:

1a) I 2003 er Norges RT-industri forventet å utgjøre 2,6 prosent av brutto nasjonalprodukt (BNP) (41,9 milliarder NOK av 1611,5 milliarder NOK).

1b) I 2003 er RT-økonomi forventet å utgjøre 10,2 prosent av brutto nasjonalprodukt (163,9 milliarder).

2a) Antall jobber 2003 i RT-industri tilsvarer 2,3 prosent av samlet sysselsetting

RT-industri 2003	RT-industri 2004-2013	RT-økonomi 2003	RT-økonomi 2004-2013
74.167 arbeids-plasser	0,8 prosent økning per år som blir 80.068 arbeidsplasser i 2013	310.715 arbeidsplasser	0,6 prosent stigning per år som blir 328.266 arbeids-plasser i 2013

Tabell 2.3: Sysselsetting: 2003 (kilde: WTTCs rapport om Norge fra 2002)

2b) Antall jobber 2003 i RT-økonomi tilsvarer 13,4 prosent av samlet sysselsetting

ANP ser viktigheten av å stimulere mindre bedrifter, som ofte får problemer på grunn av globaliseringen, i kampen mot de største selskapene. De store selskapene har et større potensial til å rasjonalisere, og slik som tallene fra WTTC viser så vil sysselsettingen i Norge ikke stige proporsjonalt med omsetningen i næringen (0,8 prosent sysselsettingsvekst mot 2,8 prosent omsetningsvekst).

Sett ut fra et ønske om å bevare lokalkulturer og lokal syssellsetting i distriktene er det av ytterste viktighet at disse får den hjelp de trenger for å kunne sikre sine arbeidsplasser.

Autentisitet
Innen reisenæringen er dette et område som det har vært forsket på i mange tiår, og dette står svært sentralt i Authentic Norway Project.

For å fremme autentisitet i min løsning måtte løsningen ha en struktur som kunne fremme det unike hos hver reisedestinasjon.

Autentisk reising som e-handel
Det er viktig å få en samlet enhetlig tankegang bak deltagerne i prosjektet, fordi innen salg, markedsføring og CRM er det viktig å fremme samme budskap i alle kanaler[13, Strøm, 2002] slik at kundene ikke er i villrede om hva slags konsept dette er. Derfor mente jeg en integrert e-handelsløsning mellom de potensielt deltagende bedriftene var et godt utgangspunkt.

Ut fra dette bør man sette opp en oversikt over hvilke temaer som e-

handelsløsningen skal inneholde, med avgrensinger og krav deltagende bedrifter skulle ta hensyn til. Reisedestinasjonene skal oppnå større fleksibilitet enn de har i dag, men de må likevel leve opp til en policy som skal styrke helheten av merkenavnet 'Authentic Norway Project", og en viktig oppgave var å samle de deltagende hotellene i en samkjørt enhet.

"Målet med Authentic Norway Project er å designe det som er nødvendig for å skape en ny forretningsmodell kalt "Verdibasert franchise", som kan lede autentisk turisme inn i moderne forretningsmetoder".[29, Karabeg og Akkok mfl., 2002]. Begrepet "autentisk reising" (eng: authentic travel) er et uttrykk som i dag brukes av flere turistdestinasjoner. OpenTravel, som har Internettadresse www.opentravel.org, er en samling bedrifter og selskaper som har medlemmer knyttet til autentisk reising.

En viktig del av de deltagende bedriftenes næringsgrunnlag er firmaer der de jobber med "team building" og å forsterke samholdet i bedriften, og et lignende prosjekt som er fullført på IFI hadde de deltagende bedriftene følgende fire definerte fokusområder forkortet til MICE: Meetings, Incentives, Conferences og Events.
Man kan derfor anta at en Internettapplikasjon for disse bedriftene skal inneholde mye informasjon om disse 4 temaene, og sannsynligvis også på sikt, ha muligheter for å reservere deltagelse til dette over Internett.

2.4 Customer Relationship Management

CRM systemer inkluderer et stort spekter av forretningskonsepter og arbeidsmåter for å organisere og skaffe kunder. Et CRM-system kan i sitt ytterpunkt være gule "post-it" lapper på en vegg, eller det kan være elektroniske enheter og systemer. Å skrive om CRM generelt og å prøve å forklare hva CRM er og ikke er, er en altfor stor oppgave fordi CRM faktisk alltid har eksistert, helt tilbake til steinalderen da folk handlet og byttet mat og våpen. I de ti siste årene, med veksten Internett har ordet CRM blitt tett relatert til programvaresystemer.

Å kutte priser er en måte å skaffe kunder, men i det lange løp vil denne måten å gjøre forretninger på ekskludere et stort antall selskaper fordi de ikke kan konkurrere med de store multinasjonale selskapene som selger enorme mengder av produktene med svært lav fortjeneste per produkt. Noen forretninger må finne andre måter å konkurrere på, og å

fokusere på “hva kundene virkelig vil ha” er en måte å gjøre forretninger på. For å gjøre dette grundig og effektivt må man vite[1, Puleo, 2003]:

1) Hvem er dine kunder?
2) Hvor er de?
3) Hvor trenger de at du befinner deg?
4) Hva trenger de?

Hvis man vet svarene på disse spørsmålene har man et bedre utgangspunkt for å få og beholde kundene uten å måtte basere seg kun på å kutte priser. Her er det viktig at man har klare intensjoner om hvilke kundegrupper man skal nå ut til, og firmaer og bedrifter som arbeidet med "team building" og arbeidsmiljø-fremmende tiltak er en viktig kundegruppe for aktørene innen autentisk turisme. Grunnen til dette er at disse reisedestinasjonene har omgivelser som er godt egnet for å drive målrettet arbeid uten mange forstyrrelser og store menneskemasser.

Det var viktig at arkitekturen i prototypene jeg lagde skulle ha mulighet til å bygges ut til et system med flere typiske CRM funksjoner. Å holde kommunikasjonen gående med kunden, ofte kalt “En lang samtale”, gir ofte resultater. Uttrykket “en lang samtale” refererer til dialogen mellom en selger og en kunde der selger lytter til og tilpasser seg til hva kunden trenger, og dette er velkjent i alle næringer, også i reiselivsnæringen. Dette skaper lojalitet fra kunden. For å få til dette mente jeg at den endelige løsningen måtte kunne ha funksjonalitet for markedsautomasjon, da dette kan forenkle en del av kommunikasjonen med kundene, for eksempel automatisk generering av e-postutsendelser, introduksjonsbrev, SMS og fakturaer eller e-fakturaer.

Å lage en Internettapplikasjon der flere selskaper deler samme løsning vil forenkle arbeidet med å skape et helhetlig inntrykk ut mot kundene, og dette kan føre til mersalg for alle de deltagende bedriftene. Dersom en kunde er fornøyd med sitt opphold på èn reisedestinasjon, vil hun eller han lettere velge en slik reise igjen innen fra samme Internettsider hvis selskapene framstår som en sterk enhet.

2.4.1 CRM og datasystemer

CRM-systemer er laget for å bygge sterkere relasjoner til kundene, og selskaper og bedrifter har de 2-3 årene vist en stadig økende interesse

for systemer som skal hjelpe de med å skaffe og ikke minst beholde kundene.

Hvilket konsept og hvilken teknologi som passer best må bestemmes fra bedrift til bedrift, men trenden det siste året (2002) har vært at de store leverandørene har redusert kompleksiteten i deres systemer. For å tydeliggjøre kompleksitetsproblematikken skrev Bob Thompson fra CRM-Guru om CRM-systemene og deres brukere: "Ville du gitt dine Ferrari bilnøkler til en person uten førerkort?" Med dette mener han at i dag så styres CRM-systemer med ekstremt potensial og kompleksitet av personer uten nok erfaring, og på grunn av mangel på forståelse for hva disse systemene gjør og kan gjøre, så gjør man feil antagelser, analyser og prognoser.

Jay Chang [8, Chang, 2002] fastslår "CRM-produkter har tradisjonelt blitt assosiert med front-office funksjoner som markedsføring, salg og kundeservice." I dag har largescale data mining og analyser blitt en del av disse systemene. For eksempel har Internettsideovervåkning resultert i store datamengder som kan brukes analytisk, nom som resulterer i større segmentering og "targeting capabilities". Slike overvåkningssystemer for Internettsider kan brukes til å holde rede på nesten all trafikk and alle aktiviteter på en Internettside, for eksempel på hvilke tidspunkter det er mest trafikk på sidene, hvilke sider som er mest besøkt, på hvilken måte brukerne manøvrerer gjennom sidene, og på hvilken side brukerne forlater siden.

2.4.2 Markedsautomasjon

Markedsautomasjon er viktig i CRM. Markedsautomasjonssystemer, det vil si IT-systemer, hjelper selskaper å automatisk kommunisere riktig melding til rett tid til rett kunde i alle kontaktpunkter, og dette prinsippet er også viktig for bedriftene i ANP. Det er likevel viktig å være klar over at automatisk markedsføring kan være mer vanskelig enn det høres ut. "Gartner Group" ga i 2002 ut en studierapport som konkluderte med at "mer enn 50 prosent av alle CRM implementasjoner var sett på som mislykket sett fra kundens side."[8, Chang, 2002]. Jeg tror det er vanskelig å få til å helautomatisere markedsføring for de fleste bransjer. Uansett er CRM-leverandørenes forventninger til markedsautomasjon høye. En ting er i allefall sikkert: full markedsautomasjon er en mye mer umoden teknologi enn tradisjonelle CRM aktiviteter som for

eksempel kundesentre. I dag har svært få bedrifter og organisasjoner implementert full markedsautomasjon, men systemleverandører spør at innen 3 år vil dette være mer vanlig.

Internettportaler har blitt svært viktig del av CRM-systemene, og en viktig grunn til dette er at man kan lage systemer uten å måtte kjøpe dyr maskinvare og oppgradere infrastrukturen, det vil si oppgradere klientmaskinene, til de ansatte. For eksempel, hvis man bruker en ASP (Application Service Provider) løsning trenger ikke selskapet å oppgradere mer enn at nettleseren på klientmaskinen støtter teknologien som ASP-løsningen krever (for eksempel Shockwave og Flash plug-ins).

I dag finnes det et stort antall CRM portaler som kan kjøpes fra programvareleverandører. Internettportalen CRMBuyer.com skriver at en av de mest viktige kriteriene for å velge CRM portal er “back end” integrasjonen. Dette betyr at Internettportalen må kommunisere med eksisterende systemer som “ligger i bunn” av selskapets forretningssystem, deriblant "legacy" systemer., og dette var spesielt viktig i min oppgave der jeg så på underliggende arkitekturer.

AMR Research skrev om 15 leverandører av Internettportaler som seriøst bør vurderes/evalueres hvis man vurderer en Internett, ekstranett eller intranett CRM-portal. Disse selskapene er: Vignette, Plumtree, Epicentric, SAP Portals, Oracle, PeopleSoft, BEA, BroadVision, Corechange, Sybase, Tibco, Art Technology Group, Computer Associates, Sun Microsystems og IBM. Hvis man bruker Microsoft .Net plattform bør man også evaluere Microsofts SharePoint Portal Server siden denne tjeneren er designet for .NET plattform. AMR Research konkluderer med at alt i alt er Plumtree, Epicentric, Oracle og BEA selskapene som er best egnet til å tjene som “Enterprise” portaler, men de skriver også at Vignette, Episentric, ATG, BEA og Sybase har “den mest tekniske fleksible arkitekturen.[22, crmbuyer.com, 2003]. Jeg mener dette er systemer man kan undersøke nærmere for å lære av.

2.5 Markedsføring

Tradisjonelt i markedsføring snakker man om de fire P-ene: Positioning, Promotion, Pricing og Placement. Når det gjelder handel på Internett så ser situasjonen ut til å være mer kompleks, fordi å fokusere på disse aspektene ofte ikke har ført til noe bedre salg. La oss først legge merke

til at e-business markedet nådde USD 240 milliarder i 2002. Studier har vist at e-forhandlere taper USD 6.3 milliarder i tapte salg, og dette utgjør 13 prosent av det totale salget[3, Pollock, 2002].

For eksempel "75 prosent av mottakerne forlater handlevogna uten å fullføre handelen i følge Bizrate og 80 prosent ifølge ATKearny".[3, Pollock, 2002]. Dette betyr at kundene ikke fullfører transaksjonen, noe som betyr at de ikke fullfører kjøpet. Slik er ikke situasjonen i vanlige butikker. Det er tydelig at det er aspekter rundt Internetthandel som er annerledes enn vanlige butikker, og det er viktige å finne ut hvordan man kan løse noen av de problemene som oppstår. I vanlige butikker har man gjerne reist en stund for å komme dit, parkert bilen og lignende, og dersom man skal finne en annen butikk betyr det å finne ny parkeringsplass, og reise kanskje 10 minutter til. Internettkunder kan forlate butikken og finne en annen med kun et par tasteklikk. En annen grunn er at folk er mer utålmodige, ting skal gå fortere og fortere, og tiden det tar å fullføre skjemaene som skal utfylles før man kan kjøpe kan være en grunn til at mange ikke fullfører handelen.

De vanligste grunnene for ufullstendige transaksjoner/kjøp er: [3, Pollock, 2002]

1. Lange eller forvirrende utsjekkingsprosesser (Bizrate skriver 43 prosent og ATKearny 27 prosent).
2. Dårlig design ("Creative Design" anslår at 43 prosent av alle startede salg ikke fullføres på grunn av dette.)
3. Mangelfull investering i e-business løsningen. Omtrent 6 ganger så mye som det faktisk investeres er det som trengs for at løsningen skal bli bra nok.
4. Dårlig båndbredde

Ut i fra punkt 1, 2 og 3 i lista over kan man se at ANPs arkitektur ikke måtte bli så kompleks at navigeringen gjennom sidene og designet ikke kunne realiseres slik man ønsket, og dette ble vektlagt i mine prototyper.

Kapittel 3

Om arkitekturer og prototypene

3.1 Overordnet om arkitekturer

Å velge riktig arkitektur i Internettapplikasjoner er svært viktig.
Arkitekturen i løsningen er fundamentet for alt det skal bygges på, og det har skjedd svært mange forandringer innen Internettspråkene de siste par årene, noe som gjør at dette er et uoversiktlig område innen programmering.

Valg av teknologi som operativsystem og programmeringsspråk vil kunne få store konsekvenser, og feil teknologivalg kan gi store begrensinger på hva som er mulig å utvikle til en overkommelig pris.

Når man skal lage et system som dette, er det mange teknologier og arkitekturer man kan velge mellom. I denne oppgaven ønsket jeg å se på tre arkitekturer som jeg trodde kunne være interessante og velegnede til å utvikle fleksible Internettapplikasjoner i, for å se på ulemper og fordeler med disse.

Når jeg evaluerte arkitekturene i denne oppgaven delte jeg opp i to hovedkategorier som jeg evaluerte i hver av prototypene:

1. Databasevalg for lagring av data.

2. Valg av programmeringsspråk i Internettapplikasjonen.

3.1.1 Databasevalg

Her brukte jeg to forskjellige databaser til å lagre dataene i:

1. Relasjonsdatabasen MySQL
2. Tamino XML Server

MySQL er en svært mye brukt database som er OpenSource. Databasen er lett å administrere og krever lite maskinressurser. Valget av denne databasen var således et realistisk valg for en e-handels løsning som dette.

Som XML-database brukte jeg Tamino XML Server. Denne vil bli nærmere redegjort for i kapittel 5 (XML Native Database).

3.1.2 Valg av programmeringsspråk på Internettsidene

Til prototypen kunne jeg velge mellom språk som PHP, Java Applets, Java Servlets, JSP(Java Server Pages), ASP (Microsoft Active Server Pages), C Sharp (Microsoft .NET) blant flere. Det finnes ingen fasitsvar på hva som er det beste språket, fordi de har forskjellige egenskaper, både i muligheter og kompleksitet. For eksempel med JSP og Java Servlets programmerer man i Java, og dette gir muligheter innen objektorientering som er svært gode. C Sharp fra Microsoft, som er en del av .NET, er også objektorientert.

PHP lar programmereren velge mellom å bruke objekter eller datastrukturer uten objekter, man kan derfor velge hva som er best egnet.

Jeg valgte PHP på alle mine tre prototyper fordi språket godt utbygget og fordi det på flere områder er et enklere språk å bruke enn JSP. Jeg siterer fra boka "Developing Web Sites with PHP: "PHP gjør arbeidet med variabler ekstremt enkelt".[23, Greenspan og Bulger, 2000]. I tillegg er PHP svært mye brukt, og det er lett å få hjelp på Internett, siden dette er et Open Source programmeringsspråk.

3.2 Implementerte funksjoner

I denne oppgaven tok jeg utgangspunkt i at det i dag finnes mange bedrifter innen reiseliv i Norge som er små , ofte med mindre en ti ansatte.

	Prototyp 1	**Prototyp 2**	**Prototyp 3**
Operativsystem	Windows 2000 Server	Windows 2000 Server	Windows 2000 Server
Database	MySQL	Tamino XML Server	MySQL
Programvare	PHP installation	PHP installation, TomCat, Java SDK1.4	PHP installation, AXIS, Java SDK1.4, PHP PEAR SOAP Client
Språk	PHP, HTML, SQL XSD	PHP, HTML, XML, XSL, SQL	PHP, HTML, Java, XML,
Web Server	IIS	Apache på Web Services server og IIS på Web Services klient	IIS

Tabell 3.1: Programvare og språk brukt i prototypene

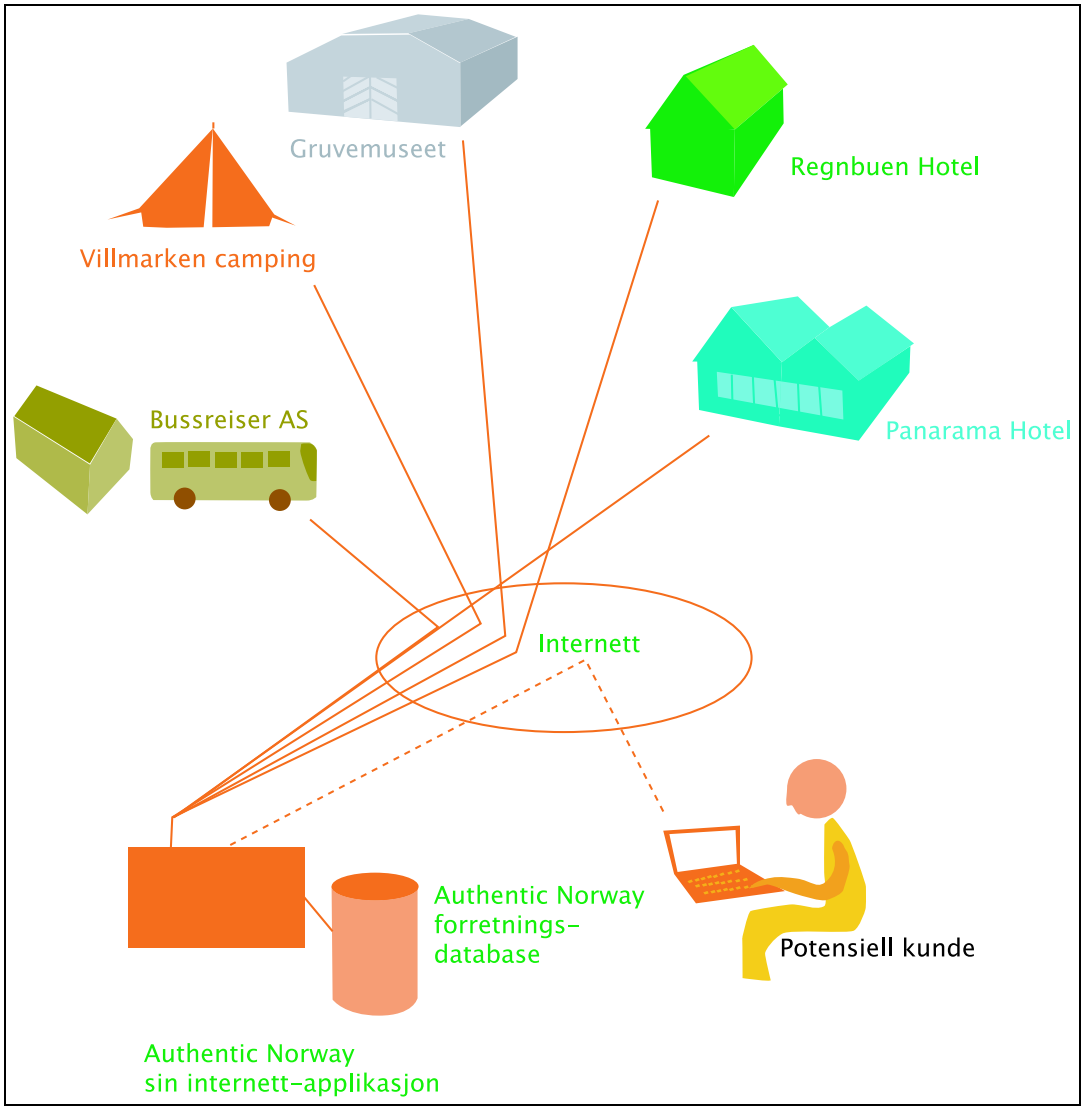
Måten de arbeider på er ofte 'gammeldags' i den forstand at de har investert lite penger og ressurser til utvikling av IT løsninger. Mens de store selskapene som konkurrerer i samme marked har avanserte løsninger med sentralbord, markedsautomatisering, og avanserte bookingløsninger på Internett, har de mindre bedriftene ikke disse løsningene. Dette resulterer i at de store bedriftene tar mer og mer av markedet av tilreisende turister, og dette er en eksistensiell trussel for de små bedriftene.

En reisedestinasjon har mange aktivitetsmuligheter, men en kunde som sitter i utlandet ønsker ikke alltid å ringe til de forskjellige aktivitetsarrangørene eller å sende mange e-post for å finne ut av dette.

Dersom man kunne få til en Internettløsning, kunne en potensiell kunde i Canada, USA, Japan eller Tyskland finne all den informasjonen hun eller han ønsket, og dermed velge en slik reise i stedet for en tur arrangert av de store reisearrangørene.

Jeg ønsket å lage prototyper som skulle understøtte disse kravene for kunden:

1. Løsningen skulle være effektiv og brukervennlig, slik at kunden slapp å manøvrere mye rundt på Internettsiden.



Figur 3.1: Slik kan en reisedestinasjon se ut. De heltrukne linjene betyr at selskapene er deltagere i Authentic Norway Project

2. Det skulle være enkelt å legge inn en bestilling/påmelding.

3. Kunden skulle til enhver tid under reservasjonen se hva hun eller han hadde valgt ved å vise en dynamisk timeplan. En dynamisk timeplan kan sammenlignes med en handlekurv som brukes i Internettbutikker
   Det var viktig at det ikke skulle være vanskelig å melde seg på aktiviteter og overnatting på reisedestinasjonen. Det skulle også være mye velge mellom, og kunden skulle ha god oversikt over sin egen bestilling og timeplan.

4. Ha høy sikkerhet for personopplysninger, salgsopplysninger og betalingsopplysninger. Jeg implementerte ikke autentisering på Internettsidene utover brukere til databasen, men det var et krav at prototypen skulle kunne implementeres med høy sikkerhet.

5. I use case diagrammet i figur 3.2 har jeg tegnet den funksjonaliteten jeg ville implementere i prototypene.

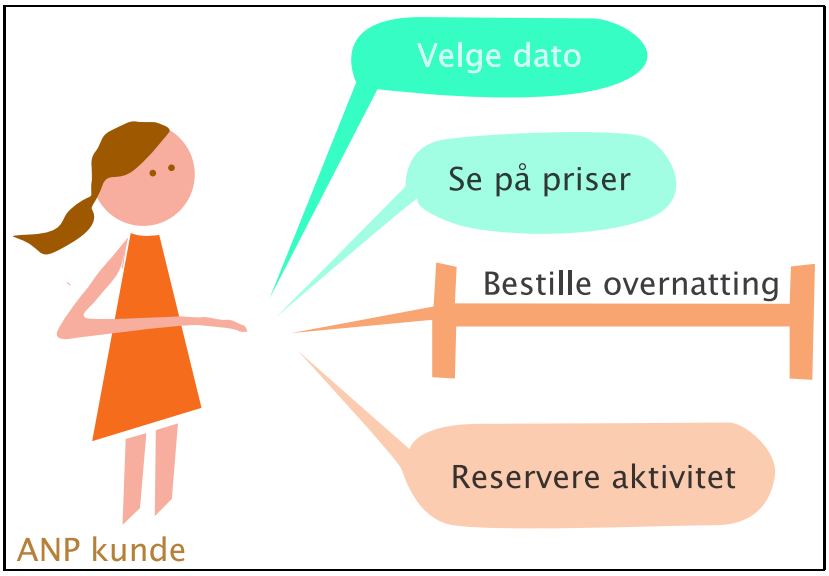
6. Funksjonaliteten i prototypene skulle være at kunden som brukte applikasjonen oppga en dato for oppholdet, og han eller hun laget en timeplan for perioden der man haket av for aktiviteter og overnatting. Dette ble laget i "Wizard" stil, og man gikk gjennom 2 ”Wizards”, èn for overnatting og en for aktiviteter, for å reservere.

I tillegg ville jeg lage en applikasjon som skulle ha eller kunne enkelt bygges ut til å:

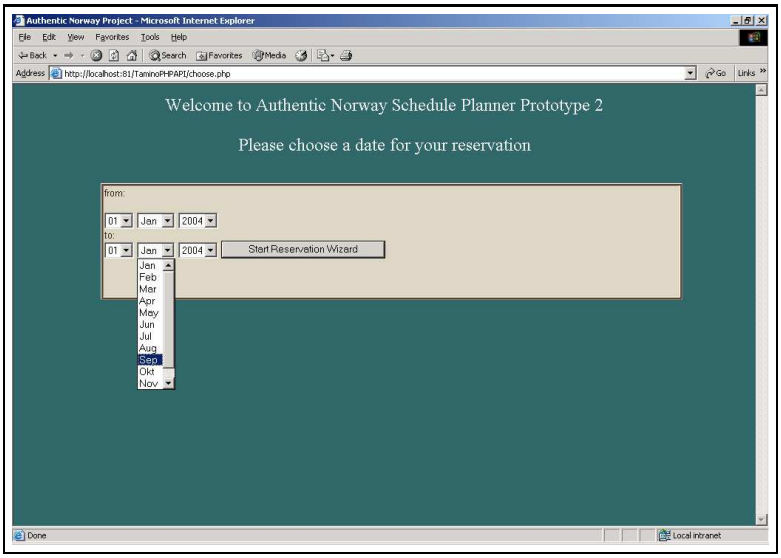
1. Ha høy sikkerhet

2. Være enkel å vedlikeholde

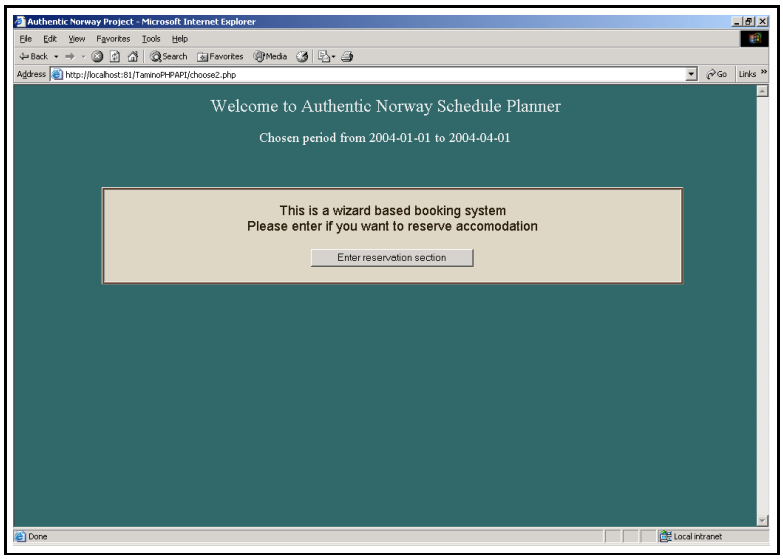
3. Være pålitelig, for eksempel ved at dobbeltreservasjon av rom ikke skal kunne forekomme.



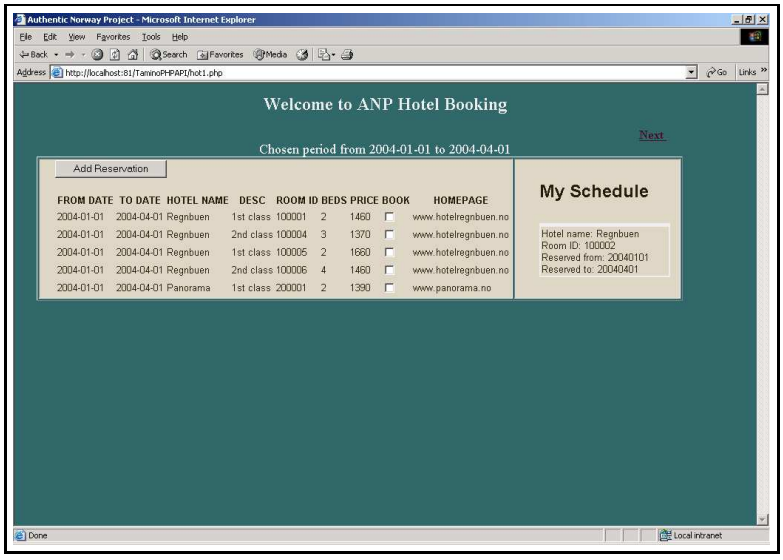
Figur 3.2: Jeg implementerte disse funksjonene.



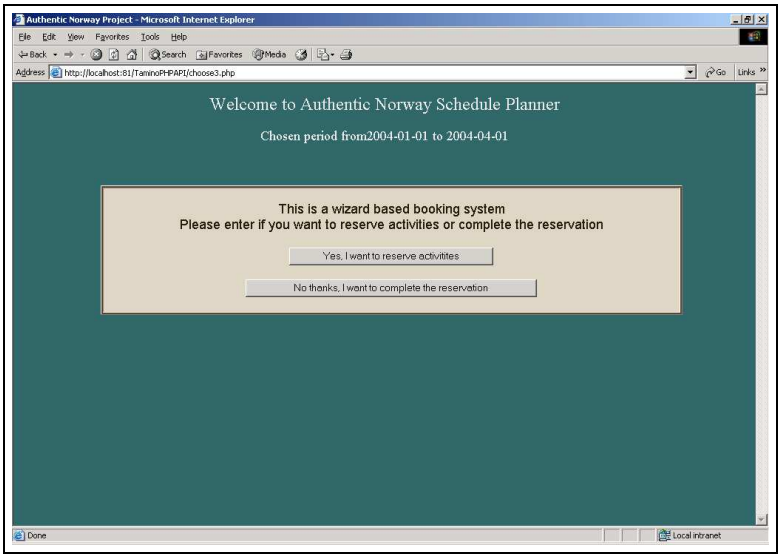
Figur 3.3: Dette er det første skjermbildet i applikasjonen, og i dette velger kunden start og sluttdato for reisen. Skjermbildene er tatt fra prototyp 2.



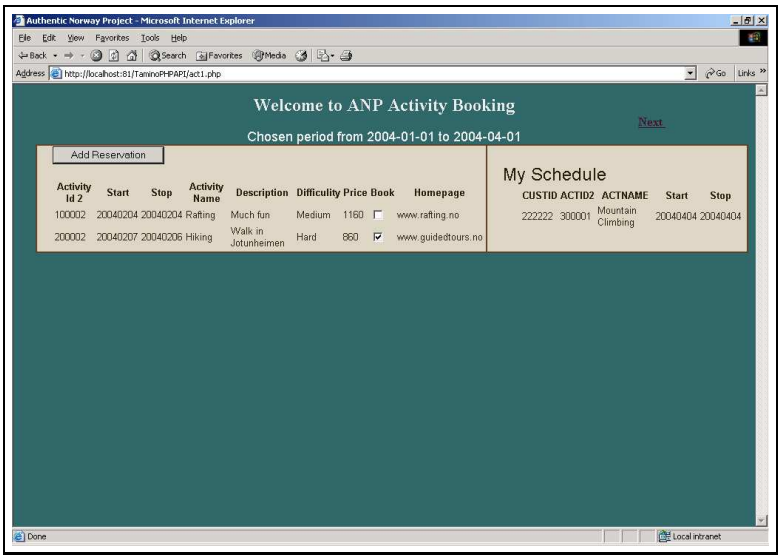
Figur 3.4: Skjermbilde 2 inneholder informasjon om reservasjonsprosessen.



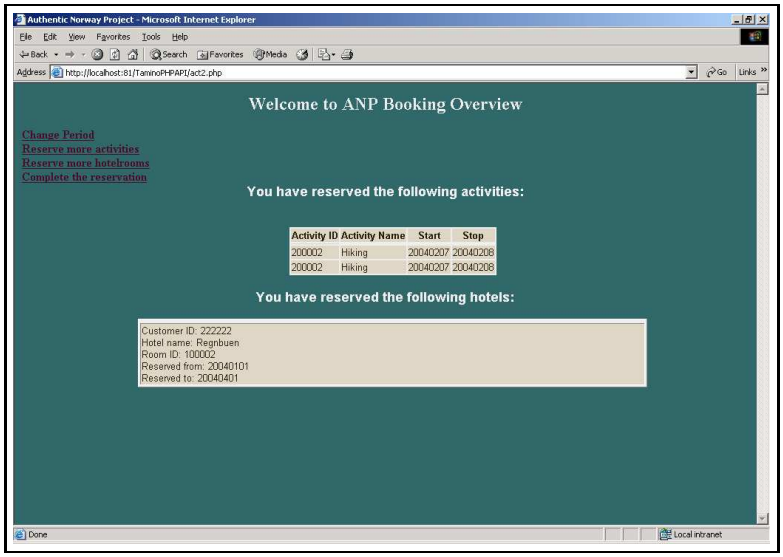
Figur 3.5: Dette er det tredje skjermbildet i applikasjonen. Til venstre viste jeg en liste med hotellrom som var ledige i den oppgitte perioden. Til høyre viste jeg en liste med det eller de hotellrom som kunden hadde reservert.



Figur 3.6: Skjermbilde 4 lar kunden velge mellom å reservere aktiviteter først eller å fullføre kjøpet med en gang.



Figur 3.7: I det femte skjermbildet kunne kunden reservere aktiviteter. Alle tilgjengelige aktiviteter var oppgitt til venstre i skjermbildet, og på høyre side ble det vist de som kunden hadde meldt seg på.



Figur 3.8: Etter at kunden hadde satt opp sin ønskede timeplan med overnatting og aktiviteter kom denne kvitteringssiden opp i skjermbilde 6. Her kunne kunden gå tilbake for å bestille flere overnattinger eller aktiviteter, eller fullføre kjøpet. Dersom de fullførte kjøpet skulle det komme opp betalingsinformasjon, samt for eksempel mulighet til å betale med VISA, men jeg implementerte ikke betalingsløsning.

Kapittel 4

Prototype 1 - MySQL og PHP

4.1 Bakgrunn om teknologien

Denne prototypen var en sentralisert løsning med relasjonsdatabasen MySQL og en PHP Internettapplikasjon. Databasen og applikasjonen lå på samme server. Slike løsninger er ofte de raskeste å implementere, og de er veldokumenterte og godt utprøvd, og dersom det er tilstrekkelig med en løsning som er ikke-distribuert, velger man gjerne å bruke èn database.

4.1.1 MySQL

Bruken av relasjonsdatabaser er over 30 år gammel. 1970 ga Codd ut sin artikkel om relasjonsdatabaser, og fra denne artikkelen kom ut, regner man relasjonsdatabasens alder. Relasjonsdatabaser i ikke-distribuerte miljøer er svært godt dokumentert, og derfor beskriver jeg de ikke ytterligere på et generelt grunnlag i denne oppgaven. Det er likevel viktig å se på MySQL sine egenskaper som relasjonsdatabase, samt hva denne tilbyr av funksjonalitet.

MySQL er svært populær som database på Internettløsninger for små til mellomstore bedrifter. Databasen er svært rask, og den er i tillegg svært rimelig i innkjøp, da den kun koster 3000 NOK dersom man skal bruke den kommersielt.

Det var likevel verdt å merke seg at denne databasen, i allefall fram til nå, ikke er like godt utbygget som for eksempel Oracle 9i og MS SQL Server. MySQL hadde ikke støtte for nøstede SQL setninger, og den tilbød hell-

er ikke fremmednøkler eller lagrede prosedyrer. Fram til nå har MySQL kommet til versjon 4, og versjon 4.1 i betaversjon, og denne betaversjonen støtter nøstede SQL-spørringer. Ellers oppfyller databasen alle krav til å være en database som ad-hoc spørringer, autentisering og backup.

Når man skal bruke MySQL versjon 4.0 er det viktig at man er klar over at nøstede SQL-setninger må skrives om, og dette kan gi en del mer kompliserte SQL-setninger, og blant annet må SQL-spørringer bruke DDL språk(Data Definition Language) til å lage temporære tabeller som brukes i stedet for nøstede spørringer. Til forsvar for denne mangelen skal sies at det er lagt mye vekt på å dokumentere og gi eksempler på hvordan disse SQL-spørringene skal skrives.

4.1.2 PHP

PHP er et scripting språk som er svært populært. Språkets bruk av variabler er enkelt, og språket har en del smarte og unike muligheter ved enkel bruk av variabler, for eksempel assosiative arrays, som jeg bruker i prototyp 3, og dette forenklet programmeringen svært mye.

I tillegg er det svært mange fora og nyhetsgrupper på Internett der man kan stille spørsmål om PHP, og dette bekrefter at PHP er et programmeringsspråk som er svært utbredt og anvendelig.

Hva gjorde jeg:
Jeg ville lage en standardisert enkel prototyp med 7 skjermbilder som skulle være grunnlaget for sammenligningen av de tre arkitekturene. Internettapplikasjonen var laget på en lite kompleks måte, og løsningen hadde lite visuelt design utover den logiske strukturen. Jeg laget ikke administrasjonsgrensesnitt for løsningen og heller ikke autentiseringsløsning. Grunnen til dette var at å implementere dette på alle prototypene ville bli for omfattende. Poengene mine i sammenligningene av arkitekturene kom fram uten disse delene av løsningen.

4.2 Presentasjon av løsning

Løsningen besto kun av .php filer og èn database. I løsningen brukte jeg 8 php-filer, og kompleksiteten var mellom lav og middels, og databasemodellen besto av 7 tabeller.

PHP og MySQL blir ofte brukt sammen, og i løsningen brukte jeg funksjoner i PHP som er laget spesifikt mot MySQL, og disse virket uten noen uforutsette problemer. Dataene ble lagret i temporære tabeller fram til kunden fullførte salget ved å fullføre bestillingen eller forlot nettstedet uten å fullføre kjøpet.

4.3 Testresultater

4.3.1 Installasjon

Jeg trengte kun å installere PHP og MySQL, og installasjonen av denne programvaren var både enkel og rask. Programmene som skulle installeres var begge små og lite ressurskrevende på maskinvare.

4.3.2 Implementasjon

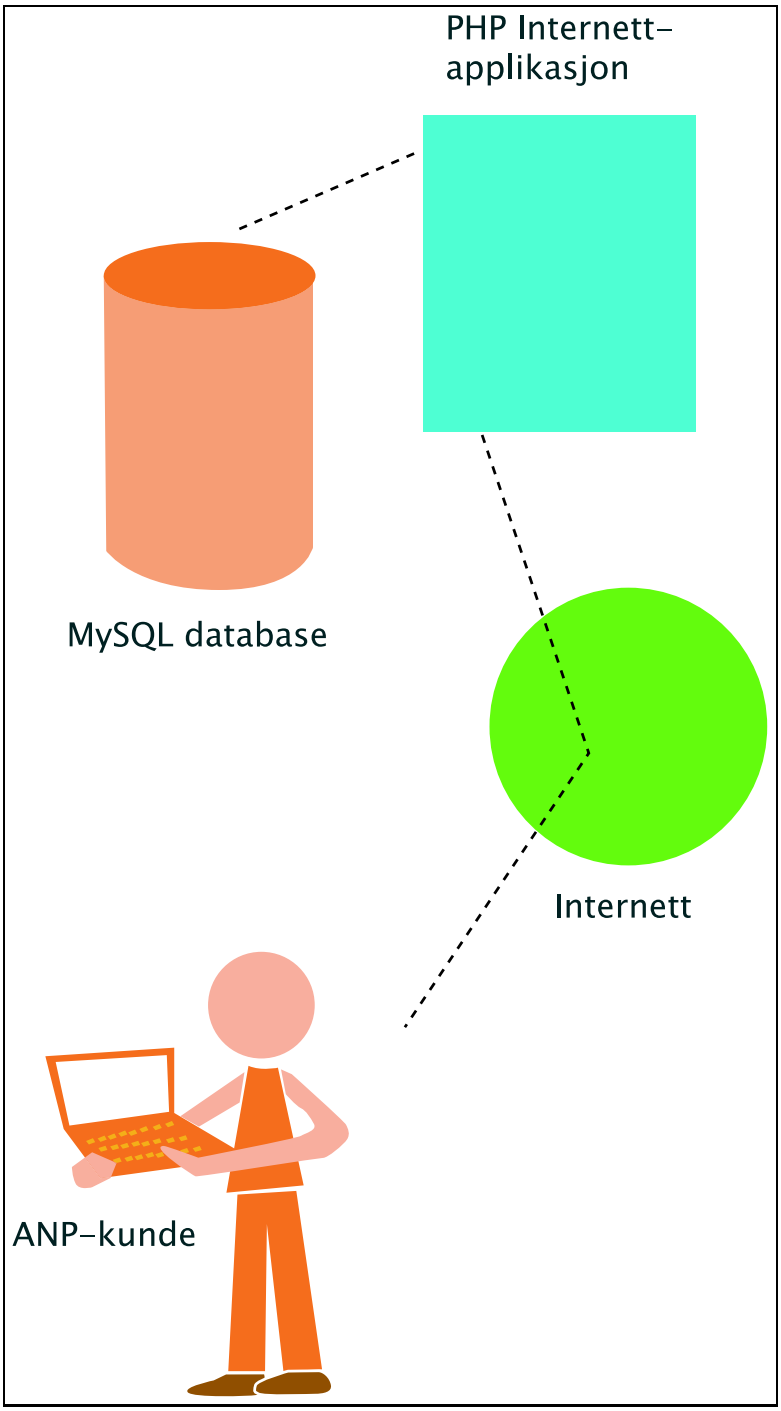
Under implementasjonen oppsto det ingen uforutsette problemer, men noe av arbeidet med databasen ble vanskeligere fordi MySQL ikke støttet nøstede SQL-setninger.

Jeg gjorde en forenkling i databasemodellen ved at jeg lagret hotellnavn i to tabeller. Grunnen var at jeg hadde problemer med å lage en select med uttrykket ”not exists” der jeg skulle hente data fra 3 tabeller. I versjon 4.1 er det støtte for nøstede selects, og da blir det lettere å lage denne spørringen. Spørringen virket som den skulle når hotellnavn både lå i tabellen ”hotels” og ”hotelroom”.

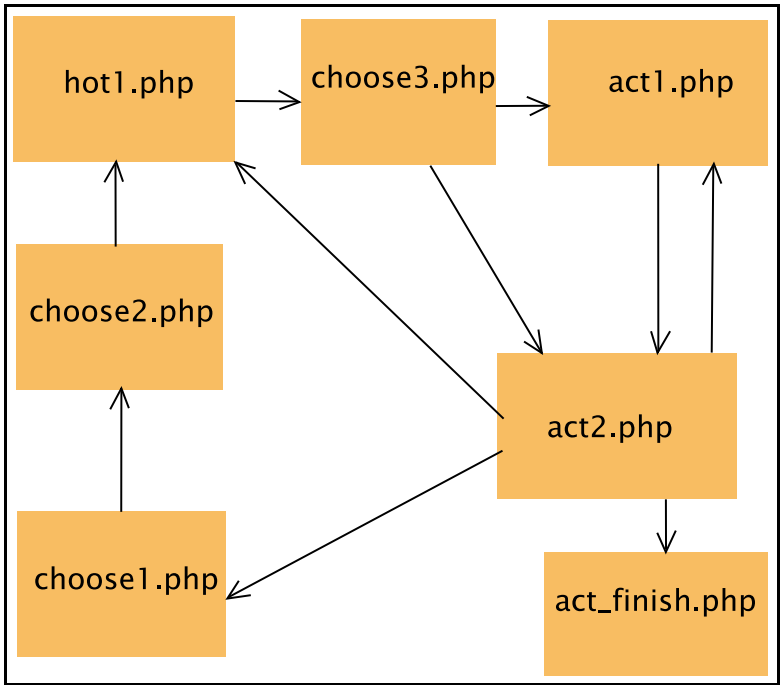
Jeg kunne ha brukt mer tid på å prøve å løse dette, men fordi MySQL 4.1 snart kom i standard versjon, ville det virke med "not exists" i denne.

Å bruke PHP på Windows plattform, spesielt under IIS Web Server kan være problematisk fordi PHP ikke ble laget for dette operativsystemet og denne Internettjeneren. Det oppsto imidlertid ikke problemer med dette under implementasjonen.

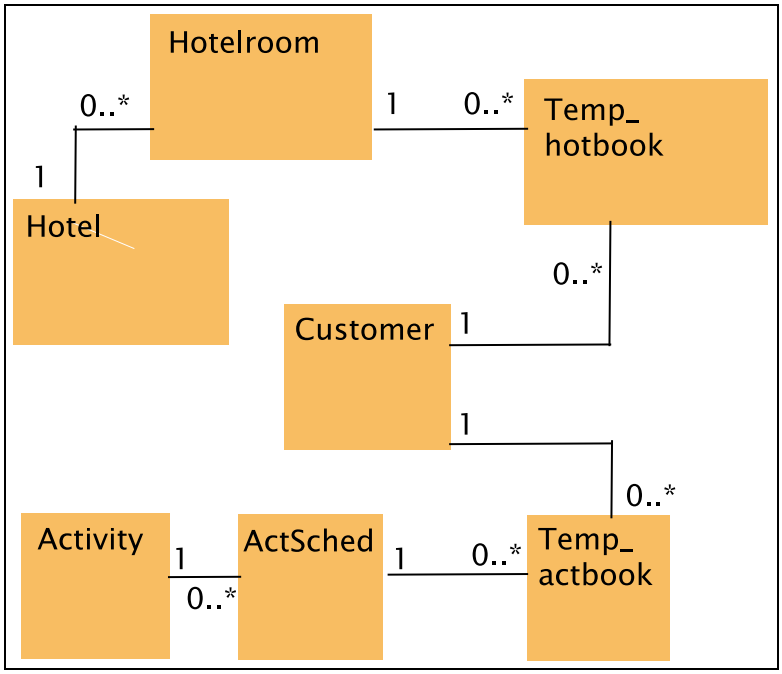
Denne løsningen hadde kortest implementasjonstid av de tre fordi løsningen var basert på en standard tankegang som man lærer i databasekurs. MySQL APIen til PHP var meget fullstendig og godt dokumentert.



Figur 4.1: Prototyp 1 hadde en standard kombinasjon av relasjonsdatabase og applikasjon.



Figur 4.2: Rekkefølgen på skjermbildene.



Figur 4.3: ER-modell av databasen i prototype 1. Temp_hotbook og Temp_actbook inneholdt temporære reservasjoner brukt under reservasjonssesjonen. ActSched(forkortelse for Activity Schedule) inneholdt start- og stopptidspunkter for de enkelte aktiviteter.

Kapittel 5

Prototype 2 - XML Native database

Prototyp 2 var en sentralisert løsning der jeg brukte en XML database og PHP. XML databaser har på en del områder høyere fleksibilitet enn tradisjonelle relasjonsdatabaser.

5.1 Bakgrunn om XML

Da XML kom på slutten av nittitallet ble det sett på som en stor nyskapning. Man hadde tidligere hatt andre markup-språk (SGML, HTML), men XML gjorde det svært mye lettere å strukturere data enn SGML, og HTML er ikke ment for å strukturere data slik XML er.

I mange sammenhenger er XML en ypperlig måte å holde rede på data på, og XML har gjort integrasjonsarbeid med overføring av data mellom applikasjoner lettere enn det var før XML eksisterte. XML brukes i dag ofte til kommunikasjon mellom applikasjoner. For eksempel brukes XML i Windows 2003 Server til overføring av data mellom komponenter i operativsystemet. XML har også forenklet integrasjonen mellom CORBA og DCOM med SOAP (Simple Object Access Protocol) som er XML basert.

Andre fordeler med XML er:

1. XML har innebygd support for internasjonalisering fordi den bruker unicode.
2. XML er plattformuavhengig.
3. XML er tekstbasert, og "human readable".

4. XML er selvbeskrivende.

5. XML er "utvidbart" (eng:extensible) fordi man kan legge til ekstra informasjon i et format uten å ødelegge for applikasjonen som trengte det gamle formatet av dataene.

6. Det finnes mange verktøy og programmeringsspråk som har funksjonalitet får å prosessere XML dokumenter, for eksempel PHP.

7. XML er egnet for å lagre tre-strukturer og graf-strukturer.

XML brukes i svært mange utbredte teknologier, og fordi XML er en standard og ikke et programmeringsspråk, kan det brukes overalt der applikasjoner kan prosessere XML. Blant annet er det brukt i SVG grafikk.

5.2 Bakgrunn om XML databaser

XML databaser gir mulighet får å lagre alle XML-baserte data i en database. De vanligste brukene av XML databaser er til å lagre semi-strukturerte data, dokument-sentriske data og som basis i "Content management" systemer.

- Dokument-sentriske dokumenter er: Dette er dokumenter som ofte er laget for å forstås av mennesker, og de har en uforutsigbar struktur. Dokument-sentriske data kan være bøker, kvitteringer, reklamer og bestillingsordrer.

- Semi-strukturerte data er: Dette er data som ikke kun består av tekst, og som har et innhold som er uregelmessig, og der forskjellen mellom skjema og data er uklar, samt at skjemaet endrer seg ofte. Å lage tabeller for alle de ulike kombinasjonene for semi-strukturerte data i en relasjonsdatabase vil bli svært mange tabeller med svært mange nullverdier.

- Content Management systemer er: Systemer som er ment for dokumenter skrevet av mennesker, der systemet gir spesiell funksjonalitet for å behandle innholdet i dokumentet. Content Management har et eget underkapittel.

Dataene som disse begrepene omfatter deler egenskapen at de er vanskelige å modellere i en relasjonsdatabase.

For å avgjøre hvilken type database man skal velge bør man finne ut om man skal lagre data-sentriske eller dokument-sentriske dokumenter. "Dette er en av de viktigste indikasjonene på om man bør velge en relasjonsdatabase eller en XML database."[24, Farges, 2002]. Dersom man kun skal lagre data som er data-sentriske bruker man tradisjonelt relasjonsdatabaser, fordi også disse har funksjonalitet for eksport til XML. Derfor er det i seg selv ikke nødvendig å velge en XML database for å eksportere XML data.

"XML Native" databasenes store fordel ligger i å kunne motta dokument-sentriske data eller semi-strukturerte data der strukturen til dataene er ukjente eller endres over tid. Dersom man har data i disse kategoriene (dokument-sentriske, semi-strukturerte eller Content Management system), så kan og bør man vurdere en XML database. Hvis man for eksempel skal gjenbruke et dokument i flere formater, for eksempel ved hjelp av XSL-transformasjoner kan en slik database være godt egnet fordi man vil lagre XSL-filer i databasen. Grunnen til dette er at XML filene, ofte XSL filer, ikke har en fast struktur fordi elementer skal kunne legges til og fjernes fortløpende, og databasen må kunne takle dette. Dette minner om mulighetene som finnes i Content Management systemer, der man mottar datafiler med ukjent struktur.

En slik måte å jobbe på passer ikke i en relasjonsdatabase der alle kombinasjoner i så fall måtte modelleres i databasen i forkant eller i runtime, eller lagres som en BLOBs(Binary Large Object). BLOBs gir ikke samme fleksibilitet og muligheter, og XSL-transformasjoner er sentralt i prototyp 2.

5.2.1 Content Management

Content Management systemer er en egen type systemer som har likheter med lagring av dokument-sentriske data, og til slike systemer er XML databaser egnet. Content Management systemer er varierte og vanskelige å beskrive i detalj, men de har ofte funksjonalitet som [7, Vandersypen side 550, 2002]:

1. Revisjonskontroll: At en bruker kan låse deler av et dokument (check-

in/check-out)

2. Versjonskontroll: At en bruker kan låse et helt dokument

3. Component assembly: Bruker eller system kan sette sammen et sluttdokument basert på revisjonskontrollen og versjonskontrollen.

Content management systemer bruker ofte XML databaser for å kunne godta dokument-sentriske data der de innkommende datastrukturene ofte er ukjente. For eksempel er slike systemer egnet hvis man vil vise data fra samme kilde i forskjellige formater, for eksempel hvis samme datakilde skal transformeres til CD-plate, Excel-regneark, PDF-format og XHTML.

5.2.2 Motivasjon for å bruke XML database

Jeg ville bruke ideer fra "Content Management"-systemer i prototyp 2, men med mye mindre funksjonalitet. Jeg ville benytte meg av bruk og gjenbruk av XSL-maler til å presentere dynamisk informasjon ved hjelp av XSL-transformasjoner. Årsaken til dette var at i denne oppgaven ønsket jeg å se på nye arbeidsmetoder og programmeringsteknikker som ANP kunne gjøre forretninger på Internett på.

I motsetning til når presentasjonsdata er hardkodet eller programmert inn i applikasjonskoden, skiller XML og XSL sterkt mellom

1. Data

2. Visning av data

3. Applikasjonskode

På denne måten kan samme kilde i XML format vises på flere måter slik det er definert i XSL-transformasjonen, og dette blir for eksempel generert til XHTML. En XHTML-fil har samme krav til å være "well-formed" som alle andre XML-filer, og å bedømme kompleksiteten for å få til dette var et mål i oppgaven.

Det er forholdsvis enkelt å skrive XSL filer, som jeg vil kalle "maler", for å vise XML data, og man kan enkelt skifte stil på applikasjonen eller forandre visningen av data.

Fordelene ved å bruke XSL transformasjoner er blant annet:

1. Gjenbruk av fragmenter av data: Med dette menes at forskjellige deler av et XML dokument kan vises basert på maler.

2. Mange output formater 1: XML dataene kan eksporteres til forskjellige media som papir, eller elektronisk til forskjellige presentasjonsformater som manualer, rapporter, brev. Man kan for eksempel legge til standard stil på dataene, for eksempel bedriftsmaler (Corporate Stylesheets) fortløpende uten mye arbeid.

3. Mange output formater 2: Man kan lettere inkludere nye plattformer, nettlesere, håndholdte PC-er, SMS eller mobilapplikasjoner.

4. XML OG XSL gjør det mulig å normalisere den samlede programmeringskoden på en mye bedre måte.
   De fleste web applikasjoner i ASP/JSP/PHP har en blanding av 4 eller flere språk i applikasjonen, for eksempel:

   1 - scripting kode i JSP/PHP/ASP
   2 - HTML
   3 - Trolig noe SQL
   4 - Server side includes på datafiler
   5 - Trolig cascading stylesheets (CSS)
   6 - Noe Javascript på klienten

   Dette gjør vedlikehold av applikasjonen vanskelig. XML og XSLT muliggjør å skille helt mellom innhold(content data), programmeringskode, og stil(style), og dette gjør det lettere å kontrollere de ulike delene av applikasjonen. Kodenormalisering og plattformuavhengighet gir en mye ryddigere struktur, og XSL og XSLT er essensielt i denne måten å strukturere data på.

Strukturen i prototyp 2 var ment å inneholde noen data som kunne forandre struktur fortløpende, og dette var XSL-filene.

Jeg hadde noen ideer til maler som kunne vært tenkt brukt i denne applikasjonen, og dette var:

1. Språkavhengige maler: Dersom man har mange versjoner av Internettsidene på forskjellige språk, kan den eller de delene av skjermbildet som er avhengig språk styres ved maler.

2. Rolleavhengige maler: Skjermbildene i applikasjonen kan styres ut fra brukernavn og passordet dersom en bruker er registret i applikasjonen. Et eksempel på dette kan være dersom forskjellige brukere av at CRM-system i en bedrift skal ha tilgang til forskjellig kundeinformasjon. De som man sitter på et kundesenter skal kanskje ha tilgang til en del av kundedataene til kunder som ringer, mens de som jobber med økonomi skal man kanskje ha tilgang til andre data, for eksempel betalingsinformasjon. Et slikt "differensiert" kundebilde kan man lage med XSL-transformasjoner.

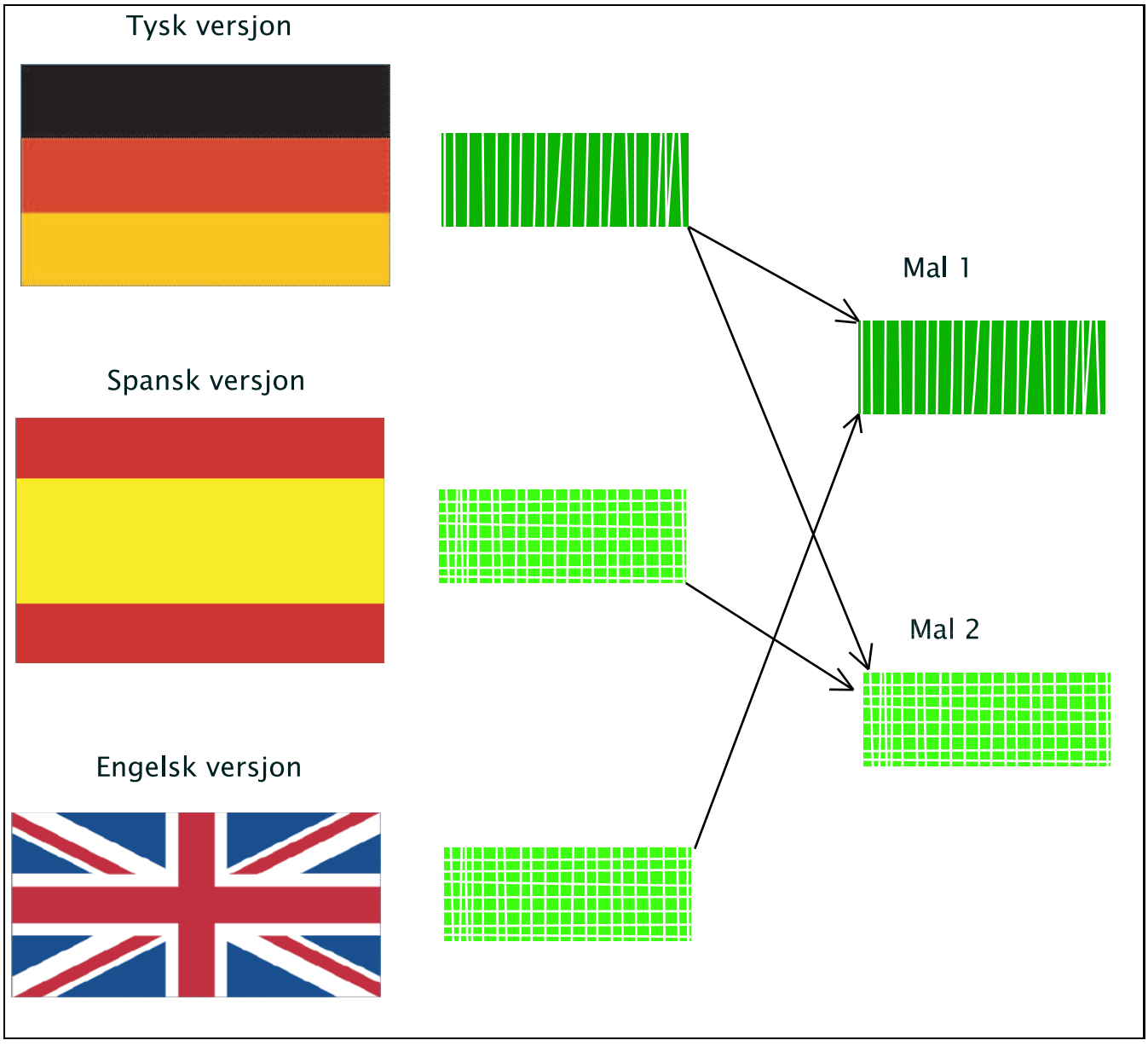
3. Valgfrie maler: En bruker av systemet kunne velge mellom maler for visning av data, for eksempel på skriftstørrelse eller bildestørrelse. Brukere som ser dårlig kan for eksempel velge skriftstørrelse 14 i stedet for 12.

Det finnes mange gode rammeverk for å publisere data med XML og XSL, for eksempel Apache Cocoon.

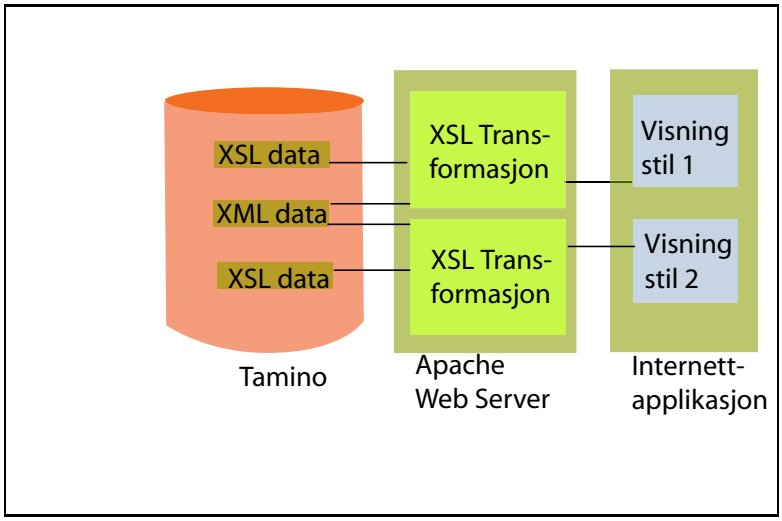
Jeg ville undersøke om dette kunne være et utgangspunkt for en Internettapplikasjon for ANP dersom jeg lagde en uniform stil på HTML sidene som malene kunne vises i. I ANP har det tidligere vært foreslått å benytte brukerbestemte XML maler, og dette var også grunnen til at jeg kunne få bruke Tamino XML Server.

Jeg ønsket å lagre alle data i XML-databasen for å undersøke om dette kunne være lettere enn å bruke en relasjonsdatabase til å generere XML-data til transformasjonen for en bestemt del av dataene, da dette ville kreve en egen administrasjon mellom Internettapplikasjon, relasjonsdatabasen og XML databasen. Jeg valgte av den grunn å kutte vekk relasjonsdatabasen, og bare jobbe med lagring av data i XML format i applikasjon tiltross for at deler av dataene i denne applikasjonen var data-sentriske og egnet for relasjonsdatabaser.

Jeg ønsket å utforske bruken av databasen Tamino XML server for å finne ut hvordan det var å bruke en "XML Native" database til lagring



Figur 5.1: En Internettside kan velge mellom flere XSL-maler, og Internettsider på ulike språk kan dele èn mal. En slik mal kan være visning av ledige hotellrom. Ved vedlikehold av visningen oppdateres kun malen.



Figur 5.2: To XSL datakilder lager to XHTML visninger fra samme XML datakilde

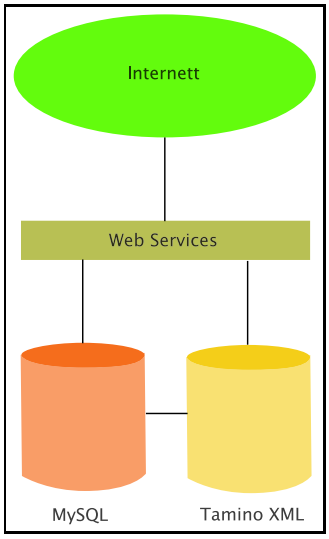
av XML. På denne måten kunne jeg lære mer om hvordan slike databaser kunne brukes, og om mulig finne ut om XML databasen kunne være basis for en applikasjon der man brukte XSL-tranformasjoner i deler av den.

I mine prototyper brukte jeg kun databaseoperasjoner for å opprette og lese XML fra databasen, ikke slette og oppdatere, men dette er selvsagt også en viktig del av en slik applikasjon.

En kombinasjon av XML database og relasjonsdatabase var også et alternativ i denne prototypen, men for å holde fokus på XML databasen ville jeg gjøre alle lagringer i denne.

5.2.3 Tamino XML Server

Denne databasen var en "XML Native" database, og det betydde at den lagret XML data uten å foreta assemblering og re-asseblering av XML dataene for å legge de i tabeller i en relasjonsdatabase. XML Native databaser ivaretar "dokumentrekkefølge, kommentarer, CDATA seksjoner og entitetsbruk"[28, Bourret, 2003], og i tillegg forbedrer det responsti-



Figur 5.3: Integrasjon mellom MySQL og Tamino over ODBC var mulig, men dette implementerte jeg ikke i noen av prototypene.

dene fra databasen.

Databasen var tilgjengelig for både Linux og Windows, og det kunne brukes både på Internet Information Services og Apache. Databasen hadde med verktøy for blant annet integrasjon med relasjonsdatabaser, og dette kunne gjøres via ODBC. Den hadde også verktøy for å lage XSD skjemaer og XQuery-spørringer.

Tamino har API for Java, C, .Net, ActiveX, JScript og PHP, og det er laget et "Developer Community" på Internett der utviklere kan diskutere programmering rundt SoftwareAGs produkter.

5.3 XML modellering

XML-databaser har en annen datastruktur enn relasjonsdatabaser. Man strukturer dataene i hierarkisk, ikke i relasjoner. Dette gjør at XML har andre muligheter innen modellering. XML modellering er fleksibelt, men til dette er det knyttet både fordeler og ulemper fordi støtten for spørrespråkene varierer.

Når man skal designe en XML-database kan man bruke en relasjonslign-

ende XML struktur ved å legge hver tabell fra en relasjonsdatabase i en XML fil, og da simulerer man relasjoner i XML, og man modellerer ikke hierarkisk. Denne framgangsmåten er det mange som bruker i XML modellering. Til mange situasjoner er det velegnet å modellere en hierarkisk struktur, for eksempel en liste med hoteller:

```
<HOTELS>
<HOTEL>
<HOTELNAVN> Hotel Regnbuen </HOTELNAVN>
<HOTELADRESSE> Kirkeveien 55</HOTELADRESSE>
<POSTNR> 0864 </POSTNR>
<POSTSTED> Oslo </POSTSTED>
<ROM>
<ROMNR> 1 </ROMNR>
<ROMNR> 2 </ROMNR>
<ROMNR> 3 </ROMNR>
</ROM>
</HOTEL>

</HOTELS>
```

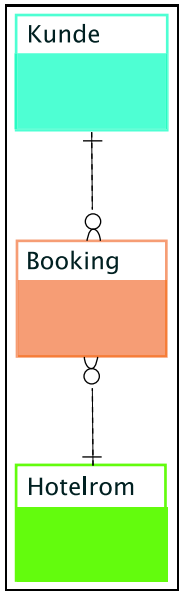
Noen datastrukturer er vanskelig å modellere hierarkisk i XML fordi de i sin natur ikke er hierarkiske.

For eksempel, dersom man skal lage en oversikt over reservasjoner på hotellrom oppstår et problem ved å modellere hierarkisk. I en reservasjonsløsning basert relasjoner er det naturlig å modellere som i figur 5.4.

Entitetene "Kunde" og "Hotellrom" er likestilt i multiplisitet mot "Booking" ved at de har samme multiplisitet mot sistnevnte entitet. I XML man må bestemme strukturen ut i fra sammenhengen, og ikke i fra normaliseringsregler, men grunnprinsippet om å unngå dataredundans gjelder også i XML-databaser.

Den er likevel slik at den semantiske bruken av dataene avgjør modelleringen sterkere, fordi dataene ofte er dokument-sentriske, og det kan fort bli større mengder dobbeltlagring av data enn i relasjonsdatabaser.

Man kan hente data fra flere XML filer enten ved bruk av XQuery eller objektorienterte programmeringsspråk som Java. Ved å bruke Java kan man jobbe med XML som objekter, og man kan lese, oppdatere data og lagre XML filene etter å ha gjort oppdateringene på dataene ved bruk av Java klasser. Man kan for eksempel bruke JAXB (JAva Xml Binding) til un-



Figur 5.4: ER-modell av forholdet mellom entitetene Kunde, Booking(Reservasjon) og Hotellrom.

marshalling og marshalling. I denne prosessen leser man XML inn i Java klasser (unmarshalling) og skriver de tilbake til XML-filer(marshalling).

Overordnet er det viktig å tenke på at XML ikke skal være er erstatning for relasjonsdatabaser, selv om alle data i min prototyp ble lagret i XML.

5.4 Min arkitektur og språkvalg

I databasen lagret jeg data fra XML filer, og jeg brukte XQuery for å hente dataene fra databasen.

5.4.1 XSL

XSL står for eXtensible Stylesheet Language, og underspråkene til XSL er XSLT, XPath og XSL Formatting Objects. Da jeg brukte XSL brukte jeg både XSLT til transformasjon av XML til XHMTL så dataene kunne leses av nettleser, og XPath til å velge ut XML dataene som skulle transformeres til XHTML. Å bruke XSL til XHTML-transformasjoner er intuitivt og raskt å lære. Det er viktig å fremheve at XSL-dokumenter også er XML-

dokumenter.

5.4.2 XQuery

XQuery er et spørrespråk som kan brukes mot flere XML filer og mot XML databaser. XQuery kan bruke XPath syntaks til å lokalisere elementer, og i tillegg støtter det å lage nye elementer, samt oppdatere og slette. XQuery kan hente data fra flere XML filer samtidig, mens XPath kun kan jobbe mot èn XML.

XQuery er et stort framskritt for spørrespråk mot XML data, og dette er noen av de viktigste egenskapene ved XQuery:

1. XQuery er et FLWOR-språk(les: flower), som er akronym for "For, Let, Where, Order, Return". Å bruke FLWOR er "en av de kraftigste og vanligste brukene av XQuery."[27, Perfectxml.com, 2003].

2. XQuery støtter XPath 2.0

3. XQuery har støtte for aggregering av data, for eksempel:
   ```
   for $b in doc("books.xml")//book
   let $c := $b//author
   where count($c) > 2
   return $b/title
   ```

4. XQuery har støtte for å opprette elementer. For eksempel:
   ```
   element book
   {
   attribute year { 1977 },
   element author
   {
   element first { "Crockett" },
   element last { "Johnson" }
   },
   element publisher {"HarperCollins Juvenile Books"},
   element price { 14.95 }
   }
   ```

   Denne lager en XML datastruktur som ser slik ut:

   ```
   <book year="1977">
   ```

<author>
<first>Crocket</first>
<last>Johnson</last>
</author>
<publisher>HarperCollins Juvenile Books</publisher>
<price>19.95</price>
</book>

5. XQuery har støtte for typiske databasespørringer med mulighet for å sortere ascending, descending og distinct, samt mulighet til å sette kriterier som "not in" og "exists".
   Et eksempel på bruk av dette er:
   for $b in doc("books.xml")//book
   where exists($b/author)
   return $b

XQuery er et omfattende spørrespråk, og det er full støtte for dette språket i Tamino.

5.5 Presentasjon av prototypen

I denne prototypen ville jeg undersøke muligheten for å lage en Internettapplikasjon basert kun på lagring av data i XML.

Det første jeg gjorde var å finne en struktur til XML filene som gjorde det så enkelt som mulig å arbeide med dataene. De 7 relasjonsdatabasetabellene som ble brukt i prototyp 1 og 3 ble denormalisert inn i fire XML filer for å kutte ned på databaseoperasjoner tilsvarende SQLs "join". Deretter overførte jeg den samme funksjonaliteten som var i prototyp 1, der jeg hadde en standard reservasjonsløsning.

Årsaken til at aktivitetsnavn og hotellnavn, og ikke kun deres unike id-er, ligger i både XML-filene for reservasjoner og XML-filen for alle hoteller og aktiviteter, er at jeg ville slippe å bruke XQuery til "join" for å finne aktivitetsnavn og hotellnavn hver gang applikasjonen ble oppdatert i nettleseren. Derfor lagret jeg disse dobbelt opp.

Jeg brukte en XML filstruktur som så slik ut:

Fil 1: allactivities.xml

```
<ACTS>
<ACT>
<actid>1000</actid>
<actid2>100002</actid2>
<start>20040204</start>
<stop>20040204</stop>
<actname>Rafting</actname>
<desc>Much fun</desc>
<difficulity>Medium</difficulity>
<price>1160</price>
<homepage>www.rafting.no</homepage>
</ACT>
</ACTS>
```

Fil 2: allhotels.xml

```
<HOTELS>
<HOTEL>
<HOTELID>1000</HOTELID>
<HOTELNAME>Regnbuen</HOTELNAME>
<DESC>1 klasse</DESC>
<ROOMID> 100001 </ROOMID>
<NR_OF_BEDS>3</NR_OF_BEDS>
<PRICE>1460</PRICE>
<HOMEPAGE>www.hotelregnbuen.no</HOMEPAGE>

    </HOTEL>
</HOTELS>
```

Fil 3: actbook3.xml (data om reservasjon av aktivitet)

```
<ACTBOOKS>
<ACTBOOK>
<CUSTID>222222</CUSTID>
<ACTID2>100002</ACTID2>
<ACTNAME>rafting</ACTNAME>
<START>20040204</START>
<STOP>20040204</STOP>
</ACTBOOK>
</ACTBOOKS>
```

Fil 4: hotbook.xml (data om reservasjon av hotell)

<HOTBOOKS>
<HOTBOOK>
<CUSTID>222222</CUSTID>
<HOTELID>1000</HOTELID>
<HOTELNAME>Regnbuen</HOTELNAME>
<ROOMID>100001</ROOMID>
<START>20041201</START>
<STOP>20041202</STOP>
</HOTBOOK>
</HOTBOOKS>

Alt som ble gjort av databaseoperasjoner gikk gjennom en PHP API mot Tamino. PHP APIen lastet jeg ned fra SoftwareAG sine hjemmesider.

Jeg implementerte prototypen ved at jeg først lagret oppgitte datoer og brukerens kundeid fra sesjonsvariabler i PHP, slik jeg gjorde i alle 3 prototypene. På PHP/HTML-sidene act1.php, act2.php og hot1.php instansierte jeg på hver av sidene to objekter av klassen TaminoAPI. I disse objektene lå det metoder for å opprette en forbindelse til databasen og metoder for å lese, legge inn og slette data. Jeg brukte kun metoder for å lese og legge til. For lese ut data brukte jeg XQuery, for eksempel:

/ACTS/ACT[start>user_start_date and stop<user_stop_date]

Jeg trengte ikke å bruke noen datofunksjoner, fordi jeg skrev datoene i formatet YYYYMMDD, og ved å gjøre dette kunne jeg bruke ”større enn” og ”mindre enn”-sammenligning direkte på datoene i tekstformat.

Da jeg skulle legge nye elementer inn i databasen, for eksempel legge til en reservasjon av aktiviteter ble disse elementene først skrevet til harddisken, og deretter lastet inn databasen. Jeg lagde metodene for å skrive data til fil selv, da disse ikke var en del av APIen til Tamino. Jeg kunne også brukt XQuery til å opprette data.

5.6 Testresultater

5.6.1 Installasjon

Installasjonen av Tamino var enkel å gjennomføre fordi det var laget et installasjons program som fant alle nødvendige innstillinger automatisk.

Å opprette en database var intuitivt fordi Tamino hadde et brukergrensesnitt som var logisk og som lagde en database med standardinnstillinger dersom man ønsket det. Det eneste man valgte var om databasen skulle være liten, medium eller stor, og om man hadde mange skriveoperasjoner i databasen eller mest leseoperasjoner. Jeg valgte det første fordi jeg brukte mange temporære lagringer av data i timeplanene jeg satt opp for kunden.

I tillegg måtte jeg installere Apache Web Server fordi PHP ikke klarer å bruke alle XML-bibliotekene når man bruker IIS. I initialiseringsfilen til PHP bestemte jeg portnummer som PHP skulle aktiveres på, og hvilke innstillinger PHP skulle ha for å skrive ut feilmeldinger, regler for timeout og maks filstørrelse. Denne systemfilen var godt dokumentert, og dette gikk uten problemer.

Jeg måtte også installere Sablotron for å bruke XSLT transformasjoner. Dette krevde flytting av 3 .dll filer inn i System32-katalogen i Windows, men dette var dokumentert i installasjonsbeskrivelsen.

5.6.2 Implementasjon

I arbeidet med datastruktur og databaseoperasjoner, der alt var basert på lagring i XML, oppsto noen problemstillinger som krevde en ny tankegang overfor databaser. Fra relasjonsdatabaser var jeg vant til at man la data i tabeller, men i Tamino eksisterte ikke tabellbegrepet. Det var ingen krav til struktur annet enn at når man lastet XML filer inn i databasen eller oppdaterte den, skulle de være knyttet til et XSD skjema (XML Schema Definition). Filer som ikke var i XML-format kunne også lastes inn i Tamino.

All kommunikasjonen, slik jeg programmerte den, fra applikasjonen mot databasen gikk gjennom en API skrevet i PHP. Å gjøre operasjoner gjennom denne APIen utover det som var beskrevet i et eksempel fra SoftwareAG var vanskelig, og SoftwareAG hadde heller ingen support på PHP APIen.

Eksempelet som demonstrerte APIen hadde metoder for å lese , legge til og slette data. Det var ikke laget et eksempel på hvordan man skulle oppdatere data tilsvarende ”update” i SQL.

Måten jeg la til data i Tamino på, var å laste inn XML datafiler og XML skjemaer i databasen med funksjonen processData(filnavn). Ved å gjøre dette ble dataene lagret i Collections, som er betegnelsen for en samling XML data i Tamino. En Collection er definert av en XSD-fil, og den har alltid et unikt navn i databasen.

Fordi dataene jeg la til i databasen først ble skrevet til fil, måtte jeg lage metoder i PHP for å skrive XML filer til harddisken. Disse filene var basert på data hentet fra sesjonsvariabler i PHP, samt brukerens valgte verdier i applikasjonen. Ved videreutvikling ay systemet ville jeg brukt XQuery til dette.

Dataflyten mellom Tamino og PHP var logisk ved at man kunne laste inn XML filene fra harddisk og hente de tilbake inn i nodesett ved å bruke XQuery. Etter at XML-filene var lastet inn i Tamino hadde de ingen relasjon til harddiskområdet de ble lastet opp ifra, og de kunne således slettes eller flyttes. Hvis ikke filene ble fjernet ville de lastes opp på nytt neste gang applikasjonen gjorde en processData med det samme filnavnet som parameter. Jeg gjorde spørringer mot Tamino i XQuery med FLWOR-syntaks

Å gjøre spørringer i XQuery utover standard XPath syntaks var ikke gitt ved noe eksempel i PHP APIen fra SoftwareAG, og det fantes heller ingen dokumentasjon på hvordan dette skulle gjøres i PHP, men Java APIen hadde dokumentasjon for dette. Spesielt kunne jeg trengt informasjon om hvordan man hentet data fra to Collections i samme spørring. Alle eksempler jeg fant som gjorde dette forutsatte en fil med èn av datakildene, men i Tamino har man ikke filer fordi man henter data fra Collections. Det bør derfor undersøkes om det er bedre å implementere applikasjonen i JSP eller ASP fordi SoftwareAG har support på disse språkene, men ikke PHP. XQuery som støtter FLWOR var ikke vist med noe eksempel, og jeg klarte derfor ikke å gjøre en spørring mot flere XML Collections i samme spørring, men det jeg løste det ved at jeg hentet de reserverte hotellrommene fra databasen og la de i en array, og så genererte jeg en XQuery-spørring ut fra denne.

Dersom man skal lagre data i "XML Native" format, for eksempel om man vil lagre XSL-filer som forandrer seg kontinuerlig, er det viktig å huske på: Å modellere en hel Web applikasjon i XML krever stor innsikt i XML generelt og XML modellering. Man bør også ha god kjennskap til XML teknologiene XSL, inkludert XPath og XSLT, samt XQuery og DOM

før man starter arbeidet.

Jeg opplevde modelleringen av XML-databasen vanskeligere enn relasjonsdatabasen. Dersom applikasjonen er liten og oversiktlig, slik som min, kan denne modelleringen være en enkel oppgave, men dersom man har en kompleks datastruktur med mange spørringer kan arbeidet bli vanskelig.

I tillegg var XQuery ikke en W3C-standard da oppgaven ble skrevet, og det var vanskelig å finne eksempler som brukte XQuery i PHP.

Det er viktig å skille vanskeligheten i å lage spørringer og modellere XML-systemer, som jeg mener har en forholdsvis høy vanskelighetsgrad, fra det å jobbe med XML-data i XSL-transformasjoner. Å jobbe med XSL-transformasjoner er forholdsvis enkelt å lære, og teknologien skiller applikasjonslogikk og programmeringskode fra data som skal presenteres, slik at helheten i systemet blir mer oversiktlig.

Da jeg skulle jobbe med XML data som kom fra XQuery-spørringer, var dette arbeidet i første omgang vanskeligere enn at dataene kom fra resultatsett fra en relasjonsdatabase. Grunnen til dette var at jeg først implementerte prototyp 2 uten å bruke XSL, men i stedet kun PHP. Jeg ville legge til XSL i etterkant for å se på forskjellene.

Dette gjorde jeg i et av skjermbildene(aktivitetsreservasjonen) i prototypen og kalte den prototyp 2b, og til denne lagde jeg to XSL-maler.

Ved å ikke bruke XSL i første omgang gjorde jeg det vanskeligere for meg selv enn nødvendig. Mye programmeringskode i applikasjonen ble brukt for å hente verdiene av XML-dataene i tillegg til formateringen av visningen av dataene, og dette gjorde at applikasjonen ble lite strukturert.
Dette ble lettere ved å bruke XSL som i prototyp 2b.

PHP APIen inneholdt et eksempel for å jobbe med XML man ikke vet elementnavnene i, fordi denne brukte DOM til å lese ut strukturen. Denne er nyttig hvis man ikke vet hvilke elementer man mottar i XQuery-spørringen. Jeg brukte denne, men jeg hadde ikke måttet det, fordi jeg viste elementnavnene som ble returnert fra spørringen, og da kunne jeg hentet verdiene ved bruk av XSL.

5.6.3 Prototyp 2b med XSL transformasjoner

Etter å ha implementert applikasjonen uten bruk av XSL, men kun med PHP "while"-løkker og "if"-tester, ønsket jeg å skille applikasjonskode, data og presentasjon i prototyp 2b. Ved å bruke XSL fikk jeg helt nye muligheter til å bruke og gjenbruke maler som jeg kunne brukt hvis jeg for eksempel skulle laget det samme systemet på flere språk.

En annen fordel var at jeg fikk generert XHTML, som har et strengere krav til struktur og derav er mer forutsigbart enn HTML-koder der man ofte har små feil, for eksempel i rekkefølgen av "markup"-taggene.

Etter at jeg hadde fått XSL-transformasjonene til å virke, lagde jeg mer funksjonalitet i prototyp 2b for å prøve ut å programmere XHTML-sider som brukte maler, og jeg lagde blant annet to maler til å vise de samme dataene med forskjellig presentasjon.
De to malene lagde jeg for å vise hvordan XSL kan brukes til å tilpasse HTML-sider for potensielle kunder som bruker applikasjonen. Den ene malen var tenkt å ligge under "Dansk" språkversjon, den andre under "Japansk", dersom man velger språk i applikasjonen.

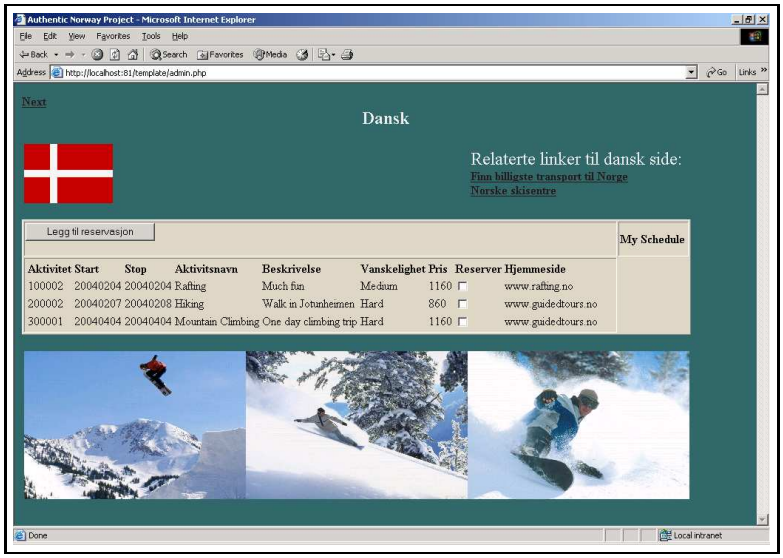
Å starte en XSL-transformasjon i PHP gjør man med èn kodelinje. Slik startet jeg transformasjonene i prototyp 2b):

```
$xml = domxml_dumpmem($domnodeResultElement);
$arguments = array('/_xml' => $xml);
$xh = xslt_create();
if ($testparam == 1)


    $result = xslt_process($xh, "arg:/_xml", "dansk.xsl", NULL, $arguments);

else


    $result = xslt_process($xh, "arg:/_xml", "japan.xsl", NULL, $arguments);
```

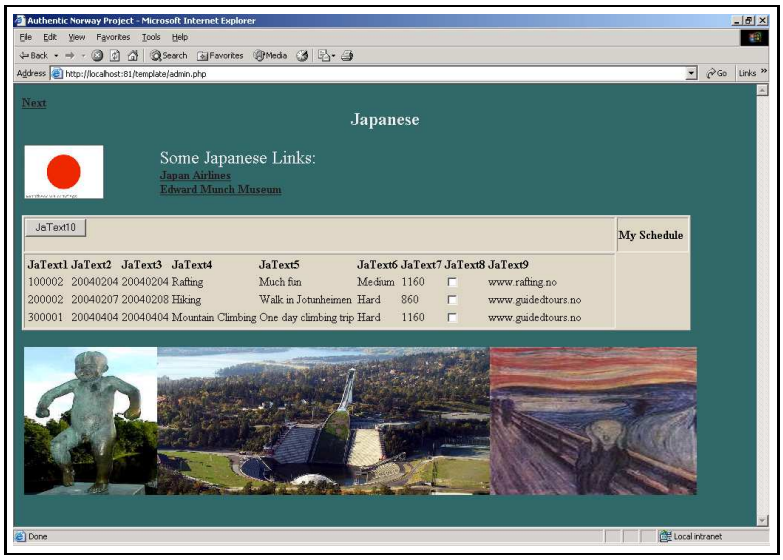
Ved å sjekke verdien av en variabel, $testparam, i applikasjonen kunne jeg starte opp en valgt XSL transformasjonsmal, og i tillegg kunne flere HTML/PHP-sider dele den samme malen. Det siste medfører at en

forandring av visningen av presentasjonsdataene kun trengs å oppdateres i malen.

Tilslutt i dette kapittelet vil jeg fremheve at det viktigste for meg ikke kun var å oppnå færre kodelinjer, men at den logiske strukturen på en stor Internettapplikasjon skulle bli lettere.



Figur 5.5: Denne malen er lagt til på en dansk versjon av skjermbildet for reservasjon av aktiviteter. XSL-transformasjonen het dansk.xsl.



Figur 5.6: Dette er den japanske versjonen av siden. Det er samme side i applikasjon som henter denne malen som den i bilde 5.5. I den japanske malen ligger det andre bilder og linker enn i den danske.

Kapittel 6

Prototype 3 - Web Services

"The most successful businesses on the Internet will hunt in packs".[9, Reed, 2001]

Prototyp 3 var en distribuert løsning med Web Services og SOAP, og denne arkitekturen muliggjorde at de deltagende bedriftene kunne bruke sine eksisterende systemer i tillegg til å tilgjengeliggjøre sine data til en annen applikasjon over Internett.

Presentasjonen av min løsning starter i 6.3. , men først kommer en oversikt over Web Services som min løsning bygger på, samt en forklaring på hvorfor jeg valgte å bruke dette.

6.1 Hva er Web Services

Det er vanskelig å finne en distribuert teknologi som er plattformuavhengig og som også har lav kompleksitet. På dette området skjer det imidlertid en utvikling. Microsoft og IBM er ledende selskaper innen en del av denne utviklingen som heter "Web Services".

Web services er løst koblede, kontraktbundne komponenter som kommuniserer via XML-baserte grensesnitt. Løst koblede betyr at Web Services og programmene som bruker dem kan forandres uavhengig av hverandre så lenge de tilbyr det samme grensesnitt mot andre applikasjoner. I tillegg er Web Services også plattformuavhengige.

Kontraktbundet betyr at tjenester er spesifisert og publisert, og fordi det er en komponent, er implementasjonen av komponenten gjemt for

Forretningsverdenen trenger bedre teknikker for å skalere forretningsløsninger uten å måtte øke kompleksiteten til uhåndterlige nivåer. I tillegg så er det et klart behov for åpne, fleksible og dynamiske løsninger for å muliggjøre globale e-handelsinteraksjoner mellom systemer. Web Services-modellen vil levere disse løsningene ved å begrense kompleksitet og kostnader ved å tilby et vanlig språk for B2B e-handel og samtidig muliggjøre visjonen om en global e-handelsplass.

Tabell 6.1: Vandersypen skriver følgende:[Vandersypen, 7, 2002]

utsiden, det vil si at komponenten er innkapslet og tilgjengelig via grensesnitt.

De to hoveddelene av Web Services er XML og HTTP. Kommunikasjonen i Web Services blir gjort gjennom en listener og en XML wrapping rundt mellomvare på en Internett server som tilbyr Web Services. En forespørsel pakket inn i XML blir foretatt, og resultatet blir oversatt videre til en forespørsel mot mellomvaren, og svaret fra denne blir konvertert tilbake til XML. I tillegg trenger man å bruke SOAP protokollen til å sende forespørsler og resultater over Internett. Man kan også bruke WSDL og UDDI til å beskrive og publisere tilgjengelig Web Services, men dette er valgfritt.

SOAP er en XML basert meldings- og fjernprosedyrekall (RPC) spesifikasjon som muliggjør veksling av informasjon mellom distribuerte systemer[7, Vandersypen side 630, 2002], og det er en protokoll som er enklere å bruke enn DCOM og CORBA. Den er XML-basert, altså tekstbasert, og leselig for mennesker. Både DCOM og CORBA er binære protokoller.

Det er også interessant og nyttig å vite at SOAP, i motsetning til DCOM og CORBA, er løst koblet. Derfor er det mulig å utveksle SOAP meldinger "on-the-fly" i kjøretid hvis JIT (just-in-time) integrasjoner støttes av disse tjenestene.

Dette gir muligheten til å lage programmer og Internettsider som automatisk kan lete gjennom titalls og hundretalls av Internettjenere for å

skaffe informasjon der Web Services tilbys. Denne måten å jobbe på har stort potensial for å gjøre informasjon tilgjengelig for sluttbrukere.

Noen snakker om "den fjerde generasjons Internet" som "som en tsunami vil ryste det konservative, og lite forandringsvillige Internett med enorme fordeler og retningslinjer". [10, Pollock, 2002]. Man snakker her om bruken av Web Services, å bevege seg fra hardkodede applikasjoner til Web Services som samarbeider i run-time. De fire nøkkelbegrepene i denne teknologien er implementering, publisering, oppdagelse og påkalling(eng:invocation). [7, Vandersypen, 2002].

En slik programmering signaliserer et paradigmeskifte i distribuert programmering. De har potensial til å forandre måten distribuerte systemer snakker sammen, noe som fundamentalt vil forandre hvordan man bruker de over Internett. Som et resultat av dette kan Web Services bidra til å etablere å ny måte å gjøre e-handel på, fordi flere kan delta uten å måtte investere store pengesummer.

6.1.1 Web Services historie

Microsoft publiserte SOAP sent i 1999, og det var da en av mer en 15 aktuelle kommunikasjonsprotokoller for XML. I april 2000 offentliggjorde IBM at de ville støtte opp om SOAP, og ikke lenge etter at dette ble publisert, så leverte ti store programvareselskaper, deriblant Microsoft, IBM, Sun (tiltross for at Sun var totalt imot SOAP i starten av 2000) og Oracle, et forslag til SOAP V1.1. W3C opprettet “XML Protocol Working Group”, og alle disse selskapene ble med i denne arbeidsgruppen

UDDI ble annonsert i august 2000, og ikke lenge etter dette annonserte alle disse selskapene støtte for UDDI og WSDL. Uttrykket "Web Service" oppsto i slutten av 2000.

I 2000 arbeidet de fleste store av de involverte, unntatt Microsoft, med å standardisere og tilgjengeliggjøre ebXML, og dette arbeidet var en viktig grunn til at "SOAP with attachments" ble utviklet. Tiltross for at det så ut til å være enighet om "SOAP with Attachments" hos de fleste store selskapene, klarte ikke Web Services Interoperability Group” å bli enige om at dette skulle godkjennes av de, og ikke lenge etterpå erstattet Microsoft “SOAP with Attachments” med sin DIME-spesifikasjon.

Likevel kom det ut en del produkter med ebXML implementasjoner, men da Microsoft og IBM ga ut et alternativ til ebXML, GXA, og IBM i tillegg startet å trekke seg i sin deltagelse fra utviklingen og satsningen på ebXML, så ble ebXML sterkt svekket. Det gjorde det heller ikke bedre at ebXML komiteene delte seg mellom OASIS og UN/CEFACT.

En viktig årsak til at ebXML ikke klarte å bli en enerådende standard var at denne teknologien var svært kompleks fordi den var et sammensatt rammeverk som tok sikte på å løse de fleste problemene innen web services. Eksperter sier at ebXML på mange måter førte web services tilbake til et å bli et EDI-system, slik man hadde hatt tidligere på sytti, åtti og nittitallet, og dette var noe man ønsket å komme bort ifra.

Etter dette har det skjedd en ytterligere splitning innen Web Services fordi de store selskapene har utgitt forskjellige spesifikasjoner for sikkerhet, "messaging" og Web Services koreografier, og i tillegg til dette har Microsoft trukket seg ut av "Web Services Architechture Group" på grunn av patent -og opphavsrettstridigheter. W3C og Microsoft samarbeider lite innen sikkerhetsarbeidet i Web Services.

6.1.2 Web Services i CRM sammenheng

Web Services er en teknologi man kan bruke for å gi tilgjengelighet til et system fra Internett. Dersom man bruker disse tjenestene kan andre Internettsider eller applikasjoner programmere mot din Internettapplikasjon og hente ut de data du ønsker andre skal ha tilgang til. På den måten kan en Internettside samle data fra mange andre steder på Internett og presentere de for eksempel samlet på èn side. Således blir det lettere for Internettbrukeren å finne mye data uten å måte gå fra Internettside til Internettside, og det blir lettere å integrere B2B systemer.

I CRM-sammenhenger kan man tenke seg mange måter å bruke dette på. Dersom man ønsker å gjøre sine priser og tjenester tilgjengelig på andre Internettsider kan man lage Web Services som tilbyr disse dataene. For eksempel kan man tilby:

- Kapasitetslister (for eksempel ledige plasser til et hotell)
- Værvarsel
- Priser

- Spesielle arrangementer

Enkelte portaler jobber i dag mot mange andre Internettsteder, og de kan samle data fra mange Internettsteder for så å presentere disse på èn side.

En mulighet for ANP er å gjøre sine data tilgjengelige på utenlandske Internettportaler, enten ved å gi tilgang til alle eller ved å bruke autentisering for bruk av tjenesten.

Det er spådd at Web Services vil bli mye mer utbredt i tiden som kommer. "30 prosent (pr. juni 2003) av 240 amerikanske bedrifter i gang med å innføre Web Services"[25, itavisen.no, 2003]. Det er derfor viktig for ANP å undersøke om teknologien kan underbygge og støtte de salgs -og markedsføringsprosesser som ANP vil bruke både i dag og i fremtiden, og jeg syntes det var interessant å lage Web Services på ANP sin portal, for å undersøke hvordan disse kunne brukes.

Bob Thompson fra CRMGuru skrev i en artikkel hans 5 viktigste spådommer for CRM i 2003[6, Thompson, 2003]: "Uavhengig av terminologi, i 2003 vil Web Services endelig komme i CRM-industrien, og du vil se virkelige casestudier, ikke kun elektronisk dokumentpublisering (eng:brochuware)".

Før jeg skriver om noe av de tekniske detaljene og hvordan dette programmeres, vil jeg først gi noen eksempler fra næringslivet på hvordan dette kan brukes.

6.1.3 Eksempler på bruk

Eksempel 1
Store Internett-portaler, for eksempel Banca Monte dei Paschi di Siena, Italias femte største bank, bruker disse tjenestene på deres Internettportal i dag. Dette er et system med 5 millioner brukere og 28.000 ansatte. Banken lagde en informasjons- og rådgivningsportal basert på Web Services. Banken bruker standard web services med SOAP og XML teknologi.

Eksempel 2
En annen bruk av XML og Web Services, er det Microsoft gjorde med Siebels CRM-system. De ansatte i Microsoft ble lei av de dårlige brukergrensesnittene i Siebels system. De lagde derfor "wrappers" til sine ek-

sisterende COM-objekter, som er teknologien som Microsoft ofte bruker i sine produkter, i XML og SOAP. På denne måten ville brukerne kunne bruke Excel og Access brukergrensesnitt til å skrive inn dataene, og fra disse grensesnittende ble dataene lagt inn i Siebels system ved hjelp av XML transformasjoner.

Dette er, i følge Forrester Research, en av de mest spennende brukene av disse tjenestene fordi med denne teknologien kan man integrere funksjonalitet fra for eksempel Microsoft og et annet system, for eksempel SAP eller Siebel, og integrere dette på en Internett-portal. Sluttbrukeren ville ikke se at hun eller han jobber mot to eller flere systemer.

Det er spådd at dette kan løse mange integrasjonsproblemer mellom programvare, og CRM systemer vil også nyte godt av dette.

6.1.4 Implementering av Web Services

Web Services gjør programvarefunksjonalitet, som tidligere skrevet, tilgjengelig på Internett, slik at nettsider kan sende forespørsler og metodekall til applikasjoner som ligger på andre tjenere så lenge de er tilkoblet til Internett. Resultatene man får kan vises på en nettside eller lagres i "back office"- programmer. Teknologien har blitt markedsført stort de siste årene blant annet av programvareleverandører som Microsoft, IBM, HP, Sun og Oracle, men denne teknologien er ikke ny. Corba og DCOM har tidligere blitt brukt i distribuert programmering, men tilgjengeligheten til å utføre og delta i distribuert programmering på Internet ved å bruke XML sammen med tradisjonelle Internettspråk er ny. Corba og DCOM er teknologier som er vanskelig å bruke.

Flere programmeringsverktøy som Microsoft Visual Studio og AXIS, som jeg brukte, kan automatisk generere disse tjenestene fra en applikasjon.

6.1.5 Problemer innen Web Services

Standardiseringsproblemene
Det finnes mange problemer innen Web Services som er vanskelig å løse, og som fortsatt ikke er løst i dag.
Når man skal lage en Internettløsning med Web Services er det mange problemer man må ta hensyn til. Det finnes flere teknologier som er basert på XML som man kan bruke, men slik det er i dag, kan man ikke

uten videre bestemme seg for at man vil ha det beste, raskeste og tryggeste av alt, slik man kan i ikke-distribuerte lukkede miljøer. Avhengig av hva som er viktigst i løsningen må man vurdere flere angrepsvinkler og teknologibruk, og man må vurdere krav til sikkerhet som kryptering, digitale signaturer, autorisering, autentisering, samt hvor langt man er villig til å strekke seg i kompleksitet og pris før man bestemmer seg for å løse problemet på en annen måte. Det er på ingen måte ønskelig at det skal være slik, og man jobber hardt på alle disse områdene for å gjøre Web Services lettere tilgjengelig, og for å oppfylle den visjonen man har rundt bruken av det.

I dag finnes det svært mange forskjellige standarder , og jeg skal gå raskt inn på å skissere noen viktige problemer, og litt om bakgrunnen. At Web Services ofte blir kalt "CORBA for the Web", sier litt om kompleksiteten som har oppstått.

Problemområde: Lite kontroll på visning av data hos andre
Dersom man ønsker et spesielt design til sine data, kan dette være vanskelig å få til. Mange Web Services henter kun ut tekstlige data og presenterer disse i lister, for eksempel sortert på pris, og dette kan være negativt dersom man ikke får fram data som underbygger prisen, for eksempel ved hjelp av design. Både høye og lave priser kan virke avskrekkende dersom dataene presenteres i feil sammenheng.

Problemområde: Reliable Messaging
Reliable messaging er løst med to forskjellige standarder. SUN bruker WS-Reliability , mens blant andre Microsoft og BEA bruker WS-ReliableMessaging.

En av de viktigste egenskapene ved Web Services er kommunikasjonen over usikre kanaler som Internett. Internett garanterer for eksempel ikke levering av pakker over HTTP, SMTP eller FTP. Flere teknologier er laget for å løse dette problemet fordi det er kritisk at man har "reliable messaging" i for eksempel virksomhetskritiske forretningsapplikasjoner.

Forløperne til "reliable messaging", og det som dagens løsninger er bygget på, er RosettaNet, Biztalk og ebXML. Teknologien som brukes er "message ques" som kan sikre at beskjeder og pakker kommer fram mellom sender og mottaker. "message ques" sikrer at systemer kan kommunisere i "ettertid" dersom nettverket ikke er operativt. Pakkene som skal sendes legges i køer som koordineres, og disse tar hensyn til om nett-

verket er operativt eller ikke.

Problemområde: Web Services Choreography
Innen Web Services koreografi, også kalt orchestring, finnes det en rekke ulike språk. WSDL er allment godtatt som basis på nivået før orchestring, men WSDL tilbyr bare tilstandløse grensesnitt. Dette betyr at WSDL ikke har noen måter å beskrive samarbeid mellom Web Services der applikasjonene kan være i forskjellige tilstander eller faser. Det som bygger på beskrivelser av tjenestene på et kompleksitetsnivå over WSDL, er lite standardisert.

De to største språkene for koreografier er BPEL4WS (Business Process Execution Language For Web Services) og WSCI (Web Services Choreography Interface).

Forsinkelser
Forsinkelser (eng: Latency) er et stort problem. For at et system skal bli en suksess bør det være kontroll på hvor lang tid brukeren må vente for å få svar fra systemet, og dette er vanskelig å forutsi med Web Services. For eksempel kan et system være inne i en databaseoppdatering og eller være ute av drift, og klientapplikasjonen som benytter tjenesten vil da ikke gi respons til sluttbrukeren som den skal.
Det er laget verktøy for å overvåke Web Services, deriblant latency, og et mye omtalt verktøy er HPs OpenView som lar brukeren av dette programmet kunne overvåke trafikk og responstider.

Transaksjoner
Et av de store problemene rundt disse tjenestene er transaksjoner.
I distribuerte databasesystemer bruker man "2-Phase Commit" til å løse dette, men med Web Services eksisterer det ikke en slik kommunikasjon mellom databaser.

Det jobbes mye med transaksjoner innen web services. Eric Newcomer skrev om transaksjoner innen Web Services som "et av de siste hindrene for mainstream bruk av Web Services".[26, Eric Newcomer, 2003]. Dette ble skrevet i september 2003, men inntil transaksjoner er en standard må det bestemmes et sett av regler for hvordan Web Services skal brukes.

Sikkerhet generelt
Sikkerhet innen Web Services er et tema som står svært sentralt når man

skal flytte sin forretningsdrift til Internett, og jeg vil gå innom noen viktige aspekter og språk innen Web Services og sikkerhet på Internett.
Flere standardiseringsorganisasjoner og selskaper jobber med sikkerhetsproblemene, og noen store er OASIS, W3C, IETF, Microsoft og Sun.

Sikkerhetsspråk 1: AVDL
I april 2003 bestemte en komite fra OASIS, kalt "Web Application Security Technical Comittee", å lage et språk for informasjon om sikkerhet innen Web Services og Internettapplikasjoner. Språket heter AVDL (Application Vulberability Description Language). Kevin Ehineman fra "AVDL Techical Commite" forklarte: "Målet med AVDL er å få bedrifter til å styre og forenkle den totale applikasjonssikkerheten ved å tilby en uniform måte å beskrive sårbarheter innen applikasjonssikkerhet, retningslinjer og hendelser ved hjelp av XML."

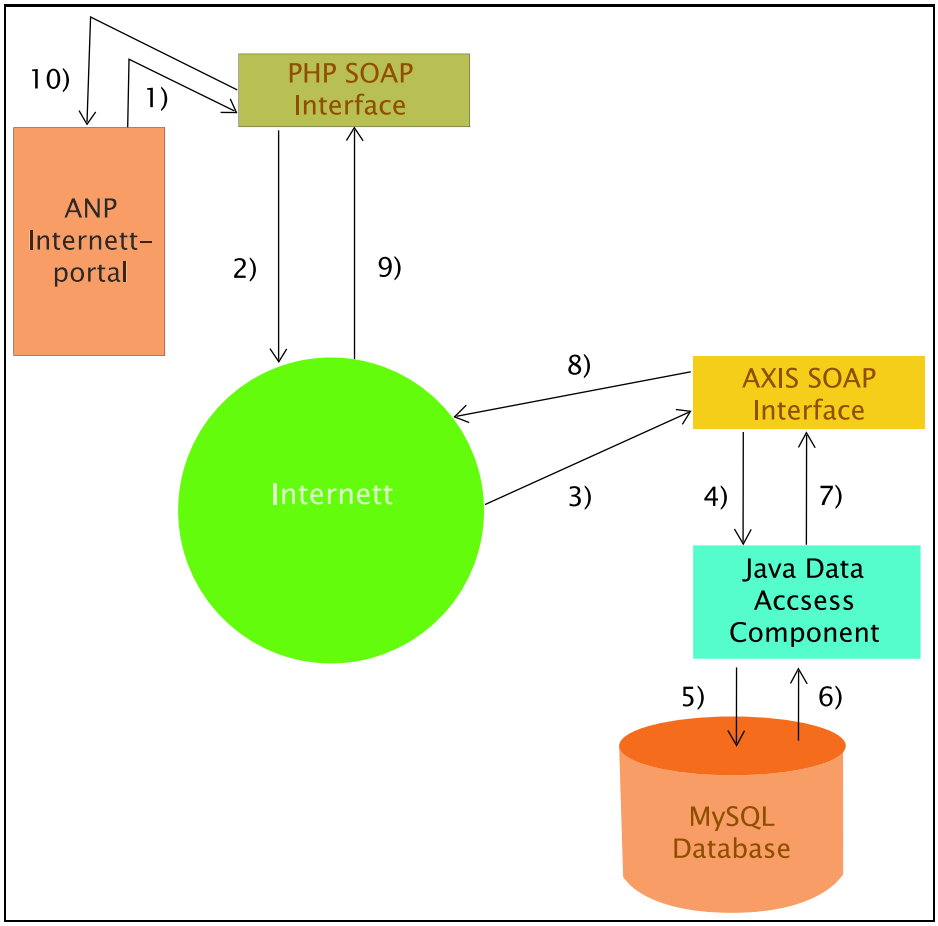
Sikkerhetsspråk 2 :SAML
OASIS har godtatt SAML (Security Assertion Markup Language) 1.1 som en OASIS standard. SAML er et XML-basert rammeverk for å spesifisere autentisering og autoriseringsinformasjon. SAML spesifikasjonen støttes i dag av mange av de store programvareleverandørene som BEA, SAP og SUN.

Digitale signaturer
Digitale signaturer er standard i sikkerhet og kryptografi og det brukes også innen Web Services. Den digitale signaturen forsikrer at pakken med data som mottakeren får er uendret og har sin originale integritet, og i tillegg kan digitale signaturer sjekke at koden eller beskjeden er sendt av den som hevder å ha sendt pakken.

6.2 Min motivasjon for å bruke web services

Det er flere ting som tyder på at reising og turisme vil bli berørt av disse nye tjenestene. En viktig grunn er at reising og turisme er styrt av at de som leverer tjenesten består av mange ledd (supply-side fragmentation). Dette betyr at mange selskaper er involvert i reising, spesielt innen reiser som innebærer overnatting eller ferie, som er en bransje der mange bedrifter fortsatt nesten bare bruker telefon til å gjøre avtaler. Det er derfor et stort potensial i dele informasjon bedre mellom leverandørene.

Mindre bedrifter vil ikke ha økonomi til å utvikle Internettbaserte tje-

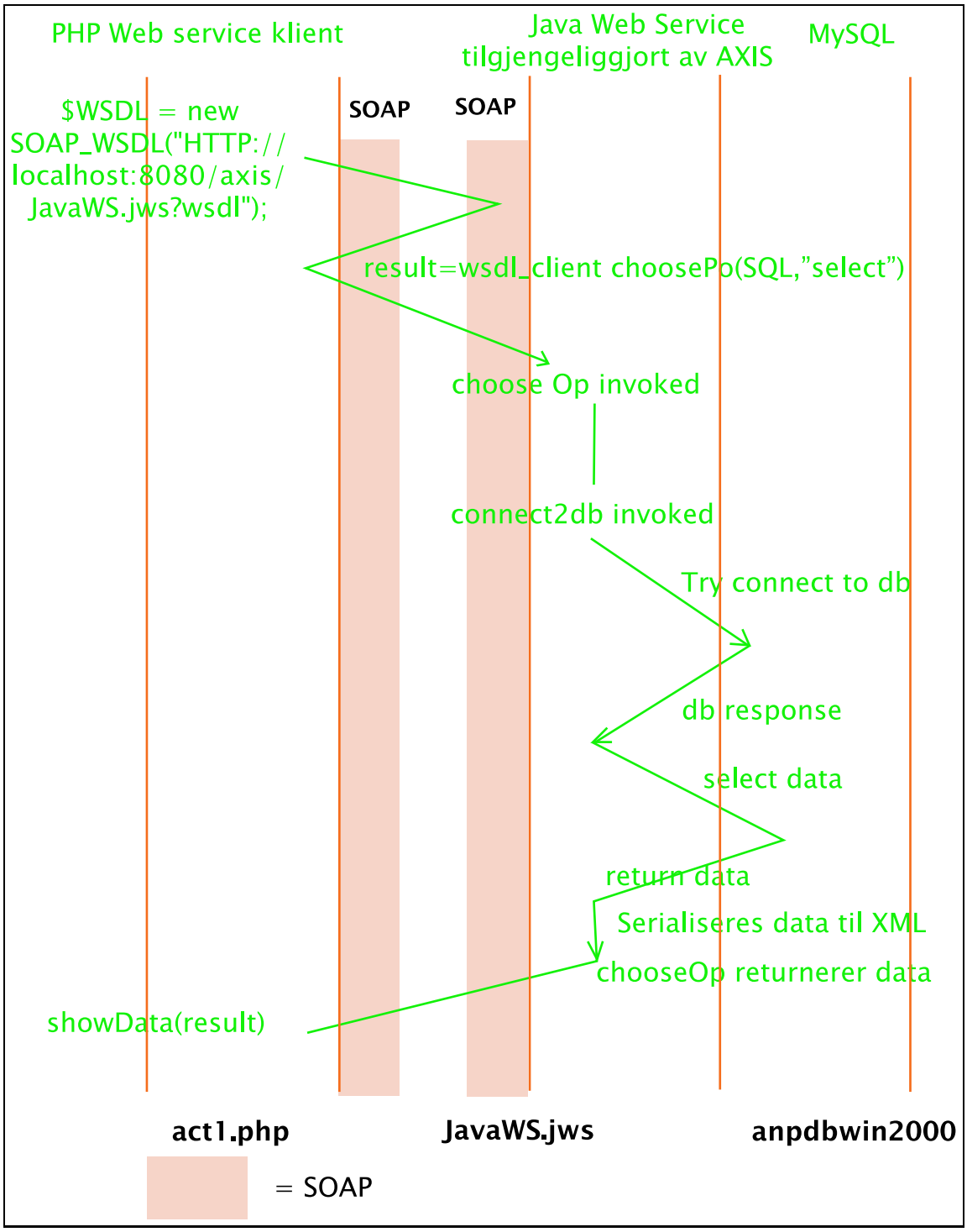


Figur 6.1: Arkitekturen i min implementasjon. Tallene ved siden av pilene viser rekkefølgen av dataflyten fra ANP portal gjør et kall på tjenesten til dataene mottas av ANP Portal.

nester selv, men hvis man kan lage standard tjenester som selskapene kan bruke, så kunne disse selskapene automatisere oppgaver som ellers måtte gjøres manuelt.

6.3 Min arkitektur og språkvalg

Min arkitektur besto av en Web Services Server og en Web Services klient.



Figur 6.2: Dataflyten mellom PHP, SOAP, Java og MySQL

6.3.1 Java

På tjeneren som tilbød Web Services brukte jeg Java, men det er viktig å merke seg at når man bruker Web Services bør programmeringsspråket på tjeneren der tjenesten tilbys, være uviktig for klienten. Klienten kommuniserer ikke direkte med applikasjonen som tilbyr tjenesten, men med SOAP.
Jeg implementerte en standardløsning med Java og MySQL der jeg brukte SQL og JDBC til databaseoperasjonene på server. Web Services klienten står fritt til å velge programmeringsspråk på klientapplikasjonen.

Jeg valgte Java fordi AXIS, som jeg brukte til å lage Web Services, var laget for Java, og i tillegg har språket et godt utbygd rammeverk for å jobbe mot databaser, og det er det viktigste programmeringsspråket på IFI i dag.

6.3.2 AXIS

Apache eXtendable Integration System er et program som er laget for å implementere Web Services, og det er laget i Java. AXIS klarer å tilgjengeliggjøre Java komponenter over SOAP, og det er godt dokumentert.

Som database brukte jeg MySQL.

6.4 Presentasjon av prototypen

Løsningen jeg laget fremstår som lik for sluttbrukeren som løsningen i prototyp 1 og 2, men den tillater at en eller flere databaser og Internettapplikasjonen kan ligge atskilt hvor som helst i verden så lenge de er koblet til Internett.
Prototypen ble laget slik at klientapplikasjonen opprettet en tekstvariabel med IP-adresse og portnummer samt katalog og metodenavn til der tjenesten lå, og man kunne sette opp så mange oppkoblinger man ville mot ulike Web Services. Figur 6.2 illustrerer kommunikasjonen mellom klient og tjener

I min løsning koblet klienten seg opp mot en applikasjon med samme database som i prototyp 1, men oppkoblingen gikk over SOAP protokollen. Ved hjelp av et program som var inkludert i AXIS bestemte jeg hvilke metoder på tjeneren som skulle være tilgjengelig over Internett

over SOAP.

Dette programmerte jeg på Web Services tjeneren:
I min løsning var det kun to metoder som var tilgjengelig fra Internett, og disse videresendte alle databasekall ved å sende metodekallet videre til interne metoder i Java klassen. Det er verdt å merke at løsningen kun besto av èn Java klasse, og denne hadde som oppgave å returnere XML eller en array(valgfritt) til klienten.

De to metodene i Java applikasjon gjorde jeg tilgjengelig for Internett via SOAP, og dette var metodene som ga returdataene til klienten.
Dette gjorde jeg ved å sette disse metodene public i Java komponenten og å velge at AXIS skulle tilgjengeliggjøre de.

AXIS genererte automatisk Web Services grensesnitt i WSDL, og dette ble gjort automatisk hver gang en fil under et valgt harddisk-område under Apache Web Server fikk endelsen .jws (Java Web Service). AXIS sjekket automatisk om filen var forandret fra forrige gang den ble den kompilert, og dersom den var forandret kompilerte AXIS den på nytt.

Hvilken returtype man skal velge fra en Web Service kan være problematisk, fordi alle språk som støtter slike tjenester skal forstå returtypen.
Man bør derfor ikke sende tilbake klasser som er spesifikke for språket der tjenesten er implementert, og derfor valgte jeg å bruke to funksjoner til returnere dataene enten som en XML fil eller en array. Klienten valgte dette ved å bestemme hvilken funksjon den kalte.

Disse funksjonene het:
1) public Object chooseOperation(String SQL, int operation)
2) public Object chooseoperationXML(String SQL, int operation)

Java klassen "Object" er en superklasse for String og array.

Metodene tok to innparametere, det første var SQL spørringen , og det andre var hvilken type spørring det var, det vil si om det select, insert, update eller delete. Den første funksjonen returnerte svaret fra "select"-spørringer serialisert inn i en 1-dimensjonal array, og den andre returnerte ved "select" et Java String Object med en XML fil som var strukturert som svaret til SQL-spørringen, og ved databaseoperasjonene "insert","update" og "delete" ble responsen fra MySQL returnert. Grunnene til at jeg ikke implementerte returtypen kun som XML var at hvis klient-

en brukte PHP på Internet Information Services(Microsofts Web Server), klarte den ikke å bruke noen av modulene i PHP som krevdes for å prosessere XML. Dette er et kjent problem, og det står også i php.ini, som er initialiseringsfilen til php, at "extensions" blir deaktivert dersom man bruker IIS.

Dette betyr at klienten med IIS ikke kunne bruke ferdige .dll pakker. Det er viktig å få fram at det virker på Windows plattform hvis man bruker Apache Web Server. Av den grunn ville klienter som brukte IIS og Windows operativsystem på en lettere måte kunne hente dataene fra en 1-dimensjonal array. Internettstedet www.webservices.org anbefaler array framfor en språkspesifikk klasse som returtype da disse vil kunne leses av nesten alle klienter.

Beskrivelse av returdata fra Java klassen via SOAP

Arrayen jeg returnerte var 1-dimensjonal, og det tradisjonelle resultatsettet som er to-dimensjonalt som returneres i en SQL-spørring, ble transformert inn i denne 1-dimensjonale arrayen. Jeg hardkodet en tekstsekvens som viste at en rad var slutt. Jeg testet i databasen at denne tekstsekvensen (—-) ikke ble returnert fra databasen i andre tilfeller, og dersom det fantes en slik tekstsekvens i en record så ble den byttet om til (–*–). Denne hardkodingen var en svakhet fordi den som programmerte mot tjenesten måtte vite at denne tekstsekvensen betydde slutten på en rad, men dette gjaldt bare i tilfellene der man brukte PHP på IIS. Alle andre klienter kunne velge å motta en XML-fil, klientene som ville ha dataene i XML format mottok et Java String objekt med XML data. Måten dette ble gjort på var å serialisere metadataene og dataene i Java fra databasen som i figur 6.4.

Dette er XML-strengen fra en spørring på hotell:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<resultset>
<hotelid>1000</hotelid>
<hotelname>Regnbuen</hotelname>
<desc>excellent</desc>
<roomid> 10001 </roomid>
<nrofbeds>3</nrofbeds>
<price>960</price>
<homepage>www.hotelregnbuen.no</homepage>
```

hotelid	hotelname	desc	roomid	nr_of_ beds	price	homepage
1000	Regn-buen	Very Good	10001	3	960	www.hotelregnbuen.no
2000	Pano-rama Hotel	medium luxuary	20001	3	1060	www.panorama.no

Figur 6.3: Et standard Java resultatsett

hotelid
hotelname
desc
roomid
nr_of_beds
price
homepage

1000
regnbuen
excellent
10001
3
960
www.hotelregnbuen.no

2000
Panorama Hotel
medium luxuary
20001
3
1060
www.panorama.no

Figur 6.4: Slik så Java resultatsettet ut etter jeg la det i array

<hotelid>1000</hotelid>
<hotelname>Panarama Hotel</hotelname>
<desc>medium luxuary </desc>
<roomid> 200001 </roomid>
<nrofbeds>3</nrofbeds>
<price>1060</price>
<homepage>www.panorama.no</homepage>
</resultset>

Web Service klienten
På klientapplikasjonen brukte jeg kun PHP, og slik jeg programmerte den, ville det ikke bety noe om man mottok en XML fil eller en array med dataene, fordi begge ble lagt inn i en PHP assosiativ array. Fordelen med en assosiativ array i PHP er at den kan motta en trestruktur hvis man ønsker dette, og man kan i tillegg automatisk gjøre attributtene fra databasen, som har blitt gjort om til XML elementer, til nøkler. Dette kalles også keys i PHP. Jeg programmerte slik at nøklene i den assosiative arrayen i PHP alltid var de samme som metadataene som kom fra resultatsettet fra MySQL-databasen.

Assosiative arrays gjorde at jeg kunne bruke attributtnavnet fra databasen da jeg refererte til dataene, og disse arrayene ble laget dynamisk, det vil si at det var ingen hardkoding av kolonnenavn i den assosiative arrayen. Antall attributter eller elementer som ble returnert fra Web Servicen var derfor uviktig.

Jeg trengte to funksjoner for å legge data inn i de assosiative arrayene på klienten:

1. Èn for å gjøre om data fra XML format til assosiativ array. Denne het getXMLTree(filename). Denne funksjonen har jeg ikke laget selv, men lastet ned fra en "News Group. Denne funksjonen er meget kompleks.

2. Èn for å gjøre om fra 1-dimensjonal array til en assosiativ array. Denne funksjonen het: make_asso_array($books2).

Ved å legge dataene inn i en assosiativ array kunne jeg jobbe med de på nesten identisk måte som om jeg jobbet direkte mot et resultatsett fra MySQL. For å jobbe med dataene måtte jeg traversere den assositive arrayen ved å henvise til radnummer og kolonnenavn(som er arrayens

result_set	hotelid	hotel name	desc	roomid	nr_of_ beds	price	homepage
Innehol -der pekere til under- elemen ter	1000	Regn- buen	Very Good	10001	3	960	www.hotel regnbuen. no
	2000	Pano- rama Hotel	medium luxuary	20001	3	1060	www.panor ama.no

Figur 6.5: Den assosiative arrayen er lik et Java resultatsett, med unntak av kolonnen result_set helt til venstre. Dersom dataene kommer fra den 1-dimensjonale arrayen dannes ikke resultset-kolonnen. Denne representer rotnoden fra XML filen.

nøkkel). For eksempel fikk man den fjerde raden av attributten "id" ved: verdien = thearray[id][3][VALUE]

Transaksjoner i min prototyp:
Dette var ikke støtte for transaksjoner i denne prototypen Det kunne lages et sett med regler for bruken av Web Services, for eksempel at hver bestilling i hver database ble sett på som en separat bestilling, og dette kan kontrolleres fordi klientapplikasjonen får et svar fra hver av databasene om bestillingen har gått igjennom ved en "insert" eller "update" eller ikke. Dette kan man sjekke på SQL-svaret fra databasen som tjenesten jobber mot.

Et eksempel vil være i mitt tilfelle, der man bestiller to rom og man sender èn bestilling på to rom i to forskjellige databaser. Dersom den ene databasen ikke er tilgjengelig, vil det likevel bestilles i den andre databasen, og transaksjonen mot denne avsluttes. Det er fare for at èn bestilling går igjennom, mens en annen ikke gjør det. Dette kunne programmeres noen regler rundt dette, men applikasjonen ville fortsatt ikke støtte transaksjoner, da en "tilbakekalling" av dataene ville være en ny transaksjon, samt at det ikke er sikkert at man fikk tilbakekalt bestillingen fordi databasen var nede. Dette problemet ble ikke løst, og dette punktet må gås igjennom på nytt dersom systemet skulle kunne reservere via Web Services.
Inntil dette er avklart er systemet best egnet kun for å lese data, ikke oppdatere, legge til eller slette.

Det var ingen fare for noen dobbeltbooking av data i databasene fordi dette sjekkes enkeltvis i hver database med primærnøkler.

6.5 Testresultater

6.5.1 Installasjon

Web Services brukte AXIS i tillegg til TomCat applikasjonsserver, Java, MySQL og PHP, og denne prototypen hadde mange krav til filstrukturen for AXIS og TomCat. Det krevdes en rekke miljøvariabler og en classpath som var stor. I denne måtte alle .jar filer ligge, som er Java-klasser som er pakket i filer. Det enkleste å gjøre for å sette classpath i dette tilfellet, var å bruke et program som satt konfigurasjonen av alle påkrevde miljøvariabler, for eksempel ANT, men jeg lagret alt i en tekstfil som jeg startet i Windows Command vinduet.

Fordi jeg implementerte programmet som tilbød tjenesten i Java, var det en standard måte å jobbe med Java og relasjonsdatabasen. Dette var godt dokumentert, blant annet på SUNs hjemmesider.

På klienten krevdes en installasjon av "PHP Pear" som er en komponentpakke som tilbyr et SOAP grensesnitt og oversetter SOAP konvolutter til ren tekst.

6.5.2 Implementasjon

Datastruktur og databaseoperasjoner på Web Services server:
Når man jobber med Web Services, er tanken at Web Services ikke er en del av programmet, og det skal derfor ikke påvirke programmeringen i stor grad. I starten av implementasjon ville jeg bruke to Java-klasser, men disse fikk jeg ikke til å kommunisere med hverandre når de lå under AXIS. Grunnen til dette var trolig at CLASSPATH måtte inkludere klassene jeg selv lagde, men dette prøvde jeg ikke. Jeg bruke i stedet èn Java-klasse. Å lage programmet på serversiden utover dette var medium komplekst fordi jeg skulle lage XML-filer i Java, samt serialisere et resultatsett inn i arrays. Server-delen av prototypen utover dette, var en standard løsning mellom Java og MySQL over JDBC.

Jeg lagde ikke løsning for kryptering, reliable messaging og digitale signaturer, men til dette finnes det verktøy som kan legges på Web Services. SotwareAG, som lager Tamino, har en partner, Vordel, som tilbyr et verktøy for å sikre autentisering og kryptering, samt at SoftwareAG har programvare for "reliable messaging". Dette er uansett vanskelig å implementere selv.

På Web Services klient:
På klienten var kompleksiteten middels kompleks, i den forstand at alle data som kommer fra Web Services må behandles slik at PHP kan bruke de.

Dersom man tar utgangspunkt i at løsningen som ble implementert i prototyp 1 etter min mening hadde lav til middels kompleksitet, så var klienten i prototyp 2 noe vanskeligere å implementere. Forskjellen var at jeg jobbet med assosiative arrays i stedet for PHPs API mot MySQL, og det var kun å forandre løkkene for å traversere de mottatte datasettene ved å bruke andre tester og andre kommandoer for å hente ut verdiene.

Løsningen ble implementert med medium grad av problemer på klienten og tjeneren, og ikke alle problemene i forhold til sikkerhet ble løst.

Kapittel 7

Fordeler og ulemper ved prototypene

De tre prototypene var små, men det var store forskjeller i struktur og implementasjonstid.

Installasjon
Prototyp 1 og 2 var enkle å installere, mens prototyp 3 krevde mer arbeid med å installere TomCat og sette miljøvariabler riktig. Kommunikasjon mellom Tomcat og AXIS krevde en spesifikk katalogstruktur der AXIS's kataloger skulle være under TomCats. Dette er vist i vedlegg.

Valg av Web Server
I prototyp 1 og på klienten i prototyp 2, brukte jeg Internet Information Services (IIS), men i ettertid ville jeg valgt å bruke Apache på disse også. Grunnen til dette er at PHP pakken XML_DOM ikke virket på IIS, og følgelig fikk jeg ikke brukt XML_DOM funksjonalitet på Web Services klienten. Dette løste jeg ved å programmere at Web Services server kunne sende data som en array, og man bør derfor bruke Apache Web Server hvis man skal jobbe med XML sammen med PHP. SoftwareAG anbefaler også bruk av Apache som Web server, framfor IIS.

7.1 Generelt om implementasjonene

7.1.1 Spørrespråkene

Den største forskjellen mellom å jobbe mot en relasjonsdatabase og en XML database var modelleringen av dataene. Fordi jeg i prototyp 2 had-

	Prototyp 1	**Prototyp 2**	**Prototyp 3**
Kodelinjer	1020, 8 filer	1664, 12 filer	1701, 9 filer
Installasjonstid	1 time	1 time	5-6 timer
Supporttjeneste	Nei	Ja	Nei
Distribuert	Nei	Nei	Ja
Transaksjoner	Nei	Ja	Nei
Teknologi moden for produksjon	Ja	Ja	Nei
Filer:Merk at disse filene ikke har samme innhold	choose.php choose2.php act1.php act2.php functions.php hot1.php choose3.php actfinish.php	choose.php choose2.php act1.php act2.php hot1.php choose3.php actfinish.php	choose.php choose2.php act1.php act2.php functions.php hot1.php choose3.php actfinish.php DAC_v2String.jws
Datafiler: XML-skjemaer		actbook.xml hotbook.xml allactivitites.xml allhotels.xml uploadact.php uploadhot.php schema.allacts.TSD schema.actbooks.TSD schema.allhotels.TSD schema.hotbooks.TSD	

Tabell 7.1: Oversikt over prototypene.

de problemer med å lage spørringer mot flere ”collections” tilsvarende SQLs "join", denormaliserte jeg dataene fra 7 tabeller i prototyp 1 og 3, til 4 XML "collections" i Tamino. Prototypen ble modellert på en måte som kun skulle dekke den brukte funksjonaliteten, men jeg vet ikke om det jeg laget var strukturmessig godt egnet til å utvide systemet med mer funksjonalitet.

Å spørre mot XML-data som lå i èn fil eller èn “collection” var overkommelig, men å strukturere flere XML-”collections” for å tilpasse det til spørrespråket syntes jeg var vanskeligere.

En annen stor forskjell mellom å jobbe mot en XML-database i forhold til en relasjonsdatabase var å bruke XQuery og filbehandling i stedet for SQL. Det var også stor forskjell på hvordan man behandlet resultatene av spørringene. Fordi man med XQuery kan spørre mot en datastruktur man ikke vet innholdet på, blir måten å jobbe med svarene av spørringene vanskeligere, fordi man med funksjoner må gå igjennom svaret dynamisk. Jeg lagret, med unntak av XSL-maler, kun strukturerte data i XML databasen, og derfor visste jeg elementnavnene i nodesettet som ble returnert fra XQuery spørringen. Jeg kunne derfor velge verdier fra elementer ved å gjøre ”if”-tester på hardkodede elementnavn eller bruke XSL stylesheet med funksjonen: xsl:value-of select="elementnavn".

Dersom man ikke kjenner datastrukturen som XQuery returnerer, er man nødt til å lese ut XML strukturen med DOM-funksjonalitet (Document Object Model), og dette er vanskeligere.

XQuery brukt i PHP var lite dokumentert med eksempler, og de fleste linkene jeg fant om XQuery pekte til W3C sitt XQuery eksempel.

Arbeidet ble også noe mer tungvint fordi det måtte defineres skjemaer i XSD for alle data som skulle lastes inn i en Tamino “collection”.

Alt i alt syntes jeg det var lettere å bruke SQL som i prototyp 1 og 3, i alle fall i påvente av at XQuery blir mer utbredt med bedre eksempler. Mesteparten av dataene jeg lagret egnet seg som relasjonsdata som kunne returneres som XML fra databasen, men unntaket her var XSL-filene.

7.1.2 Responstider

Responstidene ved bruk av XML er ofte et tema. Det er kjent at XML krever mer maskinvare-ressurser som prosessorkraft og minne. All prosessering av XML data ble foretatt på serveren der Tamino var installert, og det ble således kun sendt HTML-koder til klientene.

Taminos responstider på serveren var noe tregere enn MySQL, men alle prototypene hadde responstider på under et sekund. Med en så liten datamengde som jeg hadde, var det ikke mulig å stressteste systemet. Tamino indekserer dataene som lagres, og ved at den lagrer XML i "native format", er den optimalt raskt i forhold til hva man kan forvente for behandling av XML data. I tillegg velger man størrelsen på databasen og om det er mange skriveoperasjoner i databasen, og her er det viktig at man velger riktig.

Maskinkravene til Tamino var høyere enn MySQL, da Tamino burde ha 700 Mhz prosessor og 512 MB RAM, mens MySQL har langt lavere maskinvarekrav.

Jeg syntes det mest interessante var å finne ut hvordan responstidene i Tamino skalerte med store datamengder og et høyt antall XSL transformasjoner med mange samtidige brukere, men jeg fikk ikke undersøkt dette fordi jeg var den eneste som brukte systemet, samt at jeg ikke lagde noen applikasjon for å stressteste systemet.

7.2 Hva oppnådde jeg i de tre løsningene

7.2.1 Prototyp 1

Prototyp 1 ble implementert uten store problemer. Denne teknologien er mye utbredt, og PHP er et meget logisk språk med enkel bruk av variabler og mange ferdiglagde funksjoner.

Det er svært viktig å fremheve at jeg uttaler meg om PHP og MySQL kun innenfor denne oppgavens problemstilling. PHP er meget stort, og jeg gjorde et oppslag på de dokumenterte funksjonene på den offisielle siden www.php.net og fant cirka 5000 tusen dokumenterte funksjoner. Jeg brukte anslagsvis 50 funksjoner og brukte således 1 prosent av funksjonaliteten i PHP, og min evaluering av PHP gjelder kun det jeg brukte.

Til min løsning fungerte bruken av PHP svært bra.

7.2.2 Prototyp 2

De reserverte hotellrommene ble fjernet fra listen over de tilgjengelige i det de ble reservert. Dette gjorde jeg ved å bruke XQuery til å skille på at dersom hotellet lå i Tamino i samlingen av reserverte rom, ble det ikke vist.

I denne prototypen bør det gjøres et grundig arbeide i å modellere applikasjonen med tanke på om man vil lagre alt i XML, eller om man vil lagre dataene man vet strukturen på, i en relasjonsdatabase. Tamino er laget for å støtte en slik databaseintegrasjon, og den kan jobbe mot en relasjonsdatabase via ODBC.

Min anbefaling til prototyp 2 er å dele opp applikasjonen for å bestemme hva som skal lagres av strukturerte data og skille dette fra hvilke XSLT transformasjoner man vil lagre en XML-database. Mesteparten av dataene i denne applikasjon er lettest å lagre i en relasjonsdatabase, og jeg ville valgt å lagre de strukturerte dataene i en relasjonsdatabase og de XSL-baserte i en XML-database. Relasjonsdatabaser kan returnere svarene fra SQL-spørringene som XML, og det ville muliggjøre den samme bruken av XML og XSL. På denne måten skiller man sterkt på applikasjonsnivå mellom applikasjonslogikk og presentasjon av data, og på databasenivå sterkt mellom lagring av strukturerte og ikke-strukturerte data.

Det essensielle er at XSL-filene ikke kan lagres i en relasjonsdatabase annet enn som BLOBs (Binary Large OBjects), men dette er ikke like fleksibelt. Jeg vil anbefale å bruke en XML database til å lagre XSL-data.

7.2.3 Prototyp 3

Prototyp 3 skiller seg for sluttbrukeren ikke fra 1 og 2, men denne løsningen er mangelfull med tanke på sikkerhet. Løsningen må derfor evalueres av personer som har høy kompetanse innen sikkerhet innen Web Services og det må trolig kjøpes et verktøy for å oppnå krypter-

ing, autentisering og reliable messaging. SoftwareAG vil være en naturlig kompetansepartner da disse tilbyr disse løsningene enten selv eller via sine partnere.

Jeg vil likevel fremheve det store potensialet i Web Services, der man forespeiler seg distribuert programmering og samarbeid på applikasjonsnivå over Internett.

Kapittel 8

Konklusjon og arbeid videre

8.1 Konklusjon

I denne oppgaven arbeidet jeg med tre arkitekturer der to av de innebar noen forholdsvis nye problemstillinger.

Jeg mener at prototyp 1 var den enkleste av de tre å implementere, og dersom man skal ha en applikasjon der man kun arbeider med strukturerte data og man samler hele applikasjonen sentralt på ett sted, er PHP og MySQL det beste valget av de tre. Fordelene ved å velge en utbredt teknologi er stor, og samtidig holder man i denne prototypen applikasjonslogikken samlet i et ikke-distribuert system.

Prototyp 2 åpnet for nye muligheter innen gjenbruk av presentasjonsmaler, og dette er i tråd med "Content Management" tankegang der man tilbyr data til presentasjon i mange formater. Dette forenkler flere problemer som å tilby data til mange formater som for eksempel håndholdte PC-er, PDF-format eller XHTML.

Å bruke en XML database i prototyp 2 til hele applikasjonen bryter med tankegangen om at strukturerte data bør ligge relasjonsdatabaser, og av den grunn bør man vurdere å flytte de strukturerte dataene inn i en relasjonsdatabase. Dersom man vil lagre dokument-sentriske data er det en mulig løsning å lagre XML data i en XML database og relasjonsbaserte data i en relasjonsdatabase. Tamino støtter en slik måte å arbeide på, men i denne oppgaven ville jeg klargjøre hvordan man kunne bruke enkle databaseoperasjoner i en XML database, og derfor lagret jeg alle data i denne. I mitt arbeid ble jeg overbevist om at XML er en måte å

arbeide med data på som har en stor nytte, og XML databasen jeg jobbet med hadde et bedre og enklere administrasjonsgrensesnitt enn alle relasjonsdatabaser jeg har jobbet med.

Vanskelighetene med å jobbe med XML i prototyp 2, var å modellere XML-strukturen slik den skulle være i databasen samt å lage spørringer, men når XQuery blir mer utbredt og bedre dokumentert vil det kanskje vise seg at dette ikke er vanskeligere enn å bruke SQL mot relasjonsdatabaser.

I prototyp 2 var den største gevinsten å kunne lagre XSL-filer som maler som skrev ut data fra databasen. Å lage XHTML sider på denne måten tror jeg er lite utforsket, men har potensial til å øke i utbredelse. Årsaken til dette mener jeg er tydeligere kodenormalisering og forenklet vedlikehold av datavisning ved å ha det færre steder i applikasjonen.

Jeg erfarte at det var naturlig å bruke maler for å vise data, og disse var uavhengige av applikasjonen inntil de ble hentet fra PHP/HTML-siden som skulle bruke de.

Kompleksiteten øker på enkelte områder når man bruker XML teknologi, men XSL-maler muliggjør en ryddigere struktur. En slik deling av presentasjon og applikasjonskode er i mange situasjoner lettere å vedlikeholde enn løsninger der mye applikasjonslogikk er relatert til visning, fordi denne visningen må i sistnevnte tilfelle vedlikeholdes på mange HTML-sider.

Sett fra et overordnet ståsted er det en stor fordel å kunne skille data og innhold, men man må også ta med i betraktningene at folk ikke ønsker dette dersom det blir for komplekst å begynne å bruke det. En for stor kompleksitet har ført til at teknologier har tapt marked før, og det vil skje igjen, og dette er en viktig utfordring for XML basert lagring.

I min prototyp 2 vil jeg fremheve at gevinsten ved XSL-tranformasjoner i maler ikke ble demonstrert på en bra nok måte, men potensialet i dette bør absolutt undersøkes videre rundt utviklingen av en e-business løsning for ANP. Mine prototyper var for små til å tydeliggjøre forenklingene som XSL-maler muliggjør, og her bør arbeidsprosessene hos de deltagende bedriftene gjennomgås for å spesifisere hvor man kan bruke XSL-maler.

Prototyp 3 med Web Services støttet ikke transaksjoner og dette er en av de viktigste grunnene til den usikkerheten rundt bruken av Web Services. Web Services egner seg derfor best til å lese distribuerte data, og dette er en enkel måte å hente data fra mange Internettsteder på. Oppdatering av databaser via Web Services er ikke å anbefale enda, da dette ikke vil være pålitelig på grunn av manglende støtte for transaksjoner. Web Services er enkel og bra måte å hente data på, og man bør følge med på teknologien rundt dette i tiden framover, da det forskes mye på å løse problemet innen transaksjoner, og når dette løses kan man forsøke å legge det til på prototypen som jeg lagde.

Om ikke mange år vil systemer i .Net, Oracle, MySQL og SQL Server trolig kommunisere over Internett med transaksjoner, og dette vil forandre måten å programmere på Internett på som man har ventet lenge på, men som dessverre ikke er fullgodt enda.

I denne oppgaven har jeg konsentrert meg mye om å få prototypene til å virke med de forskjellige teknologiene fordi jeg mente dette var det første skrittet på veien mot å lage applikasjonene, og at når det mest grunnleggende virket bra kunne man begynne å spesifisere kravene ytterligere.

Jeg mener at prototyp 1, som er velutprøvd, og prototyp 2, som hadde det største potensialet innen nyskapning, begge er prototyper som kan velges til en e-handelsløsning, men at Web Services som er demonstrert i prototyp 3, fortsatt har for mange uløste problemer innen sikkerhet.

Min konklusjon er at dersom skal velge mellom prototyp 1, 2 eller 3, bør man velge prototyp 1, men den optimale løsningen er trolig en integrasjon mellom prototyp 1 og 2, ved at man lagrer strukturerte data i en relasjonsdatabase og semi-strukturerte data, som XSL-data, i Tamino. Dette vil etter min kjennskap gi den mest fleksible løsningen.

Etter hvert som flere og flere applikasjoner blir laget for Internett øker også størrelsen og kompleksiteten i de. Utgifter til utvikling og vedlikehold av applikasjoner stiger eksponentielt med kompleksiteten, og man må derfor benytte seg av alle de metoder som finnes for redusere unødvendig og fordyrende merarbeid. XML har bidratt med svært mye innen denne forenklingen, og det er ingen grunn til å tro at denne trenden skal snu. Å følge med på denne utviklingen, sett både i og utenfor sammenheng med distribuert programmering, vil gi et kraftig konkur-

ransefortrinn samt at det vil være til glede for både selger og kunde at kompleksiteten med å bruke Internettjenester går ned. Det er derfor viktig at Authentic Norway Project holder seg oppdatert på denne teknologien, slik at de kan være med å hevde seg i et marked der globaliseringen blir mer og mer trykkende og de største reiseselskapene tar mer og mer av markedet.

8.2 Utbedringer og arbeid videre

Videre vil jeg anbefale å se på de problemene jeg ikke klarte bra nok i prototypene. Jeg ser det som naturlig at man starter arbeidet med å undersøke en mulig integrasjon mellom Tamino og en relasjonsdatabase, og at man til prototyp 2 bør arbeide videre for å klargjøre i hvilken grad man ønsker "Content Management" funksjonalitet basert på XSL, og å skille dette ut i klare arbeidsprosesser som skal realiseres. Hvis dette ikke gjøres kan det fort bli mye arbeid med høy kompleksitet og liten gevinst. Mitt tips vil være å skille tydeligere mellom innhold og presentasjon og finner ut hva man ønsker å bruke XML, XSL og XSLT til av dette. Min erfaring er at dette krever en større modenhet om XML enn det jeg hadde for å se det fulle potensialet.

Et annet arbeid som bør gjøres til prototyp 2 er å undersøke om man bør skifte programmeringsspråk fra PHP til JSP eller ASP, da de to sistnevnte har offisiell support fra SoftwareAG, noe PHP ikke har. En annen utbedring på prototyp 2 er å få utnyttet hele XQuerys funksjonalitet med PHP.

Til prototyp 3 mener jeg man bør undersøke mulighetene for å oppnå den sikkerheten som prototypen krever. En naturlig sted å starte med dette er å kontakte SoftwareAG og deres partner Vordel.

Man må huske at teknologien i prototyp 2 og 3 kun er noen år gammel, og å sammenligne en slik teknologi med relasjonsdatabaser og sentraliserte systemer som har eksistert i over 30 år, er for tidlig til å gi noen klare konklusjoner.

Kapittel 9

Bibliografi

Alle oppgitte linker var aktive 12.februar 2004

(1) Paula Puleo from "Peppers and Rogers group":
"How Retailers are using customer insight to build competitive advantage", published 2003
http://www.1to1.com/View.aspx?DocID=22880
http://heim.ifi.uio.no/~tomerikv/hovedfag/1/Retail_WP.pdf

(2) Agrawal from "Indian Institute of management":
"Developing Relationship with Ecotourist", published 2002
dilbert.iiml.ac.in/ crm/Developing%20Relationship%20with%20Ecotourist1.doc
http://heim.ifi.uio.no/~tomerikv/hovedfag/2/DevelopingRelationshipwithEcotourist1.doc

(3) Anna Pollock from "DestiCorp":
"Dancing with the customer", published 2002
http://www.desticorp.com/whitepapers/dancing-with-the-customer.pdf
http://heim.ifi.uio.no/~tomerikv/hovedfag/3/dancing-with-the-customer.pdf

(4) Karabeg, Akkok, Holstskog, Fürst from the "University of Oslo":
"Authentic Norway Project", published 2002

(5) Jim Dickie from "CRMGuru":
"The CRM primer", published 2002

(6) Bob Thompson from "CRMGuru:
"CRM Predictions for 2003: Five Key Trends",publisert 2003
http://www.crmguru.com/features/2003a/0109bt.html
http://heim.ifi.uio.no/~tomerikv/hovedfag/6/CRMPredictions.htm

(7) Schmelzer, Vandersypen:
"XML and Web Services unleashed", publisert 2002 av SAM's Publishing

(8) Jay Chang from CRMGuru:
"The basics of CRM technology", publisert 2002
http://heim.ifi.uio.no/~tomerikv/hovedfag/8/index.htm

(9) David Reed from Harvard University:
"The Rule of the Pack", publisert 2001
Havard Business Report Reprint 1020C

(10) Anna Pollock, Leon Benjamin from "DestiCorp
"Why web services and grid computing will turn the travel industry on its head and why thats a good thing!", publisert April 2002
http://www.desticorp.com/resources.html
http://heim.ifi.uio.no/~tomerikv/hovedfag/10/web_services_tandt.doc

(11): Eksisterer ikke

(12) Anna Pollock: The Tourism Ecosystem in Transformation
http://www.desticorp.com/whitepapers/TourismEcosystemwhitepaperWTM.pdf

(13) Ole Strøm: "CRM Økt kundelojalitet og bedre lønnsomhet", publisert 2002

(14) Dino Karabeg, Naci Akkok, Reidar Holtskog og Karin Fürst: Authentic Norway Project, publisert 2002

(15) Oasis: "Security Assertion Markup Language (SAML) Version 1.1 Ratified as OASIS Standard", publisert september 2003
http://www.webservices.org/index.php/article/articleview/1181/
http://heim.ifi.uio.no/~tomerikv/hovedfag/15/WebServices.htm

(16) Oasis: "OASIS to Define Standard method of Exchanging Information Concering Security Vulnerabilities",
publisert mai 2003
http://www.webservices.org/index.php/article/articleview/983
http://heim.ifi.uio.no/~tomerikv/hovedfag/16/service.htm

(17) Eric Newcomer: "The Web services Standards Mess", publisert 2003

http://www.webservices.org/index.php/article/articleview/1202/
http://heim.ifi.uio.no/~tomerikv/hovedfag/17/service.htm

(18) Prasad Yendluri (pricipal Architect webMethods Inc):
"Web services reliable messaging", publisert august 2003
http://www.webservices.org/index.php/article/articleview/1148/
http://heim.ifi.uio.no/~tomerikv/hovedfag/18/message.htm

(19) Prasad Yendluri (pricipal Architect webMethods Inc):
"Web Services Choreography", publisert september 2003
http://www.webservices.org/index.php/article/articleview/1178
http://heim.ifi.uio.no/~tomerikv/hovedfag/19/chor.htm

(20) The World Travel and Tourism Council (WTTC):
"Croatia The impact of Travel and Tourism On Jobs And The Economy",
publisert 2003
http://www.wttc.org/publications/pdf/05finCroatia%202_03.pdf
http://heim.ifi.uio.no/~tomerikv/hovedfag/20/2003.pdf

(21) The World Travel and Tourism Council (WTTC): "Norway", publisert
2003
http://www.wttc.org/measure/PDF/Nowary.pdf
http://heim.ifi.uio.no/~tomerikv/hovedfag/21/Norway.pdf

(22) CRMBuyer.com: "http://www.crmbuyer.com/perl/story/18753.html",
publisert 2003

(23) Greenspan og Bulger: "PHP/MySQL Database Applications", publi-
sert 2000

(24) Nicolai Farges: "XML and Databases", publisert 2002
http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci857495,00.html
http://heim.ifi.uio.no/~tomerikv/hovedfag/24/xmldb.pdf

(25) Itavisen.no : "Web Services", publisert 2003
http://www.itavisen.no/art/1301597.html

(26) Eric Newcomer: "Web Services Transactions", utgitt 2002
artikkel ikke funnet igjen på www.webservices.org

(27) Perfectxml.com: "XQuery, A guided tour" publisert 2003

http://www.perfectxml.com/XQuery.asp
http://heim.ifi.uio.no/~tomerikv/hovedfag/27/xquery.htm

(28) Ronald Bourret: "XML and databases", publisert 2003
http://www.rpbourret.com/xml/XMLAndDatabases.htm
http://heim.ifi.uio.no/~tomerikv/hovedfag/28/xmldb.htm

(29) Peter Fingar og Ronald Aronica: "The Death of e, and the Birth of the Real New Economy", publisert 2001 på Meghan-Kiffer Press

(30) Dam Malks and Marina Sum:
"Developing Web Services with ebXML and SOAP An overview"
http://www.webservices.org/index.php/article/articleview/1015/
http://heim.ifi.uio.no/~tomerikv/hovedfag/30/xmldb.htm

Kapittel 10

Appendix

Merk at alle data som følger med til CDen til denne oppgaven kan lastes ned fra min hjemmeside på: www.tomerikv.net

- Autentisitet: Med autentisitet mener jeg kundens opplevelse av at noe er autentisk og ekte, og den er ikke nødvendigvis alltid en objektiv autentisitet. Dette betyr at rekonstruksjoner kan være autentiske.

- Authentic Norway Project: Prosjekt for å fremme autentisk turisme representert fra IFI av Dr. Dino Karabeg.
- ANP: Authentic Norway Project
- AXIS: Apache eXtendable Integration System
- Arkitektur: Med arkitektur i denne oppgaven mener jeg valget av databaseteknologi og programmeringsspråk.
- CRM: Customer Relationship Management.
- IETF: The Internet Engineering Task Force
- RT-industri: Reisning og turisme industri. Travel and Tourism Economy, Definert av WTTC. Dette er økonomien inkludert kun de som jobber direkte mot kundene innen reising og turisme.
- RT-økonomi: Reisning og turisme økonomi. Travel and Tourism Economy, Definert av WTTC. Dette er økonomien inkludert alle yrker som bidrar med tjenester innen reising og turisme.

- Tamino XML Server: Native XML-database produsert av SoftwareAG.
- W3C: World Wide Web Consortium
- WTTC: World Travel and Tourism Council.

  Uavhengig organisajon som arbeider med å fremme reising og turisme i alle land.