Quick Project Organizer

Oppdragsgiver
EDB ASA

Gruppe 16
Atle Nes
Magnus Kinn Solbjørg
Odd Christian Landmark
Ole Kristian Hoel
Martin Sleire Vatne

15. november 2001

Norges Teknisk Naturvitenskaplige Universitet
Fakultet for Fysikk, Informatikk og Matematikk
Institutt for Datateknikk og Informasjonsvitenskap
SIF8080 Kundestyrt Prosjekt

Forord

Dette er resultatet av et prosjektarbeid i faget 'SIF8080 Kundestyrt prosjekt' ved Institutt for Datateknikk og Informasjonsvitenskap, NTNU. Arbeidet er utført i en oppgave gitt av EDB ASA.

Qpro16 er et verktøy som skal hjelpe med en kjappere start på implementasjonsprosjekter ved å generere ulike typer kildekode av eksisterende modeller. Genereringen er generell, og bestemmes kun av valgte regler og maler som er spesifisert. Versjonen som vi her har implementert danner grunnlaget for at systemet kan taes i bruk, og for videre utvikling av regler og maler for andre språk.

Prosjektarbeidet, som har gått over 12 uker av høstsemesteret i 2001, har til tider dominert mye av dette semesteret, men vi føler at arbeidet har gitt oss mange verdifulle erfaringer som vi kan ta med oss videre.

Vi vil gjerne benytte anledningen til å takke vår kontaktperson hos oppdragsgiveren Ketil Aasarød for et godt samarbeid og det faktum at han har vært tilgjengelig igjennom hele perioden. Hovedveileder Reidar Conradi og hjelpeveileder Rune Rystad roses for den veiledning og tilbakemelding de har bidratt med i prosessen. Gruppen ønsker i tillegg å uttrykke sin takknemlighet til Arne Marius Hallum for gode råd og eksempler. Det rettes også et stort takk til Hilde Berg på Infohjørnet, Roar Aune i IDI ekspedisjonen, og Øystein Langerak fra EDB ASA for den hjelp gruppen har mottatt på den praktiske siden av prosjektgjennomføringen.

Trondheim, 15 november 2001

Atle Nes

Ole Kristian Hoel

Magnus Kinn Solbjørg

Martin Sleire Vatne

Odd Christian Landmark

Innledning

Dette dokumentet er en prosjektrapport for gruppe 16 i faget 'SIF8080 Kundestyrt prosjekt' og består av flere deldokumenter. Alle deldokumentene starter med et standard kapittel som forklarer kort hva dokumentet beskriver og hvordan det er bygd opp. Det anbefales at deldokumentene i all hovedsak leses i den rekkefølge de står oppført her.

Deldokumentene i denne rapporten er:

Prosjektdirektiv – rutiner for møtevirksomhet og rapportskriving, presentasjon av gruppemedlemmer og deres roller, presentasjon av oppgaven, fremdriftsplan for prosjektet og andre rammevilkår.

Forstudie – spesifiserer oppgaven ytterligere og presenterer ulike sentrale teknologier i tilknytning til oppgaven. Inneholder en grundig markedsundersøkelse med en grovevaluering av alle de eksisterende løsningene vi fant, etterfulgt av en presentasjon av aktuelle konstruerte løsninger. Etter en grundig evaluering og drøfting konkluder så dette dokumentet med hvilken løsning prosjektet skal forfølge videre.

Kravspesifikasjon – ulike former for overordnet systembeskrivelse, brukstilfeller, funksjonelle og ikke-funksjonelle krav til løsningen og en testplan.

Konstruksjon – definerer konseptet i den løsningen som skal konstrueres, og beviser at konseptet vil fungerer ved å implementere en enkel prototyp. Inneholder diverse systembeskrivelser, og hvordan systemet beskrevet oppfyller kravene fra kravspesifikasjonen. Definerer i tillegg modul- og integrasjonstester, og andre kvalitetssikrende metoder som skal benyttes i implementasjonsfasen.

Implementasjon – dokumenterer selve implementeringen av systemet med de endringer og valg som ble gjennomført her. Testresultatene for modul- og integrasjonstestene er også tatt med i dette dokumentet.

Systemdokumentasjon – dokumentasjon av systemet med fokus på å forenkle videre bruk og utvikling av det.

Prosjektevaluering – evaluering av prosjektarbeidet, faget, oppgaven og resultatet. Her kommer også den endelige systemtesten i forhold til de kravene som ble definert i kravspesifikasjonen.

Glossar – et oppslagsdokument som det blir referert til i rapporten, og som inneholder en ordliste med forklaring av nye og/eller vanskelige ord og uttrykk som benyttes.

Presentasjon - et eksemplar av foilsettet som ble benyttet under presentasjonen

Sammendrag

Vi lever i et samfunn hvor IT-baserte løsninger bare blir mer og mer viktig. Dette gjenspeiles i at det stadig blir utviklet mer programvare. Betydningen av å ha mekanismer for å begrense mengden av programkode som må utvikles er avgjørende for hvor raskt det kan skapes en slik IT-basert løsning. Hvor gode mekanismer et selskap har på å sette ned tidsforbruket ved programvareproduksjon er med på å bestemme hvor konkurransedyktig selskapet er. Det viser seg for eksempel at generering av kodeskjelett ofte tar unødvendig lang tid selv om dette ofte ligner en del fra prosjekt til prosjekt.

Det er her produktet vårt kommer inn i bildet. Qpro16 (Quick Project Organizer) er en generell kodegenerator. Generatoren kan ta generell XML som inndata og generere utdata basert på maler og regler for de språkene som skal behandles. I første omgang er det tenkt at man bruker et eksternt modelleringsverktøy for å lage en modell av programmet som skal skapes, for eksempel i UML ved hjelp av programvarepakkene Rational Rose, TogetherSoft Enterprise eller Select Enterprise. Denne eksporteres så til en XMI-representasjon, som er en XML-standard for beskrivelse av alle klasser og deres attributter, metoder etc. Ut i fra malene, som beskriver alt som er likt hver gang man genererer kode, og reglene, som beskriver hvordan XMIen skal representeres i malen, kan det genereres kode for den modellen som ble representert i XMI.

Før det ble bestemt å lage kodegeneratoren Qpro16 ble det foretatt en større markedsundersøkelse der det ble undersøkt mange eksisterende produkter på markedet. Det viste seg nemlig tidlig i prosjektet at det fantes flere produkter som syntes å ha den funksjonalitet som kunden etterspurte. Etter å ha vurdert eksisterende løsninger opp mot mulige løsninger endte vi opp med dødt løp mellom Rational Rose og Qpro16. Rational Rose er dyrt i innkjøp og Qpro16 er dyr å vedlikeholde. Det ble likevel valgt å satse videre på en egenutviklet løsning framfor en innkjøpt løsning. Dette fordi kunden ønsker å ligge i forkant av utviklingen og ønsker seg et veldig generelt verktøy som takler nye standarder uten at dette krever omprogrammering eller innkjøp av nye dyre programsystemer.

En av de mest sentralene og omfattende delene når det gjelder konstruksjon og implementasjon av Qpro16 kodegeneratoren har vært utarbeidelse og spesifisering av regelformatet og de ulike regelfilene. Det ble allerede tidlig i prosjektet bestemt at regelfilene skulle spesifiseres i XML, som gir en veldig god struktur spesielt for oppslag på regler. Det er primært blitt utarbeidet regler for import av XMI 1.0, eksport av EJB 2.0 og SQL kode som kan benyttes for å sette opp databasen for den gitte XMIen på en MS SQL 2000 Server. For å demonstrere at Qpro16 er laget for å støtte flere kode- og databasearkitekturer er det også blitt laget regler for å generere EJB 1.0 og SQL kode for Oracle databaser. Det har også vært eksperimentert med generering av VisualBasic baserte COM-komponenter.

Det må sies at prosjektet, i henhold til kundens krav og ønsker, har endret karakter ganske mye siden oppstart. Fra en kodegenerator for CMP EJB til en mer generell regel- og malbasert kodegenerator.

Det faglige utbyttet i faget har vært stort, og dette har helt klart vært det mest relevante faget i utdannelsen til nå i forholdt til den arbeidssituasjon som vi ender opp i når vi er ferdig utdannet.

Hittil har dette semesteret stort sett gått med til prosjektet. Det totale faktiske timeforbruket viste seg å bli noe større enn de opprinnelige planene. Dette skyldes i stor grad gruppens ønske om å implementere noe reelt som faktisk fungerte i stedet for å avgrense oppgaven mer i gjennomføringen av implementasjonsfasen. Vi har lært mye om ulike nye teknologier slik som XML/XMI og EJB, hvordan man håndterer arbeidsmengden i et større prosjekt og mye om hvordan man jobber sammen med ukjente mennesker og bygger en prosjektgruppe.

Overordnet innholdsfortegnelse for rapporten

Kundestyrt prosjekt

Høst 2001

Prosjektdirektiv

Versjon 1.23

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
04.09.2001	Limte sammen alle del dokumentene	0.1	Magnus
05.09.2001	Redigerte innhold	0.6	Magnus
06.09.2001	Fullføring av dokumentet	1.0	Martin
07.09.2001	Oppdatering, komplettering av prosjektmandat	1.01	Odd Christian
07.09.2001	Utvidelse av prosjektplanen, og mer spesifikk ansvarsfordeling i organiseringen.	1.02	Martin
09.09.2001	Oppdatert kapitlet om Kvalitetssikring, rettet noen skrivefeil.	1.03	Ole Kristian
09.09.2001	Oppdatert kapittel om Prosjektplanlegging	1.04	Atle
10.09.2001	Kalender lagt til	1.05	Martin
11.09.2001	Endret prosjektmandat i henhold til kundekommentarer	1.06	Odd Christian
17.09.2001	Integrere risikohåndtering. Legge til risikoanalyse for forstudiet.	1.07	Atle
26.09.2001	Oppdatert prosjektplan i henhold til faktisk forløp	1.08	Martin
30.09.2001	Legge til risikoanalyse for kravspesifikasjonsfasen	1.09	Atle
08.10.2001	Oppdatert prosjektplan i henhold til faktisk forløp	1.10	Martin
09.10.2001	Legge til risikoanalyse for konstruksjonsfasen.	1.11	Atle
19.10.2001	Endring av layout	1.12	Ole Kristian
30.10.2001	Legge til risikoanalyse for implementasjonsfasen	1.13	Atle
07.11.2001	Legge til risikoanalyse for test- og vurderingsfasen	1.20	Atle
08.11.2001	Gjennomlesning med småretting	1.22	Martin
12.11.2001	Korrekturlesing	1.23	Atle

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dette dokumentet skal legge rammene for resten av prosjektgjennomføringen, og strukturere gruppeprosessen. Dokumentet skal legge forholdene til rette for en forutsigbar og effektiv prosess.

1.2 Avgrensning

Dokumentet omhandler kun prosjektgjennomføringen og direktiver knyttet til denne, og ikke oppgaveløsning og fordypning knyttet til oppgaven.

1.3 Spesielle definisjoner

Se [GLO]

1.4 Dokumentreferanser

[GLO] Qpro16 – Glossar

1.5 Dokumentoversikt

Dette dokumentet består av fem hoveddeler som til sammen utgjør det fullstendige prosjektdirektivet. De fem delene i rekkefølge er prosjektmandatet gitt i kapittel 2, prosjektplanen i kapittel 3, maler og standarder i kapittel 4, prosjektoppfølging i kapittel 5 og kvalitetssikring i kapittel 6. Til slutt følger de ulike vedleggene i kapittel 8.

Prosjektmandatet tar for seg mål og omfang av prosjektets oppgave. Prosjektplanen viser hvordan prosjektet skal gjennomføres i tid. Maler og standarder gir retningslinjer for hvordan dokumenter skal skrives, og hvor de skal lagres. Prosjektoppfølging gir de hverdagslige rutinene som skal følges i prosjektet. Kvalitetssikring gir metoder for å sikre kvaliteten på dokumenter og produkter.

Vedlagt dette dokumentet ligger gamle planer og risikovurderinger.

2 Prosjektmandat

Prosjektmandatet fastlegger det innledende forholdet mellom prosjektgruppen og kunden. Mest sentralt i kapittelet er målene med prosjektet og omfanget av oppgaven.

2.1 Generelt om prosjektet

Dette avsnittet beskriver valg av prosjektnavn, hvem som er oppdragsgiver og bakgrunnen for prosjektet.

2.1.1 Prosjektnavn

”QPro16”
Navnet står for Quick Project Organizer. 16 er nummeret på gruppa, og er med for å forsterke gruppeidentiteten. Hovedpoenget med prosjektet er å utvikle en løsning for å gjøre en bestemt fase i et systemutviklingsprosjekt raskere. Bakgrunnen for navnet kommer klarere fram i avsnitt 2.1.3 Bakgrunn.

2.1.2 Oppdragsgiver

Firma: EDB ASA
Kontaktperson 1: Ketil Aasarød, ketil.aasarod@edbasa.no, 93096411
Kontaktperson 2: Bo Kähler, bo.kahler@edbasa.no, 73545760

2.1.3 Bakgrunn

I prosjekter hvor man jobber med komponentbaserte løsninger, brukes mye tid i starten av implementasjonsfasen til å gjøre en del rutinepreget arbeid. I overgangen fra designspesifikasjon til implementasjon, skriver programmererne mye kode direkte ut ifra klassediagrammer og andre designdokumenter. Et eksempel er overgangen fra en UML-modell av en database til SQL-skript og Entity Enterprise Java Beans. Det er som regel en direkte avbildning mellom UML-modellen og EJB-klassene, og derfor kan dette arbeidet automatiseres. Det er også stort behov for å automatisere overgangen til andre typer komponenter, som for eksempel COM-komponenter. En slik automatisering vil spare utviklerne for mye arbeid, og vil senke kostnadene knyttet til implementasjon av komponenter.

2.2 Oppgave

Her beskrives den oppgaven som ble gitt og litt om ansvar og myndighet.

2.2.1 Beskrivelse og omfang av oppgaven

Vi skal konstruere en generell løsning for automatisk generering av komponentklasser ut fra en samling entiteter og relasjoner. Input til systemet er en datamodell av et bestemt format, og det skal genereres SQL-skript for å opprette databasetabeller og kode for klasser som gir lese/skrive-aksess til tabellene. At løsningen er generell vil si at den skal gi mulighet til å generere kode innen flere forskjellige komponentteknologier. Konstruksjonen av løsningen skal også ta høyde for generering av klasser som tar seg av forretningslogikken knyttet til det modellerte systemet. Integrering av ferdige delløsninger på markedet skal vurderes, og eventuelt iverksettes.

Implementasjonen skal ikke nødvendigvis være generell, men gi full funksjonalitet for en bestemt type komponenter. Komponentene skal bygges på Enterprise Java Beans teknologien. Implementasjonen skal få UML-modell som input, og generere SQL-skript for oppretting av databasetabeller og Container Managed Entity EJB-klasser som gir lese/skrive-aksess til tabellene.

2.2.2 Ansvar og myndighet

Oppgave
Kravspesifikasjon skal hele tiden være i henhold til kundens ønsker, og konstruksjonen skal bygge på kravspesifikasjonen.

Utviklingsressurser
Innkjøp av ressurser knyttet til systemutviklingen i prosjektet må tas opp med kunde når behovet oppstår.

2.3 Effektmål

1 Den tiden det tar å generere komponentklasser for dataaksess og forretningslogikk, skal ved bruk av det produserte systemet reduseres til 50% av tiden som brukes til dette i prosjekter i EDB ASA i dag.
2 EDB ASA skal i løpet av 2002 få tilslag på et prosjekt der en implementasjon av systemet brukes som salgsargument.

2.4 Resultatmål

For å nå effektmålene må systemet designes i dette prosjektet og en implementasjon tas i bruk i EDB ASA i løpet av 2002.

2.5 Rammebetingelser

Dette avsnittet beskriver tilgangen på datamaskiner og utviklingsverktøy, samt hvilke ressurser som kan stilles til disposisjon fra kundens side.

2.5.1 Datamaskiner

Prosjektgruppa får en datamaskin på universitetet til rådighet hele døgnet, hvor ekstra programvare kan installeres, og har ellers adgang til alle datamaskiner på skolen, uten mulighet for installering av ekstra programvare.

2.5.2 Utviklingsverktøy

Vi har mulighet til å bruke utviklingsverktøy som skolen har studentlisenser på, eller som kunden kan skaffe for oss. Kunden er villig til å hjelpe oss med dette.

2.5.3 Utviklingsmiljø

Kunden har sagt seg villig til å konfigurere og sette opp en maskin/server som skal benyttes til utvikling og testing i prosjektet.

2.5.4 Fordypningsstoff

Vi har tilgang til å låne bøker på skolens og kundens bibliotek, og kunden kan utvide biblioteket etter ønske fra prosjektgruppa.

2.5.5 Forankring hos kunde

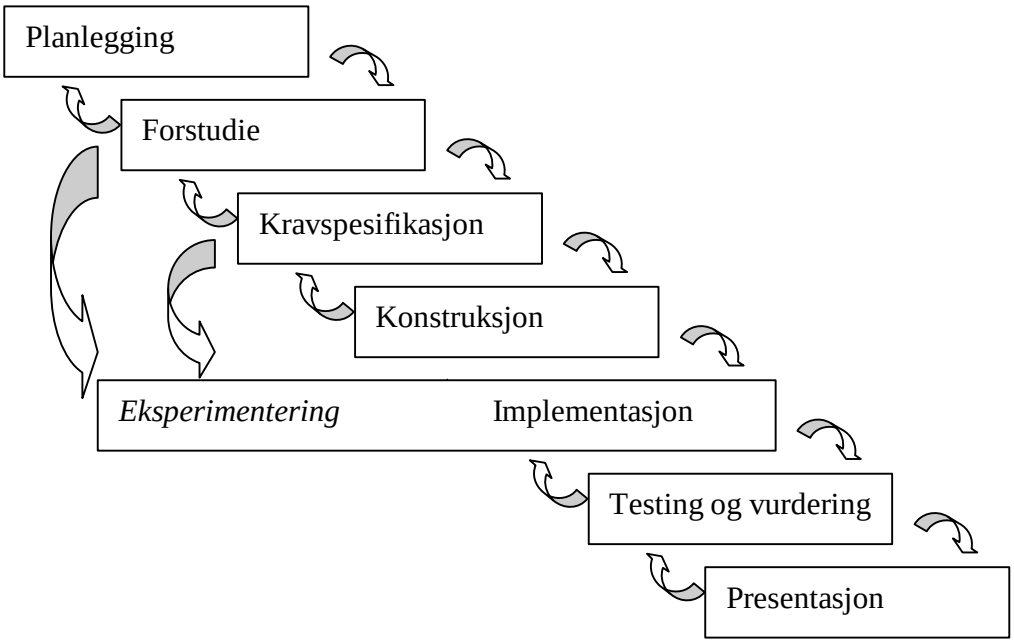
Ledelsen hos kunden, EDB ASA, er svært interessert i å få et godt utbytte av prosjektet, og kunden er derfor innstilt på å stille ressurser til disposisjon.

3 Prosjektplan

Dette kapittelet skal vise hvordan vi har planlagt at gangen i prosjektet skal være. Disse planene er utgangspunktet lagt etter skjønn og tidligere gode eksempler, og vil fungere som en veiviser etter hvert som prosjektet skrider frem.

3.1 Prosess

På grunn av prosjektets stramme tidsbudsjett har vi valgt å bruke en prosess som er basert på vannfallsmodellen da dette er den mest kjente og enkleste prosessformen for gruppens medlemmer. Fasene vi ser for oss i vår gjennomføring er planlegging, forstudie, kravspesifikasjon, konstruksjon, implementasjon, testing & vurdering, og presentasjon. Hver fase vil ha et knippe aktiviteter og dokumenter knyttet til seg, og alle fasene avsluttes med en milepæl der fasen godkjennes, og alle nødvendige endringer i tidligere faser fullføres.

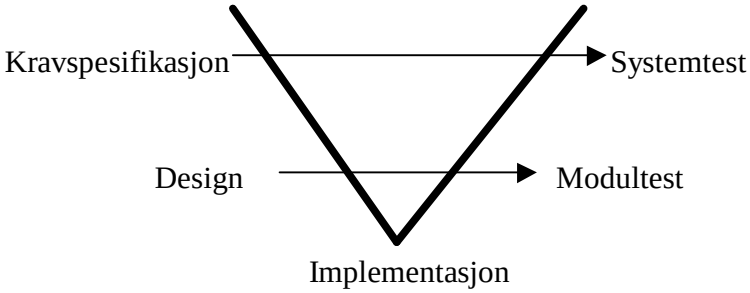


Figur 3.1 – Sekvensen av de ulike fasene

Man ser her at prosessen vi har valgt å følge er veldig lik vannfallsmodellen, bortsett fra den utbredte bruken av eksperimentering i, og parallelt med, forstudiet og kravspesifikasjonsfasen.

En mer detaljert beskrivelse av innholdet, aktivitetene og dokumentene i de ulike fasene er gitt i vedlegg 1.

Utviklingsprosessen har vi kvalitetssikret ved å planlegge testingen etter V-modellen [se figur 3.2]. Dette medfører at testing blir et gjennomgripende tema alle fasene bortsett fra forstudiet.



Figur 3.2 – V-modellen for testing

I tillegg benytter prosjektet TROKK prinsippet for risikohåndtering i alle fasene.

3.2 Estimering av tidsforbruk

I utgangspunktet er estimering av tidsforbruk på IT-prosjekter fortsatt et ungt og veldig usikkert fagfelt. De tall og planer som her blir presentert er dermed kun estimater og vil fungere som en veiledning til fremdriften i prosjektet.

I dette prosjektet regnes en arbeidsdag som 5 timer, og en arbeidsuke blir dermed 25 timer for en person. Dette gir grunnlag for følgende regnestykke:

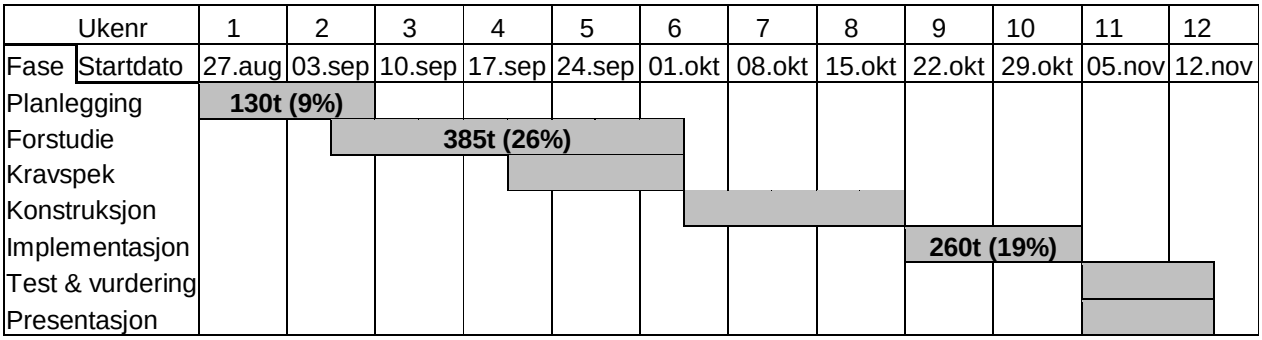
Total arbeidstid for gruppen per uke:	25?5	=	125
Antall uker til rådighet:		=	12
Totalt antall timer til rådighet:	125?12	=	**1500**

Basert på vårt innblikk i oppgavene som ligger fremfor oss, og med litt rettledning av tidligere tidsforbruk i dette faget har vi satt opp en grovfordeling av tidsforbruket i fasene som vist i tabell 3.1 på neste side.

Fase	**Tidsforbruk**	**Milepæl**
Planlegging	130 t (9%)	06.09.2001
Forstudie	385 t (25%)	03.10.2001
Kravspesifikasjon	100 t (7 %)	03.10.2001
Konstruksjon	325 t (22%)	19.10.2001
Implementasjon	290 t (19%)	02.11.2001
Testing & vurdering	90 t (6 %)	10.11.2001
Presentasjon	70 t (5%)	15.11.2001
Samlet sum	*1390 t*	
Forelesninger	110 t (7%)	
Samlet sum	*1500 t*	

Tabell 3.1 – De ulike fasene med tidsforbruk og milepæler

Estimatene kan også representeres som et overordnet gantt-diagram som i figur 3.3.



Figur 3.3 – Et overordnet gantt-diagram

Grunnlaget for de grovestimatene og milepælene som opprinnelig var gitt her er et mer spesifisert gantt-diagram gitt i prosjektkalenderen gitt i vedlegg 3.

3.3 Organisering

En oversikt over prosjektmedlemmene med kontaktinformasjon er gitt i vedlegg 2.
I prosjektet har vi definert 5 administrative roller som skal løpe igjennom hele prosjektets levetid. I tillegg har vi definert 2 spesifikke roller som kun eksisterer i implementasjon og testing & vurderingsfasen.

Her følger en definisjon av de ulike rollene med oppgaver og overordnede dokumentansvar, samt tildelingen av rollene til prosjektets medlemmer. Se vedlegg 1 for dokumentene sine plasseringer i de ulike fasene.

3.3.1 Prosjektleder

Hva: Organisere og lede møtene, holde oversikt over prosjektet samt alle administrative og personellspørsmål. Har det overordnede ansvaret for at prosjektet kommer i mål og gjennomføringen av de ulike fasene.
Dokument: Hovedansvarlig for *prosjektdirektivet* og *prosjektevalueringen*, samt innholdet i *implementasjonsdokumentet.*
Hvem: Martin Sleire Vatne

3.3.2 Ambassadør

Hva: Stå for all kontakt ut mot kunden. Skal lede møtene dersom prosjektleder ikke er tilstede. Har ansvaret for at kravspesifikasjonen er i samsvar med kundens faktiske ønsker og behov.
Dokument: Hovedansvarlig for *kravspesifikasjonen* og *presentasjonen*.
Hvem: Odd Christian Landmark

3.3.3 Teknisk ansvarlig

Hva: Ansvarlig for maler, standarder, dokumenter, versjonskontroll, felles lager og alle andre tekniske spørsmål. Har hovedansvaret for valg av teknisk arkitektur og konstruksjon.
Dokument: Hovedansvaret for *glossar* og *konstruksjonsdokumentet*.
Hvem: Magnus Kinn Solbjørg

3.3.4 Økonomiansvarlig

Hva: Ansvarlig for å innhente timerapporter og holde regnskap på disse, samt alle økonomisk relaterte spørsmål, herunder markeds- og kost/nytte-evalueringer.
Dokument: Hovedansvaret for *forstudiet.*
Hvem: Ole Kristian Hoel

3.3.5 Kvalitetsansvarlig

Hva: Ansvarlig for innsamling av krav og utforming av et rimelig test sett for systemet basert på kravene, og testfasen i prosjektet. Har også ansvaret for risikovurderingen igjennom prosjektet.
Dokument: Hovedansvaret for *risikohåndtering*, *systemtestplan* og *testrapporten.*
Hvem: Atle Nes

3.3.6 Programmerer (implementasjonsfasen)

Hva: Ansvaret for implementasjonen av en eller flere enheter i systemet.
Dokument: [systemenhet]
Hvem: Alle

3.3.7 Tester (implementasjon-, testing- og vurderingsfasen)

Hva: Ansvaret for testingen av en eller flere enheter i systemet som vedkommende ikke har laget selv i henhold til enhetstestene utarbeidet i implementasjonsfasen. Har også ansvaret for å teste en modul i systemet i henhold til testene utarbeidet i konstruksjonsfasen
Dokument: [enhetstest] og [modultest]
Hvem: Alle

4 Maler og standarder

Alle dokumenter som produseres i prosjektet skal utformes og lagres på en gitt måte. En beskrivelse av hvordan dette skal gjøres følger i dette kapitlet.

4.1 Bruk av verktøy

For å skrive dokumentasjon skal Microsoft Word benyttes og gitte maler benyttes. Maler er utarbeidet og vil til en hver tid være tilgjengelig på gruppas fellesområde. De endelige dokumentene vil også bli gjort om til PDF. Møteinnkallinger skjer per mail, men skal også lagres på fellesområdet som tekstdokumenter. Til diagrammer, koding, timeregistrering mm benyttes det til en hver tid mest egnede verktøy.

4.2 Retningslinjer for dokumenter

Dette avsnittet beskriver retningslinjer som gjelder ved skriving av dokumenter. Ellers er det lagt opp til at hvis man støter på et vanskelig ord skal dette beskrives første gang det benyttes. Det henvises forøvrig til glossar som inneholder forklaringer på de fleste uttrykk og forkortelser som benyttes i rapporten.

4.2.1 Språk

Norsk skal benyttes i alle dokumenter. Unntaket er i programmeringsarbeid. Der skal engelsk benyttes, både til navngivning av variabler, metoder mm, og kommentarer. Vedlegg på engelsk er også tillatt.

4.2.2 Kildehenvisninger

Ved kildehenvisninger føres ”[navn på kilde]” inn i teksten der henvisningen skal være. Man fører så opp forfatter(e), tittel, utgitt av og utgitt år bakerst i dokumentet under litteratur.

4.2.3 Fotnoter

Fotnoter, som kan være for eksempel forklaring av fremmedord, henvises i teksten med ”(fotnotenummer)” i hevet tekst, og beskrives på bunnen av siden.

4.2.4 Referanser

Referering til tabeller, figurer, interne kapitler, krav og lignende skal skrives etter følgende mal: *(refereringstekst).* Eks: *(se kap.4, teknologier)*

4.3 Lagring/Innlevering

Dette avsnittet omhandler rutiner for lagring av møteinnkallinger, møtereferater, statusrapporter og fasedokumenter, samt hvordan alt skal syes sammen til slutt.

4.3.1 Møteinnkallinger og møtereferater

Møteinnkallinger og godkjente møtereferater lagres fortløpende på fellesområdet av den de er skrevet av. Katalogene som skal brukes er moteinnkallinger og motereferater.

4.3.2 Statusrapporter og fasedokumenter

Ved innlevering av statusrapporter og fasedokumenter skal alle deldokumenter lagres i katalogen innlevering_sammensetning, for så å bli satt sammen av teknisk ansvarlig.

4.4 Navngivning av filer

Navnet på møteinnkallinger og møtereferater skal være henholdsvis moteinnkalling_”motedato”_”initialer”.txt og motereferat_”motedato”_”initialer”.doc. For å få versjonskontroll lagres statusrapporter, fasedokumenter og andre dokumenter som endres ved å gi dem filnavn på formen: ”beskrivende navn”_vXXX_”initialer”.doc.

beskrivende navn : beskrivende navn på dokumentet, for eksempel prosjektdirektiv

vXXX : versjonen av dokumentet, for eksempel v010 for en betaversjon, eller v100 for den ferdige versjonen. Hvis dokumentet endres etter det er ”ferdig” kan man få for eksempel v110.

initialer : hindrer overlagring, for eksempel hvis to arbeider parallelt og gir dokumentet samme versjon

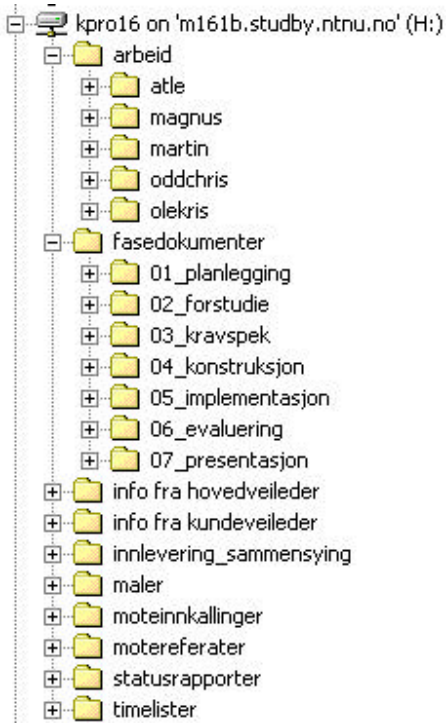
I filnavn brukes konsekvent e, o, a i stedet for æ, ø, å.

4.5 Fellesområdet

Fellesområdet vil bli liggende på ~kpro16, på m161b.studby.ntnu.no. Her vil det være en katalogstruktur for å lagre alt av dokumenter. I Windows kan man mappe opp området ved å kjøre: \\m161b.studby.ntnu.no\kpro16. Det vil også være mulig å logge seg inn med SSH. Hjemmeområdet vil selv passe på å ikke tillate at flere personer jobber på samme dokument. Det vil bli kjørt jevnlig sikkerhetskopi av hjemmeområdet over på en annen maskin. Dette vil bli utført av kvalitetsansvarlig Atle Nes som administrerer begge disse maskinene.

4.5.1 Katalogstruktur

Katalogstrukturen på fellesområdet vil se slik ut:



Figur 4.1 – Katalogstruktur for prosjektet

Katalog	**Beskrivelse**
Arbeid	Under arbeid ligger det en underkatalog for hvert av gruppemedlemmene. Denne katalogen kan brukes til å arbeide mot.
Fasedokumenter	Inneholder dokumenter i forbindelse med de forskjellige prosjektfasene.
Info fra hovedveileder	All info gruppen får fra skoleveilederne lagres i denne katalogen.
Info fra kundeveileder	All info gruppen får fra kundeveileder lagres i denne katalogen.
Innlevering_Sammensying	Her legges filer som teknisk ansvarlig skal sy sammen før innlevering.
Motereferater	Her legges alle møtereferatene.
Moteinnkallinger	Her legges møteinnkallingene.
Maler	Maler for fasedokumenter, statusrapporter, møteinnkallinger og møtereferater ligger her.
Statusrapporter	Her skal statusrapportene ligge.
Timelister	Her ligger timeregnskapet.

Tabell 4.1 – Innholdet i de ulike katalogene

5 Prosjektoppfølging

Prosjektoppfølging omhandler prosjektets møtevirksomhet. Prosjektoppfølging inneholder dessuten en felles timeplan, litt om rapportering, risikohåndtering, prosjektovervåkning og konflikthåndtering.

5.1 Møter

I dette kapittelet står det om hvordan gruppa velger møteleder og referent, samt endel om prosjektmøter, veiledermøter, kundemøter og prosjektmøter.

5.1.1 Møteleder

Gruppens prosjektleder Martin Sleire Vatne er møteleder på prosjektmøtene og gruppemøtene og skal fungere som ordstyrer på møtene. Hvis prosjektleder er borte fra prosjektmøtet overtar Odd Christian Landmark vervet som møteleder. Hvis begge disse personene er fraværende på prosjektmøtet velger man møteleder.

5.1.2 Referent

På alle veiledermøtene og kundemøtene skal en av prosjektgruppens medlemmer være referent. Denne jobben vil gå på rundgang mellom medlemmene i gruppa. Referentens oppgave er å skrive møtereferat fra møtet og maskinskrive denne før neste dag slik at den kan korrekturleses og godkjennes av resten av prosjektgruppa. Den faste malen for møtereferater skal brukes. Veilederne skal godkjenne referatene fra veiledermøtene og kunden skal godkjenne referatene fra kundemøtene.

5.1.3 Prosjektmøter

Prosjektmøtene som vi tidligere hadde hver torsdag kl.1215 er blitt flyttet til fredag kl.1015. Dette har skjedd grunnet flytting av veiledermøte til mandag. Prosjektmøtene er til for å koordinere aktiviteter, fordele oppgaver og undersøke stemningen hos medlemmene i gruppa. Et prosjektmøtene følger en fastlagt agenda beskrevet under. Det skrives en statusrapport på grunnlag av det som kommer opp på prosjektmøtet. Ansvaret for denne statusrapporten ligger hos prosjektleder.

Agenda til prosjektmøte

1. Statusrunde for hver enkelt i gruppa
2. Hva som er gjort
3. Hvor langt vedkommende er kommet
4. Når han regner med å være ferdig
5. Om han synes det han holder på med er kjedelig og eventuelt ønsker å bytte arbeid
6. Timeregnskap legges fram av timemann
7. Revisjon av plan
8. Planlegging av neste ukes aktiviteter
9. Intern godkjenning av dokumenter
10. Eventuelt

5.1.4 Veiledermøter

Veiledermøter ble først holdt hver fredag kl.1015, men ble siden flyttet til mandag kl.1815 og senere til kl.1615 samme dag. Dette møtet varer vanligvis 45 minutter. Prosjektleder har ansvaret for å reservere rom og sende ut møteinnkalling til prosjektgruppa, hovedveileder og hjelpeveileder. Prosjektgruppa bestemmer selv hva som skal diskuteres på disse møtene. Dette er gruppens mulighet til å rapportere fremdrift og status for prosjektet, få tilbakemeldinger og avklare problemstillinger. Bakgrunnsmaterialet til møtet skal leveres hovedveileder på papir og hjelpeveileder på mail senest kl.1200 dagen før møtet. Bakgrunnsmaterialet skal inneholde statusrapporten for forrige uke.

Mal for statusrapport til veiledermøtene

1. Generelt
2. Utført arbeid i perioden
    a. Dokumenter – status på dokumentene som skal utarbeides
3. Møter
4. Aktiviteter
5. Annet
6. TROKK m/avvik og tiltak
7. Problemer - Hva hindrer framdriften eller spiser ressurser
8. Planlagt arbeid neste periode
    a. Møter
    b. Aktiviteter
    c. Annet

5.1.5 Kundemøter

Gruppa har i utgangspunktet avtalt fast møtetid med kunden hver mandag kl. 1415 i elektrobygget rom F204. Hvis det skulle være nødvendig med møter utover dette ukentlige avtales det etter behov. Gruppa har i tillegg avsatt tid fra 0915 til 1000 til

koordinering og forberedelser til det ukentlige kundemøtet. Kundekontakten Odd Christian Landmark tar seg av all kontakt med kunden, har ansvaret for reservering av rom og sender ut møteinnkalling til prosjektgruppa og kunden.

5.1.6 Gruppemøter

Gruppemøter er interne møter. Disse avtales mer uformelt mellom gruppens medlemmer. Hjelpeveileder kan innkalles ved behov. Det føres det ikke møtereferat fra gruppemøtene. Gruppa har avtalt å sitte på Rose datasalen fra kl. 1215 til 1500 på onsdager, og har etterhvert også fått tildelt en maskin på en datasal som heter Gribb.

5.2 Timeplan

Prosjektgruppa satte opp en felles timeplan for å lette arbeidet med å finne møtetider. Timeplanen nedenfor viser de fagene medlemmene i gruppa tar angitt med forkortelser for hvert av fagene.

	Mandag	Tirsdag	Onsdag	Torsdag	Fredag
08.15	*	TEK R1	*	*	VIS R10
09.15	Qpro - forberedelse	TEK R1	PVK F4	*	VIS R10
10.15	SBL F4	*	PVK F4	PVK Ø F4	Qpro A161
11.15	SBL F4	*	VIS Ø R10	PVK Ø F4	Qpro A161
12.15	TEK F1	VIS R10	Qpro Rose	*	*
13.15	TEK F1	VIS R10	Qpro Rose	*	*
14.15	Qpro - Kundemøte	*	Qpro Rose	DIA F4	*
15.15	Qpro - Kundemøte	*	*	DIA F4	*
16.15	Qpro - Veileder	*	*	*	*
17.15	Qpro - Veileder	DIA Ø F4	*	SBL Ø F4	*
18.15	Qpro – Veileder	DIA Ø F4	TEK Ø F1	SBL Ø F4	*

Tabell 5.1 – Felles timeplan for prosjektgruppen

Fagforkortelser

SBL – Statisk Bildeanalyse og Læring
TEK – Teknologiledelse
VIS – Visualisering
PVK – Programvarekvalitet
DIA – Distribuerte Intelligente Agenter

Øvinger er angitt med Ø etter fagforkortelsen. I tillegg inneholder timeplanen hvilken sal forelesningene/øvingstimene foregår. Felter markert med stjerne er avtalte tider som passer for alle på gruppa og kan benyttes til fellesarbeid i prosjektet.

5.3 Rapportering

Rapportering omhandler både timerapportering, oppgaverapportering og statusrapportering.

5.3.1 Timerapportering

Timer rapporteres til økonomisk ansvarlig Ole Kristian Hoel på mail fredag hver uke. Timerapporten skal inneholde en totaloversikt over forbrukte timer fra og med lørdag til og med fredag, samt et estimat på hvor lenge man har igjen på de oppgavene man holder på med. Standard regneark med forbrukte timer skal leveres hovedveileder mandag før kl.1200. Ole Kristian legger fram timeregnskap på gruppemøtet hver fredag. Timeregnskapet skal inneholde en oversikt over forbrukte og gjenværende timer, og hvordan prosjektet ligger an i forhold til timebudsjettet.

5.3.2 Oppgaverapportering

Status på oppgaver rapporteres til prosjektleder Martin Sleire Vatne. Dette gjelder rapporter om aktiviteter som er ferdige, om aktivitetene tar lenger tid enn tidligere antatt, forespørsler om nye oppgaver, og eventuelt andre problemer som måtte dukke opp.

5.3.3 Statusrapportering

En fullstendig statusrapportering skjer på hver prosjektmøtet hver fredag.

5.4 Risikohåndtering

Prosjektgruppa har bestemt at risikohåndtering utføres ved hjelp av TROKK før hver fase eller eventuelt når gruppa finner ut at det er ønskelig å foreta en ny risikovurdering. Risikovurderingen skal oppdateres slik at man beholder historikken i vurderingene. Dette gjøres ved å legge gammel risikovurdering som vedlegg til dette dokumentet når ny risikovurdering foretaes. Hver risiko har en person ansvarlig. Denne personen skal passe på at risikoen ikke inntreffer og at tiltakene tas i bruk hvis dette skjer. Personen har også ansvar for å ta problemer knyttet til risikoer opp med gruppa.

5.4.1 TROKK

TROKK er en liste med viktige momenter som man kan benytte seg av hvis man vil bedømme risikoer til et prosjekt. Forkortelsen TROKK er sammensatt av forbokstavene til følgende momenter.

Tid - Hvordan ligger prosjektet an rent tidsmessig

Risiko - Risikomomenter, sannsynlighet, konsekvens, tiltak m/ansvar. Risikomomenter er alt som kan få betydning for prosjektets framdrift, kvalitet, kostnader, ressurser, f.eks andre systemer som dette systemet er avhengig av, ressurstilgang, kundens innvolvering, manglende avklaringer.

Omfang

Kostnad/Timer

Kvalitet - Er det noe som gjør at produktets kvalitet må reduseres

5.4.2 Risikoanalyse

Se de enkelte vedleggene for mer informasjon om risikoer for bestemte faser i prosjektet.

Vedlegg 5 – Planleggingsfasen
Vedlegg 6 – Forstudiet
Vedlegg 7 – Kravspesifikasjonsfasen
Vedlegg 8 – Konstruksjonsfasen
Vedlegg 9 – Implementasjonsfasen
Vedlegg 10 – Test- og Vurderingfasen

5.5 Prosjektovervåkning

Ansvaret for prosjektovervåkningen vil primært ligge hos prosjektleder. Det er hans ansvar at prosjektet følger fastsatt fremdriftsplan. Dersom prosjektleder ser at prosjektfremdriften avviker fra fremdriftsplanen skal det settes i verk tiltak for å få prosjektet tilbake i rute. Konkrete tiltak kan være for eksempel å justere fordelingen av arbeidsoppgaver, pålegge økt/redusert arbeidsinnsats eller foreta en endring i fremdriftsplanen. Hvilke tiltak som skal iverksettes bestemmes i fellesskap med resten av prosjektmedarbeiderne, men det er prosjektlederens ansvar å gjennomføre de vedtatte tiltakene.

5.6 Konflikthåndtering

Ved interne konflikter er det prosjektleders ansvar å sette i verk tiltak for konflikthåndtering. Gruppemedlemmene kan drøfte interne konflikter med prosjektleder. Om problemet viser seg å være prosjektleder snakker man med Odd Christian Landmark. Dersom konflikten ikke lar seg løse internt skal den tas opp med hjelpeveileder og eventuelt hovedveileder. Eksterne konflikter gjelder konflikter med kunden. Hvis gruppen ikke finner en løsning sammen med kunden vil gruppen måtte ta dette opp med hovedveileder.

6 Kvalitetssikring

Her vil vi legge fram rutiner og arbeidsprosesser som skal sikre kvaliteten på dokumenter og produkter og forenkle prosjektadministreringen.

6.1 Responstider

Vi har avtalt følgende responstider med kunden:

?? Godkjenning av tilsendt referat fra kundemøte – **24 timer**
?? Tilbakemelding på fasedokumenter kunden ønsker å få til gjennomsyn – **48 timer**
?? Godkjenning av fasedokumenter – **48 timer**
?? Svare på spørsmål – **24 timer**
?? Skaffe til veie avtalte dokumenter o.l.– **24 timer**

24 timer vil si at dersom kunden får en forespørsel før klokken 16.00 en arbeidsdag kan han eventuelt se på det på kveldstid og svare innen klokken 16.00 neste arbeidsdag. Tilsvarende vil 48 timer si før klokken 16.00 to arbeidsdager før.

6.2 Inspisering av dokumenter

Alle dokumenter bør inspiseres av minst ett gruppemedlem i tillegg til forfatteren. Dette for å få tilbakemelding og alternative innfallsvinkler i løpet av kortest mulig tid. Fasedokumenter skal gås nøye gjennom av samtlige gruppemedlemmer og deretter skal gruppa ha en felles gjennomgang før det legges frem for godkjenning.

6.3 Godkjenning av fasedokumenter

Intern godkjenning av fasedokumenter skjer på det faste fredagsmøtet. Alle i gruppen må ha satt seg inn i dokumentet på forhånd. Dersom kunden skal involveres i godkjenningen har han 48 timer responstid.

6.4 Møteinnkalling til kunden

Møteinnkalling til kunden må sendes ut på mail innen kl.16.00 to dager før møtet, og skal inneholde tid, sted, hensikt, saksliste, forventet resultat og hvilke forberedelser som er forventet av både gruppen og kunden. Det er planlagt fast kundemøte hver mandag kl.14.15. Se forøvrig kapittelet om maler og standarder.

6.5 Møtereferater fra kundemøter

Møtereferat fra kundemøtet skal være klar til utsending kl.12.00 dagen etter et kundemøte, og før den tid være godkjent av gruppen. Kunden må så godkjenne det i løpet av neste arbeidsdag. Møtereferatet skal inneholde beslutninger, aksjoner, avklaringer o.l. som har betydning for det videre arbeidet med oppgaven.

6.6 Møteinnkalling til veileder

Møteinnkalling til veileder må sendes ut senest klokken 12.00 dagen før møtet. Møtereferat fra forrige veiledermøte skal legges ved innkallingen.

7 Indekser

7.1 Figurliste

7.2 Tabelliste

8 Vedlegg

8.1 Vedlegg 1 – Planlagt innhold i de ulike fasene

Planlegging
I denne fasen fastlegges alle administrative spørsmål knyttet til prosjektet.

Aktiviteter:	Oppstartsmøter, valg av roller, fastlegging av mandat, tidsplanlegging av prosjektet og fastlegging av maler og standarder.
Dokumenter:	Prosjektdirektiv, Glossar.
Milepæl	Intern godkjenning av prosjektdirektivet og godkjenning av prosjektmandatet fra kunden.

Forstudie
I denne formuleres problemstillingen nærmere, mulige løsninger granskes og løsningen for videre arbeid velges.

Aktiviteter:	Beskrive nåsituasjon og ønsket situasjon ved å se på ulike brukstilfeller. Fastlegge kundens kriterier for å bedømme mulige løsninger. Generere et knippe mulige løsninger og evaluere disse. Inneholder også et studie av teknologier som potensielt er nyttige i forhold til oppgaven.
Dokumenter:	Forstudie, Glossar.
Milepæl	Godkjenning av forstudie og valgt løsning fra kunden.

Kravspesifikasjon
I denne fasen struktureres kunden krav til en fullstendig kravspesifikasjon.

Aktiviteter:	Spesifisere brukstilfellene fra forstudiet. Skille mellom funksjonelle og ikke-funksjonelle krav. Prioritere kravene, og koble dem til brukstilfellene og lage tester for sjekke kravene
Dokumenter:	Kravspesifikasjon, Glossar, Systemtest.
Milepæl	Godkjenning av kravspesifikasjonen fra kunden.

Konstruksjon
I denne fasen konstrueres et system som skal oppfylle kravene spesifisert i forrige fase.

Aktiviteter:	Etablere en felles objektmodell og konstruerere et klassediagram ved hjelp av sekvensdiagrammer for de ulike brukstilfellene. Konstruere modultester og integrasjonstester
Dokumenter:	Konstruksjon, Glossar.
Milepæl	Godkjenning av konstruksjonen.

Implementasjon
I denne fasen bygges det system som man har konstruert.

Aktiviteter: Koding, enhetstester, dokumentering.
Dokumenter: Implementasjonsdokument, Systemdokumentasjon.
Milepæl Ferdig kjørende system.

Testing & vurdering

Aktiviteter: Systemtest, Prosjektevaluering.
Dokumenter: Testrapport, Prosjektevaluering.
Milepæl Alle tester bestått.

Presentasjon
Den endelige presentasjonen forberedes, og øves inn. Fasen og prosjektet avsluttes med presentasjonen av prosjektet 15. november.

8.2 Vedlegg 2 – Kontaktinformasjon til prosjektets medlemmer

Magnus Solbjørg
Hermans Krags veg 3-23, 7050 Trondheim
Telefon: 73889770 / 98644355
solbjorg@stud.ntnu.no

Atle Nes
Hermans Krags veg 20-64, 7050 Trondheim
Telefon: 73889691 / 98852760
atlen@stud.ntnu.no

Odd Christian Landmark
Guttorms gate 1, 7030 Trondheim
Telefon: 95907386
oddchris@stud.ntnu.no

Ole Kristian Hoel
Lars Onsagers veg 12 -21A, 7030 Trondheim
Telefon: 73889017 / 95285786
hoel@stud.ntnu.no

Martin Sleire Vatne
Njardarvollen 13, 7032 Trondheim
Telefon: 73937285 / 90975532
martinsl@stud.ntnu.no

Hjelpeveileder:
Rune Rystad
runerys@stud.ntnu.no

Hovedveileder:
Reidar Conradi
conradi@idi.ntnu.no

8.3 Vedlegg 3 – Prosjektkalender

Uke 1 av prosjektet

27.aug	Mandag	
28.aug	Tirsdag	Oppstartsmøte
29.aug	Onsdag	
30.aug	Torsdag	Veiledermøte
31.aug	Fredag	*Gruppeprosseser*

Uke 2 av prosjektet

03.sep	Mandag	Kundemøte
04.sep	Tirsdag	
05.sep	Onsdag	
06.sep	Torsdag	**Planlegging fullført**
07.sep	Fredag	Veiledermøte

Uke 3 av prosjektet

10.sep	Mandag	Kundemøte
11.sep	Tirsdag	
12.sep	Onsdag	
13.sep	Torsdag	Prosjektmøte *Teamutvikling*
14.sep	Fredag	Veiledermøte

Uke 4 av prosjektet

17.sep	Mandag	Kundemøte *Programvarearkitektur*
18.sep	Tirsdag	
19.sep	Onsdag	
20.sep	Torsdag	Prosjektmøte
21.sep	Fredag	Veiledermøte

Uke 5 av prosjektet

24.sep	Mandag	Kundemøte
25.sep	Tirsdag	
26.sep	Onsdag	MMI
27.sep	Torsdag	*Koding og debugging i java* Prosjektmøte
28.sep	Fredag	Veiledermøte

Uke 6 av prosjektet

01.okt	Mandag	Kundemøte
02.okt	Tirsdag	
03.okt	Onsdag	**Forstudiet & Kravspesifikasjon fullført**
04.okt	Torsdag	*Konstruksjon, programmering og evaluering* Prosjektmøte
05.okt	Fredag	Veiledermøte

Uke 7 av prosjektet

08.okt	Mandag	Kundemøte
09.okt	Tirsdag	
10.okt	Onsdag	
11.okt	Torsdag	Prosjektmøte
12.okt	Fredag	Veiledermøte

Uke 8 av prosjektet

15.okt	Mandag	Kundemøte
16.okt	Tirsdag	
17.okt	Onsdag	
18.okt	Torsdag	Prosjektmøte
19.okt	Fredag	**Konstruksjon fullført** Veiledermøte

Uke 9 av prosjektet

22.okt	Mandag	**Preleveranse** Kundemøte
23.okt	Tirsdag	Bestille tid for innbinding
24.okt	Onsdag	
25.okt	Torsdag	Prosjektmøte
26.okt	Fredag	Veiledermøte

Uke 10 av prosjektet

29.okt	Mandag	Kundemøte
30.okt	Tirsdag	
31.okt	Onsdag	
01.nov	Torsdag	Prosjektmøte
02.nov	Fredag	**Implementasjon fullført** Veiledermøte

Uke 11 av prosjektet

05.nov	Mandag	Kundemøte
06.nov	Tirsdag	
07.nov	Onsdag	
08.nov	Torsdag	Prosjektmøte
09.nov	Fredag	**Testing & vurdering fullført** Veiledermøte

Uke 12 av prosjektet

12.nov	Mandag	Kundemøte
13.nov	Tirsdag	Innbinding av dokument
14.nov	Onsdag	
15.nov	Torsdag	Presentasjon

8.4 Vedlegg 4 – Tidligere tidsestimatet i prosjektplanen

Den opprinnelige tidsplanen (konstruert 01.09.2001)

I dette prosjektet regnes en arbeidsdag som 5 timer, og en arbeidsuke blir dermed 25 timer for en person. Dette gir grunnlag for følgende regnestykke:

Total arbeidstid for gruppen per uke:	25?5	=	125
Antall uker til rådighet:		=	12
Totalt antall timer til rådighet:	125?12	=	**1500**

Basert på vårt innblikk i oppgavene som ligger fremfor oss, og med litt rettledning av tidligere tidsforbruk i dette faget har vi satt opp en grovfordeling av tidsforbruket i fasene som vist i tabell 1.

Fase	**Tidsforbruk**	**Milepæl**
Planlegging	150 t (10%)	06.09.2001
Forstudie	300 t (20%)	21.09.2001
Kravspesifikasjon	225 t (15 %)	03.10.2001
Konstruksjon	350 t (ca 23%)	19.10.2001
Implementasjon	300 t (20%)	02.11.2001
Testing & vurdering	100 t (ca 7 %)	10.11.2001
Presentasjon	75 t (5%)	15.11.2001
Samlet sum	*1500 t*	

Tabell 1. Grovfordeling av tidsbruk i de ulike fasene.

Estimatene kan også representeres som et overordnet gantt-diagram som i figur 3.

Ukenr	1	2	3	4	5	6	7	8	9	10	11	12
Fase Startdato	27.aug	03.sep	10.sep	17.sep	24.sep	01.okt	08.okt	15.okt	22.okt	29.okt	05.nov	12.nov
Planlegging	**10 %**											
Forstudie												
Kravspek					**15 %**							
Konstruksjon							**23 %**					
Implementasjon												
Test & vurdering											**7 %**	
Presentasjon											**5 %**	

Figur 3. Overordnet gantt-diagram.

Tidsplanen som ble brukt under forstudiet (konstruert 26.09.2001)

I dette prosjektet regnes en arbeidsdag som 5 timer, og en arbeidsuke blir dermed 25 timer for en person. Dette gir grunnlag for følgende regnestykke:

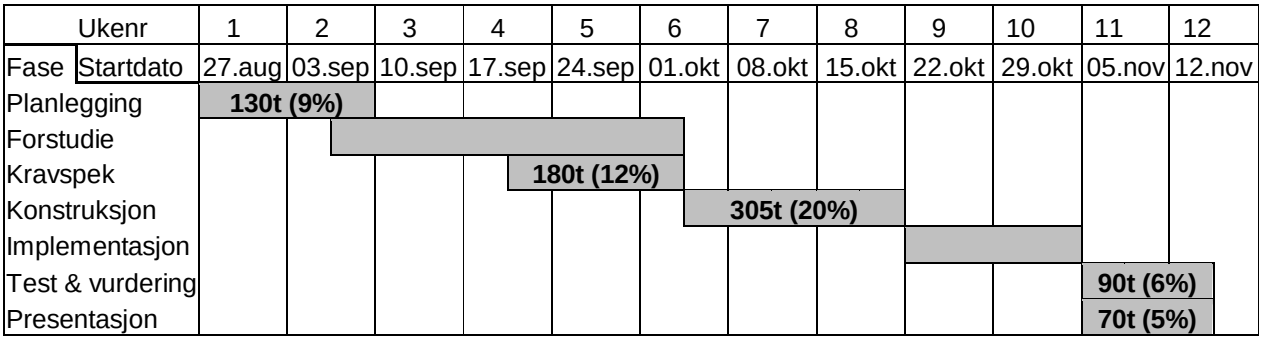
Total arbeidstid for gruppen per uke:	25?5	=	125
Antall uker til rådighet:		=	12
Totalt antall timer til rådighet:	125?12	=	**1500**

Basert på vårt innblikk i oppgavene som ligger fremfor oss, og med litt rettledning av tidligere tidsforbruk i dette faget har vi satt opp en grovfordeling av tidsforbruket i fasene som vist i tabell 1.

Fase	**Tidsforbruk**	**Milepæl**
Planlegging	130 t (9%)	06.09.2001
Forstudie	315 t (21%)	03.10.2001
Kravspesifikasjon	180 t (12 %)	03.10.2001
Konstruksjon	305 t (20%)	19.10.2001
Implementasjon	260 t (17%)	02.11.2001
Testing & vurdering	90 t (6 %)	10.11.2001
Presentasjon	70 t (5%)	15.11.2001
Samlet sum	*1350 t*	
Forelesninger	150 t (10%)	
Samlet sum	*1500 t*	

Tabell 1. Grovfordeling av tidsbruk i de ulike fasene.

Estimatene kan også representeres som et overordnet gantt-diagram som i figur 3.



Figur 3. Overordnet gantt-diagram

Grunnlaget for de grovestimatene og milepælene som er gitt her er et mer spesifisert gantt-diagram gitt i vedlegg 3 og prosjektkalenderen gitt i vedlegg 4.

8.5 Vedlegg 5 – Risikoanalyse (Planleggingsfasen)

Risikofaktor	Konsekvens	Tiltak
Tid Ettersom prosjektet har en fast sluttdato er det en fare for at det ikke vil bli tid til å få gjennomført alt det kunden ønsker. **Sannsynlighet:** Stor	I aller verste tilfelle at kunden ikke blir fornøyd med det som er blitt produsert. Det er en stor fare for at det blir for liten tid til å implementere alt som blir spesifisert.	God prosjektplanlegging og jevn arbeidsinnsats fra alle i gruppa fra starten av slik at man ikke blir hengende etter. Gruppa har avtalt med kunden at det konstrueres som om alt skal implementeres, og implementeres i henhold til en prioritert liste. **Ansvarlig:** Martin Sleire Vatne
Kunnskap Ettersom ingen av prosjektmedlemmene tidligere har jobbet noe med verken XML eller EJB kan mangel på kunnskap bli en stor risikofaktor. Alle i prosjektgruppa har gode kunnskaper om UML og Java. **Sannsynlighet:** Medium	Det kreves mer arbeid til innlæring enn hvis dette hadde vært kjent kunnskap fra før. Kvaliteten på arbeidet nedsettes om ikke gruppa får lært seg nok kunnskap i forstudiet.	Som et tiltak for å forebygge dette har alle medlemmene i prosjektgruppa fått i oppgave å lære seg så mye som mulig om XML og EJB. **Ansvarlig:** Atle Nes
Motivasjon En annen risiko kan være om en eller flere i prosjektgruppa mister motivasjonen underveis i prosjektet. **Sannsynlighet:** Liten	Lite motiverte folk fører til at innsatsen går ned og arbeidet som utføres blir vesentlig dårligere.	Prosjektleder har ansvaret for at alle i prosjektgruppa føler seg motivert. Hvis gruppa merker at et medlem begynner å bli demotivert skal man se om det f.eks. kan være aktuelt om denne personen bytter arbeidsoppgaver for å få opp motivasjonen. **Ansvarlig:** Martin Sleire Vatne

Programvare Maskinvare Gruppa trenger en maskin med weblogic server programvare installert. Det kan bli et problem både å få en dedikert maskin, samt å få konfigurert denne riktig. **Sannsynlighet:** Medium	Konsekvensen blir at gruppa får problemer med test av javabønner i et reelt servermiljø. Hvis gruppa ikke får tildelt en maskin i oppstartsfasen av prosjektet vil det bli vanskelig å nå det målet at alle prosjektmedlemmene skal lage og teste ut sin egen javabønne i løpet av forstudiet.	Gruppa har sendt inn forespørsel om å få en dedikert maskin til å installere nødvendig programvare. Kunden har sagt seg villig til å hjelpe med konfigurasjon av denne og forhåpentligvis en liten innføring i bruk. Hvis det tar lang tid å få denne maskinen er det mulig at en privat maskin kan benyttes. **Ansvarlig:** Atle Nes
Interne konflikter I løpet av prosjektet vil det helt sikkert oppstå noen uenigheter. Det trenger ikke nødvendigvis være noen stor konflikt. **Sannsynlighet:** Medium	Det kan oppstå en situasjon der det ikke blir gjort noe fordi medlemmer i gruppa er uenige og ikke får løst opp i denne uenigheten seg imellom.	Interne konflikter er en viktig grunn til at vi har prosjektmøter hvor hver enkelt medlem i gruppa skal rapportere hva som er blitt gjort, fremdrift og om han er støtt på noen problemer. **Ansvarlig:** Martin Sleire Vatne
Omfang Prosjektets omfang kan bli et risikoelement hvis prosjektgruppa ikke får klarere krav for hva som skal produseres. **Sannsynlighet:** Medium	At det implementeres et produkt som ikke samsvarer med kundens forventninger.	Gruppa skal ha et møte med kunde for å få fastsatt klare krav til produktet som skal leveres. **Ansvarlig:** Odd Christian Landmark

Overskridelse av timebudsjettet Gruppa har fått 1500 timer til rådighet fordelt på 5 personer. **Sannsynlighet:** Liten	Overskridelse av timebudsjettet vil gi mindre tid til å jobbe med andre fag.	Økonomisk ansvarlig holder kontroll med timeforbruket og kommer med en statusrapport for dette på hvert prosjektmøte. Hvis det blir brukt for mange eller for få timer på en oppgave diskuterer prosjektgruppa dette på prosjektmøtet og vil komme med løsninger for å få timebudsjettet i balanse. **Ansvarlig:** Ole Kristian Hoel
Sykdom Sykdom i gruppa er en relativt liten risikofaktor, men verd å nevne. **Sannsynlighet**: Liten	Om personene blir syke lenge kan det bli vanskeligere for dem å komme tilbake inn i prosjektet. Så lenge det ikke er flere som blir syke samtidig eller over lengre perioder så anslår vi konsekvensen som liten.	Vanskelig å gjøre noen spesielle tiltak. Magnus er ansvarlig for innkjøp av C vitaminer. Alle må trene to dager i uka og holde seg unna asiatiske gryter. **Ansvarlig:** Magnus Solbjørg

8.6 Vedlegg 6 – Risikoanalyse (Forstudiet)

Risikofaktor	Konsekvens	Tiltak
Tid Ettersom prosjektet har en fast sluttdato er det en fare for at det ikke vil bli tid til å få gjennomført alt det kunden ønsker. **Sannsynlighet:** Meget Stor	I aller verste tilfelle at kunden ikke blir fornøyd med det som er blitt produsert. Det er en stor fare for at det blir for liten tid til å implementere alt som blir spesifisert.	Flere tiltak er blitt satt i verk. Det er blitt pålagt større arbeidsinnsats for gruppemedlemmene. Planen for omfanget av hva som skal gjøres i forstudiet er blitt endret. Gruppa har avtalt med kunden at det konstrueres som om alt skal implementeres, og implementeres i henhold til en prioritert liste. **Ansvarlig:** Martin Sleire Vatne
Subjektiv evaluering Det er en fare for at evalueringen kan bli subjektiv. **Sannsynlighet:** Middels	Konsekvensen blir at vi ender opp med feil konklusjon i evalueringen og velger et dårligere produkt enn hvis evalueringen hadde vært objektiv.	Et tiltak er at selv om forskjellige medlemmer av gruppa skriver om forskjellige produkter så skal karakteren settes i fellesskap for å eliminere individuell subjektivitet. Det er dessuten opprettet maler for evaluering av produktet. **Ansvarlig:** Atle Nes
Kunnskap om produktene Det kan bli vanskelig å få tid til å skaffe seg nok kunnskap om alle de produktene som allerede finnes på markedet.. **Sannsynlighet:** Høy	Det kan være at vi overser løsninger som vi burde ha hatt med i evalueringen vår. I verste tilfelle kan vi risikere å ende opp med å velge ut et produkt som faktisk ikke er det beste.	Gruppa har i tillegg til individuell leteinnsats hatt surfedugnad hvor alle har sittet og søkt etter eksisterende løsninger på nettet og diskutert disse løsningene. Det er dessuten viktig å ikke bare lese om programmet, men også fysisk laste det ned, installere det og teste det. **Ansvarlig:** Magnus Solbjørg

Kunnskap om teknologiene Ettersom ingen av prosjektmedlemmene tidligere har jobbet noe med XML eller EJB før kan mangel på kunnskap bli en stor risikofaktor. Alle i prosjektgruppa har gode kunnskaper om UML og Java. **Sannsynlighet:** Medium	Det kreves mer arbeid til innlæring enn hvis dette hadde vært kjent kunnskap fra før. Kvaliteten på arbeidet nedsettes om ikke gruppa får lært seg nok kunnskap i forstudiet.	Alle i gruppa har fått i oppgave å lære seg så mye som mulig om EJB. Dessuten har hver av medlemmene fått hver sin teknologi. Det er ønskelig at gruppemedlemmene holder en liten presentasjon til de andre på gruppa om sin teknologi. Dokumentene er forøvrig lagt ut på fellesområdet, og er dermed leselig for alle de andre på gruppa. **Ansvarlig:** Atle Nes
Motivasjon En annen risiko kan være om en eller flere i prosjektgruppa mister motivasjonen underveis i prosjektet. **Sannsynlighet:** Medium	Lite motiverte folk fører til at innsatsen går ned og arbeidet som utføres blir vesentlig dårligere.	Hvis gruppa merker at et medlem begynner å bli demotivert skal man se om det f.eks. kan være aktuelt om denne personen bytter arbeidsoppgaver for å få opp motivasjonen. Gruppa har avtalt endel sosiale sammenkomster for å gjøre noe annet enn bare jobbe sammen. Vi forsøker også å gjøre mer praktiske tester av programvare i stedet for bare å lese. **Ansvarlig:** Martin Sleire Vatne
Programvare Maskinvare Gruppa har fått en maskin med weblogic server programvare installert. Vi har imidlertid ikke fått fulle administratorrettigheter på denne. **Sannsynlighet:** Medium	Konsekvensen blir at gruppa får problemer med å installere flere av programmene som skal testes ut i forstudiet. Kunden kan dessuten få problemer med konfigurasjon av weblogic serverprogramvaren.	Gurukontoret har blitt kontaktet og gruppa har nå administratorrettigheter på maskina. Kunden har sagt seg villig til å hjelpe med konfigurasjon av maskina og forhåpentligvis en liten innføring i bruk. **Ansvarlig:** Atle Nes & Magnus Solbjørg

Interne konflikter I løpet av prosjektet vil det helt sikkert oppstå noen uenigheter. Det trenger ikke nødvendigvis være noen stor konflikt. **Sannsynlighet:** Medium	Det kan oppstå en situasjon der det ikke blir gjort noe fordi medlemmer i gruppa er uenige og ikke får løst opp i denne uenigheten seg imellom.	Interne konflikter er en viktig grunn til at vi har prosjektmøter hvor hver enkelt medlem i gruppa skal rapportere hva som er blitt gjort, fremdrift og om han er støtt på noen problemer. Det er også derfor vi har sosiale sammenkomster hvor vi får anledning til å gjøre andre ting sammen enn bare å jobbe. **Ansvarlig:** Martin Sleire Vatne
Omfang Prosjektets omfang kan bli et risikoelement hvis prosjektgruppa ikke får klarere krav for hva som skal produseres. **Sannsynlighet:** Liten	At det implementeres et produkt som ikke samsvarer med kundens forventninger.	Gruppa har ukentlige møter med kunden. Hvis gruppa lurer på noe som har med omfanget å gjøre sender Odd Christian forespørsler til kunden. **Ansvarlig:** Odd Christian Landmark
Overskridelse av timebudsjettet Gruppa har fått 1500 timer til rådighet fordelt på 5 personer. **Sannsynlighet:** Høy	Overskridelse av timebudsjettet vil gi mindre tid til å jobbe med andre fag.	Økonomisk ansvarlig holder kontroll med timeforbruket og kommer med en statusrapport for dette på hvert prosjektmøte. Hvis det blir brukt for mange eller for få timer på en oppgave diskuterer prosjektgruppa dette på prosjektmøtet og vil komme med løsninger for å få timebudsjettet i balanse. **Ansvarlig:** Ole Kristian Hoel

Sykdom Sykdom i gruppa kan bli en risikofaktor. **Sannsynlighet**: Medium	Om personene blir syke lenge kan det bli vanskeligere for dem å komme tilbake inn i prosjektet. Så lenge det ikke er flere som blir syke samtidig eller over lengre perioder så anslår vi konsekvensen som liten.	Vanskelig å gjøre noen spesielle tiltak. Magnus er ansvarlig for innkjøp av C vitaminer. Alle må trene to dager i uka og holde seg unna asiatiske gryter. **Ansvarlig:** Magnus Solbjørg
Genererer for få/dårlige mulige løsninger Det kan bli et problem at gruppa ikke er kreative nok når det gjelder å finne fram til mulige egne løsninger. **Sannsynlighet:** Medium	Dette kan bli et problem siden kunden planlegger å bruke det vi har produsert. I verste fall blir det generert en løsning som bare må forkastes.	Vi må rådføre oss med kunden, samt ha runder i gruppa med kreativ tenking. **Ansvarlig:** Odd Christian Landmark

8.7 Vedlegg 7 – Risikoanalyse (Kravspesifikasjonsfasen)

Risikofaktor	Konsekvens	Tiltak
Tid Ettersom prosjektet har en fast sluttdato er det en fare for at det ikke vil bli tid til å få gjennomført alt det kunden ønsker. **Sannsynlighet:** Meget Stor	I aller verste tilfelle at kunden ikke blir fornøyd med det som er blitt produsert. Det er en stor fare for at det blir for liten tid til å implementere alt som blir spesifisert.	Flere tiltak er blitt satt i verk. Det er blitt pålagt større arbeidsinnsats for gruppemedlemmene. Gruppa har avtalt med kunden at det konstrueres som om alt skal implementeres, og implementeres i henhold til en prioritert liste. **Ansvarlig:** Martin Sleire Vatne
Ikke nok krav Det blir et problem for gruppa hvis det ikke dukker opp nok krav som begrenser prosjektets omfang. **Sannsynlighet:** Stor	Gruppa kan komme i skade for å produsere noe som ikke kunden kan bruke. Dette fordi bruksområdet bygger på visse krav som ikke ble definert i kravspesifikasjonsfasen.	Det beste som kan gjøres her er å diskutere krav med kunden på hvert møte. Gruppa har bedt om å få en punktliste med krav fra kunden. **Ansvarlig:** Odd Christian Landmark
Misforstår krav Det er en liten fare for at gruppa og kunden misforstår hverandre og definerer de ulike kravene forskjellig. Faren er liten ettersom dette er krav som til slutt skal godkjennes av kunden. **Sannsynlighet:** Liten	Dette kan føre til at det implementeres et produkt som ikke samsvarer med kundens krav.	Som tiltak har gruppa valgt å bruke brukstilfeller som øker forståelsen mellom gruppa og kunden. Dessuten har gruppa ytret at de ønsker spesifiserte og målbare krav med lite tvetydige spissformuleringer. **Ansvarlig:** Atle Nes

Ikke dekkende testplan Det er viktig at alle kravene i denne fasen er dekket i en testplan. Det er en fare for at testplanen blir for subjektivt lagt opp. **Sannsynlighet:** Medium	Hvis ikke alle kravene er dekket kan det medføre at visse krav aldri blir testet før kunden får sitt produkt. Det kan da hende at nettopp disse kravene enten ikke er oppfylt eller er feil implementert.	Det største tiltaket er først og fremst at vi ikke lager testplanen før mot slutten av kravspesifikasjonsfasen. Dette gjør at vi kan gå tilbake og se på alle kravene og at disse er dekket i testplanen. **Ansvarlig:** Atle Nes
Prioritering av krav Det er en fare for at vi lager krav som ikke lar seg implementere. Derfor har vi valgt å prioritere kravene. Da er det igjen en fare for at vi prioriterer kravene i gal rekkefølge. **Sannsynlighet:** Medium	Dette medfører videre at vi lager krav som ikke lar seg implementere.	En som ikke jobber direkte med kravene slik som prosjektleder ser over at kravene er riktig prioritert. **Ansvarlig:** Martin Sleire Vatne
Kunnskap om teknologiene Ettersom ingen av prosjektmedlemmene tidligere har jobbet noe med XML eller EJB før kan mangel på kunnskap bli en stor risikofaktor. Alle i prosjektgruppa har gode kunnskaper om UML og Java. **Sannsynlighet:** Medium	Det kreves mer arbeid til innlæring enn hvis dette hadde vært kjent kunnskap fra før. Kvaliteten på arbeidet nedsettes om ikke gruppa får lært seg nok kunnskap i forstudiet.	Alle i gruppa har fått i oppgave å lære seg så mye som mulig om EJB. Dessuten har hver av medlemmene fått hver sin teknologi. Det er ønskelig at gruppemedlemmene holder en liten presentasjon til de andre på gruppa om sin teknologi. Dokumentene er forøvrig lagt ut på fellesområdet, og er dermed leselig for alle de andre på gruppa. **Ansvarlig:** Atle Nes
Motivasjon En annen risiko kan være om en eller flere i prosjektgruppa mister motivasjonen underveis i prosjektet. **Sannsynlighet:** Medium	Lite motiverte folk fører til at innsatsen går ned og arbeidet som utføres blir vesentlig dårligere.	Hvis gruppa merker at et medlem begynner å bli demotivert skal man se om det for eksempel kan være aktuelt om denne personen bytter arbeidsoppgaver for å få opp motivasjonen. **Ansvarlig:** Martin Sleire Vatne

Interne konflikter I løpet av prosjektet vil det helt sikkert oppstå noen uenigheter. Det trenger ikke nødvendigvis være noen stor konflikt. **Sannsynlighet:** Medium	Det kan oppstå en situasjon der det ikke blir gjort noe fordi medlemmer i gruppa er uenige og ikke får løst opp i denne uenigheten seg imellom.	Interne konflikter er en viktig grunn til at vi har prosjektmøter hvor hver enkelt medlem i gruppa skal rapportere hva som er blitt gjort, fremdrift og om han er støtt på noen problemer. Det er også derfor vi har sosiale sammenkomster hvor vi får anledning til å gjøre andre ting sammen enn bare å jobbe. **Ansvarlig:** Martin Sleire Vatne
Overskridelse av timebudsjettet Gruppa har fått 1500 timer til rådighet fordelt på 5 personer. **Sannsynlighet:** Stor	Overskridelse av timebudsjettet vil gi mindre tid til å jobbe med andre fag.	Økonomisk ansvarlig holder kontroll med timeforbruket og kommer med en statusrapport for dette på hvert prosjektmøte. Hvis det blir brukt for mange eller for få timer på en oppgave diskuterer prosjektgruppa dette på prosjektmøtet og vil komme med løsninger for å få timebudsjettet i balanse. **Ansvarlig:** Ole Kristian Hoel
Sykdom Sykdom i gruppa kan bli en risikofaktor. **Sannsynlighet**: Medium	Om personene blir syke lenge kan det bli vanskeligere for dem å komme tilbake inn i prosjektet. Så lenge det ikke er flere som blir syke samtidig eller over lengre perioder så anslår vi konsekvensen som liten.	Vanskelig å gjøre noen spesielle tiltak. Magnus er ansvarlig for innkjøp av C vitaminer. Alle må trene to dager i uka og holde seg unna asiatiske gryter. **Ansvarlig:** Magnus Solbjørg

8.8 Vedlegg 8 – Risikoanalyse (Konstruksjonsfasen)

Risikofaktor	Konsekvens	Tiltak
Ikke generelle nok regler og maler Det er en fare for at reglene og malene ikke blir konstruert slik at de tar høyde for alle teknologiene de er tenkt til. **Sannsynlighet:** Høy	Problemet med dette er at generatoren ikke kan benyttes til å generere kode for alle de ulike komponentteknologiene slik den er tiltenkt.	Gruppa skal sende en kopi av forslaget sitt til kunden for gjennomlesing. Et annet tiltak er å rett og slett tilegne seg mer kunnskap spesielt om XML. **Ansvarlig:** Magnus Solbjørg
Ikke generell nok parsegenerator Det er en fare for at parsegeneratoren ikke blir generell nok konstruert. **Sannsynlighet:** Høy	Dette gir samme konsekvens som ovenfor da vi ikke hadde generelle nok regler og maler.	Et av tiltakene er å lære seg så mye som mulig om EJB. Ellers få flere til å komme med innspill på konstruksjonen av en parsegenerator slik at det ikke kun blir en liten gruppe personer som gjør dette. **Ansvarlig:** Martin Sleire Vatne
Tid Ettersom prosjektet har en fast sluttdato er det en fare for at det ikke vil bli tid til å få gjennomført alt det kunden ønsker. **Sannsynlighet:** Meget Stor	I aller verste tilfelle at kunden ikke blir fornøyd med det som er blitt produsert. Det er en stor fare for at det blir for liten tid til å implementere alt som blir spesifisert.	Flere tiltak er blitt satt i verk. Det er blitt pålagt større arbeidsinnsats for gruppemedlemmene. Gruppa har avtalt med kunden at det konstrueres som om alt skal implementeres, og implementeres i henhold til en prioritert liste. Det er ingen god ide å slurve med konstruksjonen. Arbeidsuka er økt til 30 timer i uka. **Ansvarlig:** Martin Sleire Vatne

Konstruksjonen for lite utvidbart Kunden har tenkt å bruke koden videre for å implementere utvidelser etter hvert som det dukker opp nye teknologier. **Sannsynlighet:** Medium	Konsekvensen blir at implementasjonen vår rett og slett må forkastes. Vi kan komme til å lage et produkt som kunden ikke kan bruke videre.	De viktigste tiltakene blir å tilegne seg nok kunnskap om teknologiene, samt å tenke aktivt på utvidbarhet i implementasjons- og konstruksjonsfasen. **Ansvarlig:** Magnus Solbjørg
Konstruksjonen ikke fungerer i praksis Det er en risiko at vi konstruerer et system som ikke lar seg implementere. **Sannsynlighet:** Medium	Et lite gjennomtenkt konstruksjon vil gi utslag i implementasjonsfasen. Det kan være at vi må endre konstruksjonen og dermed ødelegger generalitet og funksjonalitet hos programmet.	Tiltak er drøfting i fellesskap av konstruksjonen. Dessuten har gruppa bestemt seg for å lage et "Proof of Concept" i konstruksjons-fasen. Dette skal vise at de ulike teknologiene fungerer sammen. **Ansvarlig:** Ole Kristian Hoel
Kunnskap om teknologiene Prosjektmedlemmene føler fortsatt at de ikke kan nok om de sentrale teknologiene som skal benyttes. Dette gjelder i første omgang XML eller EJB. Alle i gruppa har gode kunnskaper om UML, SQL og Java. **Sannsynlighet:** Medium	Det kreves mer arbeid til innlæring enn hvis dette hadde vært kjent kunnskap fra før. Kvaliteten på arbeidet nedsettes om ikke gruppa får lært seg nok kunnskap i forstudiet. Det er en fare for at konstruksjonen blir galt, at det ikke blir optimalt og at det blir tungvindt.	Alle i gruppa har fått i oppgave å lære seg så mye som mulig om EJB. Dessuten har hver av medlemmene fått hver sin teknologi. Alle dokumenter som gruppa har fått er forøvrig lagt ut på fellesområdet, og er dermed leselig for alle på gruppa. **Ansvarlig:** Atle Nes

Motivasjon En annen risiko kan være om en eller flere i prosjektgruppa mister motivasjonen underveis i prosjektet. **Sannsynlighet:** Medium	Lite motiverte folk fører til at innsatsen går ned og arbeidet som utføres blir vesentlig dårligere.	Hvis gruppa merker at et medlem begynner å bli demotivert skal man se om det kan være aktuelt om denne personen bytter arbeidsoppgaver for å få opp motivasjonen. Gruppa har avtalt endel sosiale sammenkomster for å gjøre noe annet enn bare jobbe sammen. **Ansvarlig:** Martin Sleire Vatne
Interne konflikter I løpet av prosjektet vil det helt sikkert oppstå noen uenigheter. Det trenger ikke nødvendigvis være noen stor konflikt. **Sannsynlighet:** Medium	Det kan oppstå en situasjon der det ikke blir gjort noe fordi medlemmer i gruppa er uenige og ikke får løst opp i denne uenigheten seg imellom.	Interne konflikter er en viktig grunn til at vi har prosjektmøter hvor hver enkelt medlem i gruppa skal rapportere hva som er blitt gjort, fremdrift og om han er støtt på noen problemer. Det er også derfor vi har sosiale sammenkomster hvor vi får anledning til å gjøre andre ting sammen enn bare å jobbe. **Ansvarlig:** Martin Sleire Vatne
Overskridelse av timebudsjettet Gruppa har fått 1500 timer til rådighet fordelt på 5 personer. **Sannsynlighet:** Medium	Overskridelse av timebudsjettet vil gi mindre tid til å jobbe med andre fag.	Økonomisk ansvarlig holder kontroll med timeforbruket og kommer med en statusrapport for dette på hvert prosjektmøte. Hvis det blir brukt for mange eller for få timer på en oppgave diskuterer gruppa det på prosjektmøtet og vil gi løsninger for å få timebudsjettet i balanse. **Ansvarlig:** Ole Kristian Hoel

Sykdom Sykdom i gruppa kan bli en risikofaktor. Denne risikoen har allerede slått til og personen det gjaldt ble sendt til lege. **Sannsynlighet**: Medium	Om personene blir syke lenge kan det bli vanskeligere for dem å komme tilbake inn i prosjektet. Så lenge det ikke er flere som blir syke samtidig eller over lengre perioder så anslår vi konsekvensen som liten.	Vanskelig å gjøre noen spesielle tiltak. Magnus er ansvarlig for innkjøp av C vitaminer. Alle må trene to dager i uka og holde seg unna asiatiske gryter. **Ansvarlig:** Magnus Solbjørg

8.9 Vedlegg 9 – Risikoanalyse (Implementasjonsfasen)

Risikofaktor	Konsekvens	Tiltak
Folk følger ikke kodestandard Gruppa antar at de ulike medlemmene har ulike programmeringsmåter og programmerings-ferdigheter. **Sannsynlighet:** Høy	Det kan bli et problem at ikke koden ser mest mulig lik ut i alle deler av programmet, og dermed gjøre det vanskeligere for utenforstående å lese koden.	Alle må lese kodestandarden, som ligger som vedlegg bakerst i implementasjons-dokumentet. **Ansvarlig:** Magnus Solbjørg
Tid Ettersom prosjektet har en fast sluttdato er det en fare for at det ikke vil bli tid til å få gjennomført alt det kunden ønsker. **Sannsynlighet:** Meget Stor	I aller verste tilfelle at kunden ikke blir fornøyd med det som er blitt produsert. Det er en stor fare for at det blir for liten tid til å implementere alt som blir spesifisert.	Flere tiltak er blitt satt i verk. Det er blitt pålagt større arbeidsinnsats for gruppemedlemmene. Gruppa har avtalt med kunden at det konstrueres som om alt skal implementeres, og implementeres i henhold til en prioritert liste. Det er ingen god ide å slurve med konstruksjonen. Arbeidsuka er økt til 40 timer i uka. Dessuten jobber gruppa nå en dag i helga. **Ansvarlig:** Martin Sleire Vatne
Konstruksjonen ikke fungerer i praksis Det er en risiko at vi konstruerer et system som ikke lar seg implementere. **Sannsynlighet:** Medium	Et lite gjennomtenkt konstruksjon vil gi utslag i implementasjonsfasen. Det kan være at vi må endre konstruksjonen og dermed ødelegger generalitet og funksjonalitet hos programmet.	Tiltak er drøfting i fellesskap av konstruksjonen. Dessuten har gruppa laget et ”Proof of Concept” i konstruksjonsfasen. Dette skal viste at de ulike teknologiene fungerte sammen. **Ansvarlig:** Ole Kristian Hoel

Konstruksjonen blir ikke fulgt Det kan bli et problem hvis gruppemedlemmene bruker egne ad hoc løsninger som medfører at konstruksjonen ikke blir fulgt. **Sannsynlighet:** Medium	Konsekvensen kan bli nedsatt generalitet eller tap av funksjonalitet hos kodegeneratoren.	Alle setter seg grundig inn i konstruksjonen slik at det ikke skal være uoverensstemmelser i tolkningen. Dessuten sitter vi på stort sett på samme rom under kodeprosessen, så da blir det enkelt å spørre andre hvis det skulle være noen uklarheter. **Ansvarlig:** Ole Kristian Hoel
Kunnskap om teknologiene Enkelte av prosjektmedlemmene føler fortsatt at de ikke kan alt, og sliter litt med implementasjonen, selv om de sentrale teknologiene som skal benyttes nå er kjent. **Sannsynlighet:** Liten	Det kreves mer arbeid til innlæring enn hvis dette hadde vært kjent kunnskap fra før. Kvaliteten på arbeidet nedsettes. Det er en fare for at implementasjonen blir noe tungvind og i verste fall ikke kommer i mål.	Tiltak er god kommunikasjon mellom medlemmene så informasjon kan flyte fra de som kan mye til de som ikke kan så mye. Hvis noen lurer på noe sendes dette på mail til de andre. Eventuelt kan kunde spørres om det er noe man lurer på. **Ansvarlig:** Atle Nes
Implementasjonen blir ikke testbar Det kan bli et problem at implementasjonen blir utformet på en slik måte at det ikke enkelt lar seg teste. **Sannsynlighet:** Medium	Det kan bli vanskelig og i verste fall bli umulig å få testet moduler. Dette gjør igjen at feil kan bli skjult og først dukke opp ved integrasjonstest eller i verste fall etter leveranse.	Alle har fått i ansvar å skrive sin egen lille test til sin egen modul. Dette medfører at programmerer hele veien må tenke på at det han produserer skal bli testet. **Ansvarlig:** Atle Nes
Testen tester ikke alt Det kan være at ettersom testmodulene konstrueres av de samme som lager programmene så utelates momenter som burde vært testet. **Sannsynlighet:** Stor	Det kan også her bli sittende igjen ukjente bugs/feil til integrasjonen eller i verste fall etter leveranse.	Modultesten som utarbeides av hver enkelt skal i likhet med all annen kode inspiseres av en annen person som ikke har vært med på å programmere nettopp denne modulen. **Ansvarlig:** Atle Nes

Koden inneholder ukjente feil Det er en risiko for at den som programmerer ser seg blind på sin egen kode. **Sannsynlighet:** Høy	Dette medfører at det kan oppstå feil som ikke blir oppdaget verken ved programmering eller ved testen som utarbeides.	Tiltak for å slippe dette er inspeksjon av kode av en av de andre programmererne. Dette medfører at koden får en objektiv bedømming fra en som ikke har vært med på nettopp denne modulen. **Ansvarlig:** Atle Nes
Integrasjonen av moduler skjærer seg Når modultesten av alle modulene er ferdig kan det bli vanskeligheter med å sette disse modulene sammen til et større system. **Sannsynlighet:** Høy	Konsekvensen blir at implementasjonen kan ta lengre tid enn opprinnelig planlagt. I verste fall kan vi få et produkt som ikke virker i det hele tatt.	Hovedlinjene i konstruksjonen skal følges. Det blir benyttet CVS for versjonskontroll på programkoden. Samarbeid er et viktig tiltak og mail skal sendes de andre medlemmene på gruppa ved større endringer. Javadoc må oppdateres kontinuerlig. **Ansvarlig:** Odd Christian Landmark
Motivasjon En annen risiko kan være om en eller flere i prosjektgruppa mister motivasjonen underveis i prosjektet. **Sannsynlighet:** Medium	Lite motiverte folk fører til at innsatsen går ned og arbeidet som utføres blir vesentlig dårligere.	Hvis gruppa merker at et medlem begynner å bli demotivert skal man se om det kan være aktuelt om denne personen bytter arbeidsoppgaver for å få opp motivasjonen. Gruppa har avtalt endel sosiale sammenkomster for å gjøre noe annet enn bare jobbe sammen. **Ansvarlig:** Martin Sleire Vatne

Interne konflikter I løpet av prosjektet vil det helt sikkert oppstå noen uenigheter. Det trenger ikke nødvendigvis være noen stor konflikt. **Sannsynlighet:** Medium	Det kan oppstå en situasjon der det ikke blir gjort noe fordi medlemmer i gruppa er uenige og ikke får løst opp i denne uenigheten seg imellom.	Interne konflikter er en viktig grunn til at vi har prosjektmøter hvor hver enkelt medlem i gruppa skal rapportere hva som er blitt gjort, fremdrift og om han er støtt på noen problemer. Det er også derfor vi har sosiale sammenkomster hvor vi får anledning til å gjøre andre ting sammen enn bare å jobbe. **Ansvarlig:** Martin Sleire Vatne
Overskridelse av timebudsjettet Gruppa har fått 1500 timer til rådighet fordelt på 5 personer. **Sannsynlighet:** Medium	Overskridelse av timebudsjettet vil gi mindre tid til å jobbe med andre fag.	Økonomisk ansvarlig holder kontroll med timeforbruket og kommer med en statusrapport for dette på hvert prosjektmøte. Hvis det blir brukt for mange eller for få timer på en oppgave diskuterer gruppa det på prosjektmøtet og vil gi løsninger for å få timebudsjettet i balanse. **Ansvarlig:** Ole Kristian Hoel
Sykdom Sykdom i gruppa kan bli en risikofaktor. Denne risikoen har allerede slått til og personen det gjaldt ble sendt til lege. **Sannsynlighet**: Medium	Om personene blir syke lenge kan det bli vanskeligere for dem å komme tilbake inn i prosjektet. Så lenge det ikke er flere som blir syke samtidig eller over lengre perioder så anslår vi konsekvensen som liten.	Vanskelig å gjøre noen spesielle tiltak. Magnus er ansvarlig for innkjøp av C vitaminer. Alle må trene to dager i uka og holde seg unna asiatiske gryter. **Ansvarlig:** Magnus Solbjørg

8.10 Vedlegg 10 – Risikoanalyse (Test- og Vurderingsfasen)

Risikofaktor	Konsekvens	Tiltak
Tid Det er nå sikkert at det ikke vil bli tid til å få gjennomført alt før sluttdato for prosjektet 15. november 2001. **Sannsynlighet:** Meget Stor	I aller verste tilfelle at kunden ikke blir fornøyd med det som er blitt produsert. Det er en stor fare for at det blir for liten tid til å implementere alt som er blitt spesifisert, og samtidig å få dette testet og evaluert.	Flere tiltak er blitt satt i verk. Det er blitt pålagt større arbeidsinnsats for gruppemedlemmene. Arbeidsuka er økt til 40 timer i uka. Dessuten jobber gruppa nå begge dagene den siste helga for å komme i mål. **Ansvarlig:** Martin Sleire Vatne
Implementasjonen blir ikke testbar Det kan bli et problem at implementasjonen blir utformet på en slik måte at det ikke enkelt lar seg teste. **Sannsynlighet:** Medium	Det kan bli vanskelig og i verste fall bli umulig å få testet moduler. Dette gjør igjen at feil kan bli skjult og først dukke opp ved integrasjonstest eller i verste fall etter leveranse.	Alle har fått i ansvar å skrive sin egen lille test til sin egen modul. Dette medfører at programmerer hele veien må tenke på at det han produserer skal bli testet. **Ansvarlig:** Atle Nes
Testen tester ikke alt Det kan være at ettersom testmodulene konstrueres av de samme som lager programmene så utelates momenter som burde vært testet. **Sannsynlighet:** Høy	Det kan også her bli sittende igjen ukjente bugs/feil til integrasjonen eller i verste fall etter leveranse på grunn av at testen ikke er utført på en objektiv og rettferdig måte.	Modultesten som utarbeides av hver enkelt skal i likhet med all annen kode inspiseres av en annen person som ikke har vært med på å programmere nettopp denne modulen. Dette gjør at forståelsen av modulene ikke bare ligger hos den som har hovedansvaret for modulen. **Ansvarlig:** Atle Nes

Evalueringen blir ensidig Kan være et problem ettersom ikke alle på gruppa er med på å skrive på evalueringsdokumentet. **Sannsynlighet:** Medium	Konsekvensen blir at vurderingen ikke representerer hele gruppens synspunkter.	Et av tiltakene som har vært tilstede under hele prosjektet er at gruppemedlemmene har fått i oppgave å notere ned evalueringer underveis slik at de ikke blir glemt. Alle har fått anledningen til å komme innspill på hva som burde være med i evalueringen, og alle har fått i oppgave å lese igjennom evalueringsdokumentet etter hvert som det blir ferdig. **Ansvarlig:** Martin Sleire Vatne
"Feature Creep" Det kan være et problem at programmererne på prosjektgruppa blir for interessert i at alle funksjoner skal fungere. **Sannsynlighet:** Høy	Dette kan føre til at det fort går mye tid på finpuss av et produkt som fungerer tilfredsstillende nok.	Tiltak er at vi har satt en deadline for videre implementasjon 7.11.2001. **Ansvarlig:** Ole Kristian Hoel
Kodeinspeksjonen taes for lett Det kan fort bli at de som sjekker programkode tar for lett på inspeksjonen. **Sannsynlighet:** Høy	Dette kan føre til at det er programfeil som burde ha blitt oppdaget som ikke blir oppdaget.	Et av tiltakene er at inspektørene som inspiserer skal benytte seg av sjekklisten som ligger vedlagt til konstruksjonsdokumentet når koden blir inspisert. **Ansvarlig:** Atle Nes

Koden inneholder ukjente feil Det er en risiko for at den som programmerer ser seg blind på sin egen kode. **Sannsynlighet:** Høy	Dette medfører at det kan oppstå feil som ikke blir oppdaget verken ved programmering eller ved testen som utarbeides.	Tiltak for å slippe dette er inspeksjon av kode av en av de andre programmererne. Dette medfører at koden får en objektiv bedømming fra en som ikke har vært med på nettopp denne modulen. **Ansvarlig:** Atle Nes
Integrasjonen av moduler skjærer seg Når modultesten av alle modulene er ferdig kan det bli vanskeligheter med å sette disse modulene sammen til et større system. **Sannsynlighet:** Høy	Konsekvensen blir at implementasjonen kan ta lengre tid enn opprinnelig planlagt. I verste fall kan vi få et produkt som ikke virker i det hele tatt.	Hovedlinjene i konstruksjonen skal følges. Det blir benyttet CVS for versjonskontroll på programkoden. Samarbeid er et viktig tiltak og mail skal sendes de andre medlemmene på gruppa ved større endringer. Javadoc må oppdateres kontinuerlig. **Ansvarlig:** Odd Christian Landmark
Motivasjon En annen risiko kan være om en eller flere i prosjektgruppa mister motivasjonen underveis i prosjektet. **Sannsynlighet:** Medium	Lite motiverte folk fører til at innsatsen går ned og arbeidet som utføres blir vesentlig dårligere.	Hvis gruppa merker at et medlem begynner å bli demotivert skal man se om det kan være aktuelt om denne personen bytter arbeidsoppgaver for å få opp motivasjonen. Gruppa har avtalt endel sosiale sammenkomster for å gjøre noe annet enn bare jobbe sammen. **Ansvarlig:** Martin Sleire Vatne

Interne konflikter I løpet av prosjektet vil det helt sikkert oppstå noen uenigheter. Det trenger ikke nødvendigvis være noen stor konflikt. **Sannsynlighet:** Medium	Det kan oppstå en situasjon der det ikke blir gjort noe fordi medlemmer i gruppa er uenige og ikke får løst opp i denne uenigheten seg imellom.	Interne konflikter er en viktig grunn til at vi har prosjektmøter hvor hver enkelt medlem i gruppa skal rapportere hva som er blitt gjort, fremdrift og om han er støtt på noen problemer. Det er også derfor vi har sosiale sammenkomster hvor vi får anledning til å gjøre andre ting sammen enn bare å jobbe. **Ansvarlig:** Martin Sleire Vatne
Overskridelse av timebudsjettet Gruppa har fått 1500 timer til rådighet fordelt på 5 personer. **Sannsynlighet:** Medium	Overskridelse av timebudsjettet vil gi mindre tid til å jobbe med andre fag.	Økonomisk ansvarlig holder kontroll med timeforbruket og kommer med en statusrapport for dette på hvert prosjektmøte. Hvis det blir brukt for mange eller for få timer på en oppgave diskuterer gruppa det på prosjektmøtet og vil gi løsninger for å få timebudsjettet i balanse. **Ansvarlig:** Ole Kristian Hoel
Sykdom Sykdom i gruppa kan bli en risikofaktor. Denne risikoen har allerede slått til og personen det gjaldt ble sendt til lege. **Sannsynlighet**: Medium	Om personene blir syke lenge kan det bli vanskeligere for dem å komme tilbake inn i prosjektet. Så lenge det ikke er flere som blir syke samtidig eller over lengre perioder så anslår vi konsekvensen som liten.	Vanskelig å gjøre noen spesielle tiltak. Magnus er ansvarlig for innkjøp av C vitaminer og hostesaft. Alle må trene to dager i uka og holde seg unna asiatiske gryter. **Ansvarlig:** Magnus Solbjørg

Kundestyrt prosjekt

Høst 2001

Forstudie

Versjon 1.10

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
10.09.2001	Opprettet dokumentet, lagt til overskrifter	0.01	Ole Kristian
17.09.2001	La inn ny info	0.02	Magnus
17.09.2001	La til COM og Evaluering	0.03	Martin
18.09.2001	Oppdaterte XML-info	0.04	Magnus
18.09.2001	Oppdatert evaluering litteraturliste, eksisterende løsninger og innledning.	0.05	Martin
19.09.2001	La til produktinformasjon	0.07	Atle
20.09.2001	Fullførte første utkast av evalueringsmal.	0.08	Martin
20.09.2001	Oppdaterte presentasjon av løsninger	0.09	Atle
20.09.2001	La inn nåsit., visjon, mål	0.10	Odd Christian
20.09.2001	Oppdaterte innleding til eksisterende løsninger	0.11	Magnus
20.09.2001	Lagt til om UML, samt flere løsninger	0.12	Ole Kristian
20.09.2001	La inn eks. Løsninger	0.13	Odd Christian
20.09.2001	Rask gjennomgang før første draft.	0.2	Martin
20.09.2001	Lime inn XMI + detaljer på eks.løsn	0.21	Odd Christian
21.09.2001	Utvidet drøfting på eks. Løsn.	0.22	Martin
24.09.2001	La til grovsortering++	0.23	Magnus
26.09.2001	Småkorrigeringer	0.24	Martin
26.09.2001	Limt inn drøftinger på eksisterende løsninger.	0.25	Ole Kristian
26.09.2001	Limt inn konstr. Løsn. + mer om xmi	0.26	Odd Christian
27.09.2001	La til finevaluering av Together Control Center	0.3	Magnus
27.09.2001	Skrevet mer på kap. 2,3,4	0.31	Odd Christian
27.09.2001	La grundigere evaluering av RR	0.4	Martin
28.09.2001	Første utkast på konstruerte løsninger.	0.5	Martin
28.09.2001	Grovsortering oppdatert, generell gjennomgang	0.55	Magnus
30.09.2001	Avsnittet om StructureBuilder lagt til	0.61	Ole Kristian
01.10.2001	La til kategorier i markedsundersøkelsen	0.7	Magnus
01.10.2001	Oppdaterte evaluering	0.71	Martin
01.10.2001	Oppdaterte flerlagsarkitektur.	0.72	Atle
01.10.2001	La til mer på konstruert løsning	0.73	Odd Christian
01.10.2001	Førsteutkast til evaluering av parser	0.8	Martin
01.10.2001	Case for kodegenerering	0.82	Ole Kristian
02.10.2001	Evalueringen fullført, utkast til valg lagt til	0.9	Martin
04.10.2001	Valg, stikkord til konklusjon	0.91	Ole Kristian
04.10.2001	Første utkast til hele dokumentet.	1.0	Martin
05.10.2001	Korrigering av layout	1.01	Martin
15.10.2001	Retting av skrivefeil	1.02	Ole Kristian
18.10.2001	Korrektur, noe nytt innhold, flyttet XMI til vedl.	1.03	Magnus
19.10.2001	Korrektur av minst hundre skrivefeil	1.04	Atle
22.10.2001	Skrivefeil og formuleringer	1.05	Odd Christian
22.10.2001	Layout, figurliste	1.06	Ole Kristian
08.11.2001	Gjennomlesning	1.07	Martin
13.11.2001	Korrektur	1.10	Odd Christian

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dette dokumentet er resultatet av forstudiet i prosjektet og ønsker derfor å spesifisere prosjektoppgaven nærmere. Målet med dette dokumentet er i all hovedsak å avgjøre hvilken løsning prosjektet skal følge videre.

1.2 Avgrensning

Dette dokumentet omhandler kun de løsninger og teknologier som er relevant for prosjektets mandat som spesifisert i prosjektdirektivet [PRO].

1.3 Spesielle definisjoner

Se glossar dokumentet [GLO].

1.4 Dokumentreferanser

[PRO] Prosjektdirektiv, QPRO 2001
[GLO] Glossar, QPRO 2001

Alle eksterne dokumentreferanser er oppsummert i litteraturlisten i kapittel 11.

1.5 Dokumentoversikt

Hele dette dokumentet bygger opp mot valget av løsning som skal bygges videre på.
I de tre første kapitelene (2, 3 og 4) spesifiseres oppgaven nærmere.

Kapittel 5 innholder en rask introduksjon til de sentrale teknologiene i prosjektet samt linker og kilder til videre informasjon. For lesere som ikke er kjent med de teknologiene som her er beskrevet anbefales det at dette kapittelet leses grundig før man fortsetter.

Kapittel 6 innholder de kriteriene som kunden ville legge til grunn for mulige løsninger.

I kapittel 7 kommer markedsundersøkelsen som ble gjennomført, og dette kapittelet innholder også en standard presentasjon av alle de eksisterende løsningene som ble funnet. Kapittel 8 inneholder konstruerte løsninger på problemet. I kapittel 9 evalueres de ulike løsningene opp mot hverandre før det hele summeres med vårt valg i kapittel 10.

2 Nåsituasjon

Edb ASA har egne systemutviklingsprosjekter, og er involvert i prosjekter hos sine kunder. I starten av implementasjonsfasen i et prosjekt brukes mye tid på å skrive kode direkte basert på designspesifikasjonen. Basert på en beskrivelse av datagrunnlaget for et system, lager systemutviklere databasetabeller og komponenter for å aksessere tabellene. Ofte brukes use-case modeller som grunnlag for brukergrensesnitt, og modeller for forretningslogikk som grunnlag for forretningslaget i løsningen. Opprettelsen av nødvendige klasser og komponenter gjøres stort sett "for hånd". Denne jobben er sterkt rutinepreget og tiden som brukes til å lage denne koden burde kunne brukes til mer kreative prosesser i prosjektet. For å korte ned denne delen av utviklingstiden skjer det en del klipp og lim fra tidligere prosjekter. Dette kan gå utover kvaliteten og øke feilraten på koden, særlig på grunn av. problemer knyttet til integrasjon med ny kode. Dette kan igjen føre til ytterligere økning av den totale utviklingstiden.

EDB ASA bruker en iterativ prosess i sine systemutviklingsprosjekter. Det betyr at de gjennom hele prosjektet designer og deretter implementerer utvidelser på systemet de utvikler. Hver iterasjon inneholder en designfase og en implementasjonsfase. Dette krever at man har et klargjort design hver gang man begynner å implementere en ny iterasjon.

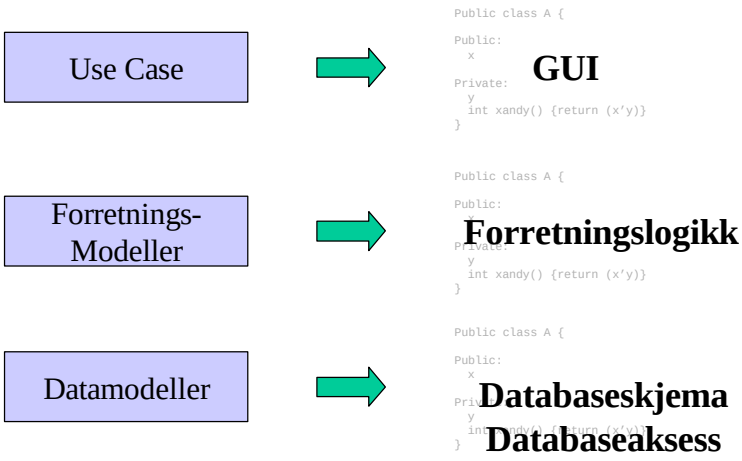
I dagens utviklingsprosjekter opererer EDB ASA og deres kunder med en litt mangelfull struktur på analyse- og designfasen. Når EDB ASA kommer inn i et prosjekt, har kunden svært ofte en uferdig kravspesifikasjon og en altfor lite oppklarende modellering av det ønskede systemet. EDB ASA begynner som regel tidlig i arbeidet med å luke ut feil i det kunden har gitt dem. Mangelfull modellering i utgangspunktet og lite vilje hos kunden til å bruke mer ressurser på dette, fører til at prosjektet går raskt over til implementasjonen av systemet, uten å ha gått gjennom en grundig designfase. Dårlige designspesifikasjoner og modeller gjør starten av implementasjonsfasen enda vanskeligere.

Det er to hovedproblemer med nåsituasjonen hos EDB ASA:

- ?? Det brukes for mye tid på rutinepreget koding
- ?? Dagens prosjekter gjennomføres med så mangelfull modellering at implementasjonsfasen ofte starter uten et klargjort design

3 Visjon

EDB ASA vil komme tidligere inn i prosjekter enn hva de gjør i dag. De vil sikre kvaliteten på kravspesifikasjonen og sørge for at systemet som skal utvikles designes grundig. Designet skal være et komplett grunnlag for implementasjonen. Implementasjonen kommer raskt igang ved hjelp av en automatisk kodegenerator, som tar som input en godt gjennomtenkt og strukturert datamodell, med alle nødvendige entiteter og relasjoner. Generatoren lager skript for opprettelse av databasetabeller og komponenter for å aksessere disse tabellene. Videre tar systemet inn modeller som beskriver forretningslogikk og genererer klasseskjeletter som legger til rette for dette i et forretningslag. Systemet bruker også use-case-modeller for å generere en passende GUI for applikasjonen som er designet.



Figur 3.1 - Skjematisk oversikt over hva som skal konstrueres

Forskjellene mellom nåsituasjonen og visjonen kan dermed oppsummeres i to punkter:

- ?? Sammenhengen mellom modeller og kode styrkes gjennom mer struktur i prosjektene
- ?? Så mye som mulig av koden genereres automatisk utifra modeller istedenfor manuelt

Forbedringene vil føre til en kombinasjon av mindre tidsbruk i implementasjonen og mer fokus på applikasjonslogikk med bedre algoritmer.

4 Finformulering av oppgaven

EDB ASA skal ha et system som realiserer visjonen i kapittel 3. Et viktig dilemma er spørsmålet om hvorvidt det trengs å utvikles et nytt system, eller om det beste for firmaet er å kjøpe og ta i bruk eksisterende verktøy. Oppgaven i prosjektet Qpro16 er først å finne ut hvilken strategi og løsning EDB ASA bør satse på. Videre i dette kapitlet tas det utgangspunkt i hva kunden synes er viktigst i en ny konstruert løsning skreddersydd for dem. Visjonen går utover det systemet vi kan klare å konstruere og implementere innen tidsrammen for prosjektet.

I prosjektet Qpro16 vil det taes høyde for utvidelser som omfatter hele visjonen, konstruksjonen vil dekke et subsett av visjonen og implementsjonen vil dekke et subsett av konstruksjonen. Prosjektet vil derfor snevre seg inn mot en kjernefunksjonalitet, fase for fase. Hvis vi ser på konstruksjonfasen som en gradvis raffinering fra et høyt nivå til et detaljert nivå, ser vi for oss denne utviklingen:

Høynivå konstruksjon	Konstruksjonen skal ligge til rette for å ta inn datamodeller, forretningsmodeller og use-case-modeller, og generere databaseskript, dataaksesskomponenter, forretningslag og grafisk brukergrensesnitt.
Detaljert konstruksjon	Konstruksjonen skal spesifisere en løsning som tar inn datamodeller og forretningsmodeller, og genererer databaseskript, komponenter for databaseaksess og klasser med forretningslogikk. konstruksjon skal være generelt nok til å kunne håndtere forskjellige komponentteknologier, som for eksempel COM og EJB 2.0.
Implementasjon	En demoversjon som tar inn en datamodell og genererer databaseskript og databaseaksesskomponenter for entitetene i datamodellen. Implementasjonen skal dekke en dialekt av SQL for generering av databaseskjema og Container Managed Entity Beans for databaseaksess. Det skal planlegges hvordan UML interaksjonsdiagrammer kan tas inn og sesjonsbønner genereres. De skal være et utgangspunkt for forretningslogikken.

5 Teknologier

Dette kapitlet tar for seg de sentrale teknologiene en løsning av oppgaven kan bygge på, og gir en kort innføring i hver av dem. I tillegg blir de ulike teknologienes plassering i prosjektet forklart. Vedlagt dette dokumentet følger mer detaljerte opplysninger og eksempler fra noen av teknologiene.

5.1 UML

Unified Modelling Language er en samling grafiske modelleringsspråk som har blitt industristandard for å designe programsystemer. UML er også velegnet for å beskrive forretningsstruktur og andre forhold.

5.1.1 Kort innføring

UML har blitt til ved å ta det beste fra flere tidligere modelleringsspråk, først og fremst: Booch OOD, Rumbaugh OMT og Jacobson OOSE. Arbeidet med UML ble startet i 1994 av Booch, Rumbaugh og Jacobson. Etter hvert kom flere med og arbeidet ble støttet av OMG. UML har etter hvert blitt brukt med stor suksess i modelleringen av store og komplekse programsystemer. [UML1]

UML inneholder 9 ulike typer diagrammer som blir brukt for å modellere fra ulike synsvinkler. En enkelt type diagram er som regel ikke tilstrekkelig for å modellere et system, men det er heller ikke vanlig å bruke alle 9 typene. Som regel så velger man ut noen få typer ut i fra hvilken type system man skal modellere og hvilken prosess man bruker i utviklingen. De 9 typene er:

- ?? Klassediagram
- ?? Brukstilfellediagram
- ?? Sekvensdiagram
- ?? Tilstandsdiagram
- ?? Samarbeidsdiagram
- ?? Aktivitetsdiagram
- ?? Komponentdiagram
- ?? Utplasseringsdiagram
- ?? Objektdiagram

En kort beskrivelse av hver av typene finnes i vedlegg 3.

For mer lesestoff om UML anbefales:

- ?? UML resource center fra Rational
  http://www.rational.com/uml/index.jsp

5.1.2 Hva er interessant for prosjektet

I og med at UML er industristandard så vil det i mange tilfeller være et naturlig valg av modelleringsspråk. Dermed vil det også være et naturlig valg av format på modellen som systemet skal ta inn. UML er et grafisk modelleringsspråk, og siden bilder gjerne er lite gunstig som grunnlag for videre behandling er det vanlig i slike sammenhenger å representere UML som XML. Det kommer mer om XML i neste kapittel.

5.2 XML

XML (eXtensive Markup Language) er et "markup"-språk som beskriver data samtidig som data er lagret i filen.

5.2.1 Kort innføring

I XML definerer man selv hvilke tager som skal brukes, og ettersom tagene igjen kan inneholde andre tager har vi muligheten for å bygge opp datastrukturer. Med XML er det altså for eksempel mulig å lagre en enkel database i en vanlig tekstfil.

XML er i grunnen bare en tekstfil der data og datastrukturen er tekst stilt opp på en gitt måte. Man følger en fast standard slik at det er mulig å bruke forskjellige ferdige verktøy på filen, uten å angi noe om hvordan filen er bygd opp.

For å beskrive dataene i en XML-fil bruker man DTD (Document Type Definition). Man kan i DTD definere entitetene og angi hva som er gyldige verdier. I tillegg forteller DTDen blant annet hvor tekst er ventet å finne, samt om mellomrom kan oversees eller ikke.

XML er en åpen standard så brukeren trenger ikke noen lisens for å ha en XML-fil eller lisens til et program som manipulerer med den. Blant annet derfor er XML spådd til å bli det nye store innen utveksling av data på Internett.

En mer detaljert beskrivelse om XML og DTD finnes i vedlegg 1.

XML-parsing

Når man har laget en XML-fil er det nødvendig å parse den videre for å få den på et format som er enkelt å jobbe med. Man kan for eksempel bruke en parser som lager javaobjekter av XML-filen. Det vil da være enkelt å jobbe videre med dataene gjennom for eksempel en javaapplikasjon.

Mer om XML-parsing finnes i vedlegg 2.

5.2.2 Hva er interessant for prosjektet

Vi er interessert i XML fordi vi skal lage et system som trenger en robust input som grunnlag for kodegenerering. XML er en godt utarbeidet standard, og har vist seg å være en god måte å representere data på. Det tenkes å vinkle XMLen til å representere UML-modeller, og da kalles det for XMI. Mer om denne standarden i neste kapittel.

5.3 XMI

XMI (XML Metadata Interchange) er en standard for representasjon av objektorienterte modeller som muliggjør enkel deling av data mellom ulike verktøy fra forskjellige produsenter.

5.3.1 Kort innføring

XMI skal utnytte fordelene med både UML og XML. XMI er en utnyttelse av XML for representasjon av UML.

Tanken er at en UML-modell brukes for å beskrive et domene, og at denne modellen brukes til å generere en XMI DTD (Document Type Definition). DTDen er her en definisjon på en måte å representere dataene fra UML-modellen på. Denne definisjonen kan flere verktøy bruke, slik at blant annet utviklingsverktøy, modelleringsverktøy og rapportgeneratorer kan dele dataene fra den opprinnelige UMLen. Det eneste kravet som stilles er at alle disse verktøyene kan oversette mellom XMI og sin egen representasjonsform.

5.3.2 Hva er interessant for prosjektet

Det som er poenget med XMI i dette prosjektet, er at XMI er en god standard for UML-representasjon som kan parses til kode. XMI-dokumenter er egentlig XML-dokumenter som utnytter XMI-tager, slik at parsing av XMI og XML er det samme (se kapittel 5.2).

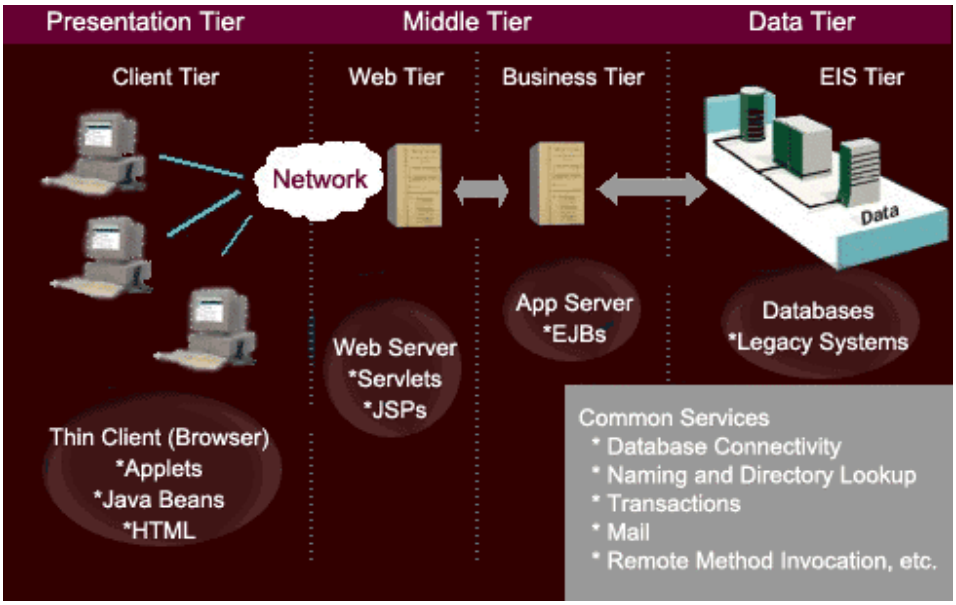
Eksempler på bruke av XMI finnes i vedlegg 5.

5.4 Enterprise JavaBeans (EJB)

5.4.1 Kort innføring

Kommunikasjon mellom datamaskiner koblet i nettverk er blitt et viktig begrep i dagens IT samfunn, og populariteten til distribuert programvare har økt dramatisk. Java 2 Enterprise Edition (J2EE) har raskt blitt en av de dominerende innenfor distribuert programvaremiljø. J2EE inkluderer blant annet den adaptive EJB teknologien som man håper er et skritt i riktig retning med tanke på komponentbasert programvareutvikling. Hovedtankene bak komponentbasert programvareutvikling handler om å lage programvare som kan kjøre på distribuerte systemer med mangelags arkitekturer og som er portable til både ulik maskinvare og ulike miljøer; servere, databaser etc. EJB er byggeklosser for komponenter som lever på serveren. I tillegg finnes det Java Servlets som tar seg av interaksjon mellom servere og web klienter og Java Server Pages (JSP) som lar utviklerne lage dynamisk innhold. J2EE gjør det enklere for proagrammerere å skrive kompleks programvare uten å være eksperter på distribuerte systemer, skalerbarhet, sikkerhet etc.

Flerlagsarkitektur



Figur 5.1 - Eksempel på flerlagsarkitektur

En vanlig firelags arkitektur kan for eksempel bestå av klient, webserver, applikasjonsserver og databaseserver. La oss anta at du handler på nettet. Din nettleser er klienten som viser HTML, Java Applet etc. Når du trykker på ”send” starter det en servlet på webserveren. På dette nivået vil du typisk finne JSP, HTML etc. J2EE introduserte dessuten en ny mulighet for å pakke webapplikasjonen inn i et Java Web Archive (WAR). Servleten trenger kanskje noen data via en EJB på applikasjonsserveren.

Applikasjonsserveren er den som tar hånd om all forretningslogikken og sørger for at transaksjoner, sikkerhet, persistens etc. EJBen må kanskje aksessere en database og hente den informasjonen du spurte om før svaret kan returneres deg tilbake. Ved å skille arkitekturen på denne måten kan man enkelt oppgradere en av lagene uten å måtte bygge om hele systemet. Utseendet til webserveren kan oppdateres uten å modifisere noe på forretningslogikken.

Enterprise JavaBeans

Den primære egenskapen til EJB er å forenkle utviklingen av forretningslogikk. Vi kan dele EJB bønner inn i to kategorier:

Session Beans
Forestill deg at disse bønnene implementerer arbeidsflyt eller prosesser, for eksempel hotellreservasjon, overføring av penger fra en konto til en annen etc. Dette er transiente aktiviteter. Straks en oppgave er ferdig, har ikke bønna lenger noen funksjon og opphører derfor å eksistere.

Det finnes to typer Session Beans:

?? **Stateless Session Beans**
Disse bønnene utfører atomiske operasjoner, det vil si at straks bønna er ferdig med sin oppgave så opphører kommunikasjonen mellom bønna og klienten. Det blir ikke lagret noen attributter for bønna så lenge den lever. Et godt eksempel er autorisering av kredittkort. Stateless Session Beans forhindrer at uvedkommende kan utnytte den gamle bønna til å autorisere enda et kredittkort.

?? **Stateful Session Beans**
Dette er bønner nyttige for mer komplekse aktiviteter. Bønnene husker ting fra en metode til en annen slik at du kan kalle opp den samme bønna flere ganger og fortsette kommunikasjonen. Hvis du handler på nettet og bruker en handlekurv vil denne handlekurven kunne representeres av en Stateful Session Bean.

Entity Beans
Entity Beans er den andre typen java bønner. Dette er objektorienterte representasjoner av persistente data hentet fra relasjonsdatabaser, for eksempel en representasjon av alle hotellene i Oslo, kundedatabasen til et firma eller dine bankkonti. En av de store fordelene med Entity Beans er at du slipper lange SQL spørringer. De er byttet ut med metodekall på bønna. Siden entitetsbønner refererer til databasetabeller må de være unikt definert med en primærnøkkel og holdes synkronisert med databasen. Denne synkroniseringsprosessen kalles persistens.

Det finnes to typer persistens for entitetsbønner:

?? **Container Managed Persistence (CMP)**
Dette er det enkleste opplegget for å oppnå persistens, hvor du bare ber en Container ta seg av alt. Den kan ta seg av forretningslogikk, spesifisere hvordan

entitetsbønner mapper til felter i en database og slappe av mens bønna genererer SQL spørringer under kjøretid.

?? **Bean Managed Persistence (BMP)**
Hvis du ønsker mer kontroll, for eksempel gjøre ting mer effektivt enn den automatisk genererte koden, da må du bruke BMP. Der kan du skrive all database aksesslogikk som en del av din egen bønne. Ulempen er at du må forstå databasestrukturen og SQL og kode endel mer enn med CMP.

Struktur i implementasjon
Hver EJB har et Remote grensesnitt såvel som et Home grensesnitt. EJBObject og EJBHome implementerer disse grensesnittene. Klienter bruker Home grensesnittet for å komme med forespørsler om hvordan finne, skape eller slette en bestemt EJB. Remote grensesnittet omhandler alt som har med nettverk, transaksjoner, sikkerhet etc. å gjøre. Dette siste grensesnittet inneholder også metoder som har med selve forretningslogikken å gjøre og vil fordele dem hvis de blir kjørt.

Følgende artikler kan gi mer detaljert informasjon om EJB:

?? Enterprise JavaBeans Technology : http://java.sun.com/products/ejb/

?? Enterprise Java and Rational Rose Part I and II : Khawar Ahmed, Loïc Julien (2001)

5.4.2 Hva er interessant for prosjektet

Enterprise JavaBeans er en mye brukt arkitektur, og det er nettopp denne arkitekturen det er lagt hovedvekt på i dette prosjektet. Den generatoren som skal velges/lages må ha utstrakt støtte for EJB. Det er derfor viktig å ha inngående kunnskap om EJB for å velge eller lage en god generator. I neste kapittel kan du lese om COM, som er en tilsvarende teknologi.

5.5 COM – Component Object Model

Innføringen er skrevet for lesere som før har vært borte i distribuerte systemer, og har vært borte i noen av konkurrentene EJB eller CORBA.

5.5.1 Kort innføring

COM er en åpen generell arkitektur for komponent basert softvare laget av Microsoft og DEC. Arkitekturen er et alternativ til CORBA eller EJB og har blitt veldig sentral i Microsoft sine programmer og operativsystem. Windows 2000 er for eksempel bygd rundt COM arkitekturen. [COM1]

Som i CORBA skjer all kommunikasjon ved hjelp grensesnitt hos komponentene, som er kontrakter mellom komponenter og/eller klienter. Mekanismene i bakgrunnen er veldig like med CORBA, og inneholder blant annet IDL, GUID, marshalling og stubber, men på

et punkt er det en voldsom forskjell: I COM kan en komponent implementere et vilkårlig antall grensesnitt. [COM2]

COM definerer en binær standard som muliggjør kommunikasjon mellom komponenter implementert i forskjellige språk. Dette medfører at alle språk som kan kalle funksjoner ved hjelp av pekere, for eksempel ADA, VB, C++, kan samarbeide med hverandre dersom de følger denne standarden. [COM3]

Når en applikasjon trenger en tjeneste funger COM som en megler som finner hvilken komponent som tilbyr den ønskede funksjonaliteten. Komponentene ligger alle som Dynamic Libary Linking (.DLL) eller Executable (.EXE) filer i binær form. Når forbindelsen er opprettet har COM gjort sitt. [COM1]

Vanskeligheten i praktisk implementasjon av COM komponenter avhenger i stor grad av valg av utviklingsspråk og utviklingsmiljø. Dersom man for eksempel velger Visual Basic (VB) som utviklingsspråk og Microsoft Visual Studio som utviklingsverktøy, blir COM gjort transparent for utvikleren og i kildekoden til stor grad.

Rundt COM har man etter hvert bygget Distribuert COM (DCOM). Ved å inkludere noen av tjenestene tidligere funnet i Microsoft Transaction Server (MTS) og tillegging av noen nye har man nå oppnådd COM+.

For videre lesning om COM anbefales:

- ?? Microsoft COM – Component Object
  http://www.microsoft.com/com/presentations/realtour.zip

- ?? COM+ Building on the Success of the Component Object Model
  http://www.microsoft.com/com/presentations/complus.zip

- ?? Component Object Model vs. Enterprise JavaBeans
  By Michael Vizard - InfoWorld Electric
  http://www.infoworld.com/cgi-bin/displayStory.pl?/interviews/981027sigsroundtable.htm

- ?? Flere COM relaterte sider
  http://www.quintuslink.com/cetus/oo_ole.html

- ?? Microsoft sine offisielle COM sider
  http://www.microsoft.com/com/

5.5.2 Hva er interessant for prosjektet

COM er en av målarkitekturene til prosjektet, det vil si at løsningene bør ta høyde å generere COM komponenter i tillegg til EJB i vår kodegenerator. Et endelig valg av språk på komponentene er ikke gjort enda, men fokuset legges på VB og C# inntil videre.

6 Kriterier for evaluering av løsninger

Dette kapittelet inneholder de kriterier og de krav kunden vil vektlegge i evalueringen av løsninger. Disse kriteriene vil ligge til grunn for vurderingen av de eksisterende løsningene, konstruksjonen av nye løsninger, og til slutt evalueringen av alle løsningene.

I praksis blir momentene i dette kapittelet en veldig overordnet kravspesifikasjon, og vil da i tillegg ligge til grunn for den senere konstruksjonen av den detaljerte kravspesifikasjonen.

6.1 Krav til inndata

Programvare som skal komme i betraktning må ha inndata som er generelle og det skal ha evnen til å motta en modellbeskrivelse. Det skal være mulig å sette generelle parametre slik som prosjektnavn og katalogstruktur. Hvis man for eksempel modellerer UML kan man tenke seg å overføre denne modellen over til en standardisert XML/XMI beskrivelse for den samme modellen. Det er ikke noe bestemt krav til at modellen skal komme i form av UML, men det er naturligvis en stor fordel ettersom denne standarden er såpass utbredt i verden. Siste versjon av UML er versjon 1.3, og det er fortrinnsvis denne som bør kunne taes som inndata. Det finnes mange eksisterende programmer som lar deg modellere visuelt i UML og få eksportert denne i diverse formater.

6.2 Krav til generator

Når det gjelder den kodegeneratoren som skal inngå i løsningen, må denne være generalisert for å takle ulike typer språk. Det er ønskelig at det skal være mulig å benytte generatoren både for EJB, DCOM/COM og eventuelt andre teknologier. Reglene som gjelder for disse språkene kan for eksempel være beskrevet i XML. Dette er kanskje den største utfordringen, nemlig at vi ikke binder oss spesifikt til en bestemt komponentteknologi. Ved evalueringen av løsninger skal det derfor også legges vekt på at programmet kan håndtere ulike komponentteknologier, i skrivende stund med EJB og COM som de to store.

Noe stort behov for et brukergrensesnitt finnes ikke, ettersom brukerne er erfarne programvareutviklere, men det er selvfølgelig ønskelig med et brukergrensesnitt. Programvare som skal komme i betraktning må kreve lite forkunnskaper, være enkelt i bruk, og må kunne automatisere og minimere oppstartstiden til EJB prosjekter hos kunden betraktelig fra dagens manuelle situasjon.

6.3 Krav til utdata

Det som er en av de viktigste prioritetene og et absolutt krav er at løsningen som skal evalueres klarer å generere kode for EJB 2.0 entitetsbønner, og da særlig bønner som er ”container managed”. Mer om ulike typer bønner kan det leses om i kapittel 5.4 Enterprise JavaBeans. Det er også ønskelig at den samme løsningen kan generere sesjonsbønner for svært enkle forretningsprosesser. I første rekke gjelder dette bønner som kan liste opp, legge inn og validere data. Som en prioritet nummer tre kommer bønner med ”bean managed persistence”, der man må skrive SQL-koden selv. Hvis programvaren i tillegg kan generere kode for andre tilsvarende teknologier, for eksempel COM, er dette en fordel men ikke et absolutt krav.

6.4 Krav til pris

Prisen spiller naturligvis en rolle. Hvis det er et lite antall EJB oppdrag vil det ikke lønne seg å ha mange dyre lisenser til et produkt som nesten ikke blir brukt. Dyre lisenser påfører bedriften ekstra kostnader som kanskje kunne vært unngått med alternative produkter. I dette spesielle tilfellet er det fullt mulig at det finnes løsninger som tilbyr samme funksjonalitet, men ligger i en betydelig lavere prisklasse enn det opprinnelige produktet.

7 Markedsundersøkelse

Dette kapittelet er resultatet av markedsundersøkelsen som ble gjennomført for å finne de eksisterende løsningene til prosjektets oppgave.

7.1 Innledning

Oppgaven i dette prosjektet, som i hovedsak er automatisk kodegenerering, er noe som det allerede eksisterer mange løsninger på. Grunnen til at slike løsninger har blitt utviklet er at det er veldig komplekst og tidkrevende å lage kode. Ved hjelp av slike spesialprogrammer har det vært mulig å redusere både tidsbruk og feilkilder.

Det er spesielle ønsker og krav for hvordan vår kodegenerator skal være, og i dette kapittelet vil endel eksiterende kodegeneratorer bli testet og vurdert.

7.2 Risikohåndtering

Da denne presentasjonen skulle ligge til grunn for senere evaluering av løsningene måtte det legges vekt på risikoer i forbindelse med markedsundersøkelsen. De største risikoene var en mangelfull markedsundersøkelse og en subjektiv eller urettferdig presentasjon av løsningene.

For å motvirke mulighetene for en mangelfull markedsundersøkelse brukte gruppen en hel dag på søke etter mulige løsninger. De løsningene som er tatt med har vært igjennom en kort silingsprosess med hovedvekt på om de faktisk kan være en løsning, eller en del av en løsningen på prosjektet. Til slutt viste 21 løsninger seg å inneholde interessant funksjonalitet.

For å unngå subjektive presentasjoner ble det er lagt vekt på å innhente informasjon om samme produkt fra flere kilder samt det ble definert en fast mal som presentasjonene skulle følge. Videre ble det definert et sett av spesifikke egenskaper løsningene skulle sjekkes mot i en standard tabell.

7.3 Mal til presentasjonene

Løsningen som skal evalueres vil bli delt inn inn i 3 kategorier:

1. Totalløsninger, både modellering og generering
2. Genereringsløsninger, bare kodegenerering
3. Modelleringsløsninger, bare modellering

Løsningene deles inn i kategorier fordi de skiller seg fra hverandre i hvor omfattende de er, og hvilken hovedoppgave de er ment å utføre.

Totalløsninger vil typisk være store programmer som tar for seg en utviklingsprosess helt fra modellering til ferdig kode. En genereringsløsning vil kunne ta imot en modell og generere kode ut fra den. En modelleringsløsning vil være utgangspunktet for en kodegenerator. Man kan kanskje se for seg at en løsning fra kategori 2 og 3 til sammen kan være en totalløsning, da med krav om at generatoren kan importere XMI, og modelleringsverktøyet kan eksportere XMI. En modelleringsløsning vil også kunne kombineres med en egenutviklet kodegenerator.

Presentasjonene er bygget opp etter et fast mønster. Først er det navn på produkt, samt link til produsent og produkt. Så følger en kort beskrivelse av programmet, med en matrise av viktige egenskaper programmene må/bør ha. Denne er fylt ut etter hva programmet oppfyller. Denne matrisen vil bli poeng gitt i evalueringen. Så har følger en liste av andre viktige egenskaper programmene har, før det hele oppsummeres med en kort drøfting av produktet.

Beskrivelse av elementene i sjekkmatrisen:
XMI-import: Mulighet til å importere XMI-formatet
XMI-eksport: Mulighet til å eksportere XMI-formatet
UML-editor: Har editor for å lage/editere UML-modeller
Reengeneering av eksisterende EJB komponenter: Mulighet til å ta inn gamle EJB komponenter for å endre på dem
Deployment på applikasjonsserver: Støtte for å automatisk legge bønner ut på server
EJB kode (home/remote): Lager home/remote automatisk
EJB 2.0 - Container Managed Persistance i entity beans: Støtte for å lage CMP bønner
EJB 2.0 – Bean Managed Persistance i entity beans: Støtte for å lage BMP bønner
EJB 2.0 – Foretningslogikk i sessions beans: Støtte for å lage sesjonsbønner med foretningslogikk
COM komponenter: Støtte for å generere COM-komponenter
GUI: Støtte for å lage brukergrensesnitt til den genererte koden
SQL databasescript: Støtte for å lage SQL-script som oppretter nødvendige databaseelementer

Tegnforklaring til evalueringsmatrise:

- ✔ Funksjonaliteten er støttet
- ✘ Funksjonaliteten er ikke støttet
- ○ Funksjonaliteten er planlagt støttet

Løsningene som blir presentert i denne markedsundersøkelsen er:

Totalløsninger:

- ?? ArcStyler
- ?? ObjectIF
- ?? Rational Rose 2001
- ?? SoftModeler 3
- ?? StructureBuilder
- ?? Together Control Center 5

Genereringsløsninger:

- ?? Avantis
- ?? CocoBase
- ?? Cool:Joe 2.0
- ?? Describe
- ?? EJBGen 1.21
- ?? EJBQuick
- ?? EJBX J2EE Code Generator
- ?? Forte
- ?? Genova 7.0
- ?? LowRoad
- ?? Panther
- ?? Pramati Studio 2.5
- ?? Software through pictures/UML

Modelleringsløsninger:

- ?? Novosoft UML Library
- ?? Select Component Factory

7.4 Totalløsninger

7.4.1 ArcStyler

Utgitt av: **Interactive Objects Software**
Web:
http://www.io-software.com
http://www.io-software.com/products/

Produktbeskrivelse

Et verktøy som implementerer Model Driven Architecture (MDA) og Rational Unified Prosses (RUP) for konstruksjon av EJB systemer for Internet. Mer om RUP kan du lese i vedlegg 4.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✔
Reengeneering av eksisterende EJB komponenter	✔
Deployment på applikasjonsserver	✔
Kodegenerering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistance i entity beans	✘
EJB 2.0 – Bean Managed Persistance i entity beans	✘
EJB 2.0 – Foretningslogikk i sessions beans	✔
COM komponenter	✘
GUI	✔
SQL databasescript	✘

PRIS: kr 240.000 (Architect Edition trenger i tillegg Rational Rose Modeler Edition)

Andre egenskaper

Et proffesjonelt program for støtte av de ulike standard prosessformene RUP og MDA. Det bygger på Rational Rose og legger egne støtteapplikasjoner oppå denne. Videre er det spesialiserte kodegeneratorer for de ulike applikasjonsserverene som baserer kodegenereringen på templates med ”best-practise” løsninger.

Drøfting

Et litt for tungt program for vår snevre oppgave. Omfatter veldig mye i seg selv, og bygger i tillegg videre på alt det som Rational Rose eventuelt tilbyr. Dette medfører at programmet er lite fleksibelt.

Mulighetene er derimot store dersom ArcStyler benyttes. All funksjonalitet som Rationale Rose tilbyr bygges på med andre muligheter som Class Responsebility Cards (CRC) og en god støtte for prosjektoppfølging.

Kodegenereringen virker veldig bra i kvalitet, og selv om ikke programmer støtter EJB 2.0 direkte kan mye av dette utføres ved hjelp av Rational Rose.

7.4.2 ObjectIF

Utgitt av: **Microttool**
Web:
http://www.microtool.de/e_index.htm
http://www.microtool.de/objectiF/en/

Produktbeskrivelse

ObjectIF hjelper til fra brukstilfelleanalysering til implementasjon.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✓
Reengeneering av ekstisterende EJB komponenter	✗
Deployment på applikasjonsserver	✗
Kodegenerering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistence i entity beans	✗
EJB 2.0 – Bean Managed Persistence I entity beans	✗
EJB 2.0 - Foretningslogikk i sessions beans	✗
COM komponenter	✓
GUI	✗
SQL databasescript	✓

PRIS: kr 40.000 (flytene lisens), 18.500 (1 arbeidstasjon)

Andre egenskaper

- ?? Støtte for UML 1.3 med brukstilfellediagram, aktivitetsdiagram, klassediagram, sekvensdiagram, tilstandsdiagram, pakker etc.
- ?? Transaksjonsorientert flerbrukerutvikling over LAN.
- ?? Kodegenerering av C++, Java, Visual Basic, ANSI C, IDL, DDL
- ?? Reverse engineering av både enkeltklasser og biblioteker i Java og C++.

Drøfting

ObjectIF tilbyr to muligheter for utvikling av EJB. Med EJB Generation tool kan utviklere modellere uavhengig av komponentteknologi og programmet vil generere en løsning. Den andre muligheten er EJB Wizard der utvikleren har mer kontroll med de tekniske klassene under implementasjonen. [OIF1] ObjectIF støtter dessverre ikke EJB 2.0 som er et relativt høyt prioritert kriteria i vårt prosjekt.

7.4.3 Rational Rose 2001 Enterprise Edition

Utgitt av: **Rational**
Web:
http://www.rational.com/
http://www.rational.com/…
.../products/rose/index.jsp

Produktbeskrivelse

Rose står for Rational Object Oriented Software Engineering og er et av de mest dominerende verktøyer for objektorientert modelldrevet analyse, modellering, design og konstruksjon av programvare.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✔
Reengeneering av ekstisterende EJB komponenter	✔
Deployment på applikasjonsserver	✔
Kodegenerering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistence i entity beans	✔
EJB 2.0 – Bean Managed Persistence i entity beans	✔
EJB 2.0 - Foretningslogikk i sessions beans	✔
COM komponenter	✔
GUI	✔
SQL databasescript	✔

PRIS: kr 37.000

Andre egenskaper

- ?? Støtte for aktivitetsdiagram, brukstilfellediagram, sekvensdiagram, samarbeidsdiagram, klassediagram, tilstandsdiagram mfl.
- ?? Kodetemplates
- ?? Kodegenerering og Reverse engineering for C++, Visual Basic, COM, Ada, Java, J2EE, Corba/IDL, XML DTD, MIDL mfl.
- ?? Web modellering av ASP og JSP sider
- ?? Importerer fra og eksporterer til IBM Visual Age, Webgain VisualCafe, Sun Forte for Java, Borland Jbuilder m.fl.

Tradisjonelt består programvareutvikling av å samle inn krav, modellere programmets forretningslogikk og struktur, skrive programkoden og til slutt test. Lenge var denne fremgangsmåten tilstrekkelig, men kompleksiteten hos de fleste av dagens moderne

dataprogrammer gjør denne prosessen kostbar, tidskrevende og ineffektiv for dagens programvareprodusenter. Rational sine verktøyer spenner over hele denne utviklingsfasen og muliggjør effektiv integrering av alle faser fra innsamling av krav til kodegenerering og vedlikehold. Rational Rose lar deg visualisere, forstå og raffinere dine krav og arkitektur før de blir omgjort til kode. Dette hjelper deg til å unngå tapt arbeidstid i utviklingsfasen.[ROS1]

Den siste versjonen av Rational Rose er versjon 2001. Denne inneholder blant annet Rational QualityArchitect som hjelper til å generere kode fra UML-designmodeller, enten for å teste hele systemet eller for å teste individuelle deler av systemet før resten av systemet er ferdig bygget..[ROS2] Rational Rose automatiserer kodingen av stubs og drivere for EJB og DCOM/COM+ komponenter og verifiserer at komponentene er konstruert i henhold til vedtatte standarder for disse teknologiene. Det finnes støtte for både EJB 1.1 og den nyere versjon 2.0. Programmet inneholder dessuten ”wizards” som kan benyttes til raskt å få opp et skjellett for ny programvare i f.eks. EJB og JSP. Full støtte for import og eksport av XMI kommer i form av en liten tilleggsmodul som kan lastes ned gratis fra Rational sine hjemmesider. Alle disse nye evnene gir utviklerne muligheten til å øke hastigheten på design og koding, med menydrevet aksess til repeteremde mønstre som raskt kan legges til den visuelle designmodellen. [ROS3] Rational Rose kan brukes både til systemkoding og til å designe brukergrensesnitt. Det finnes mange firmaer som produserer støtte for Rational sine produkter, noe som både gjør programmet svært kompatibelt med annen programvare og øker mulighetene for senere utvidelser og tillegg til Rose betraktelig. En bakdel med produktet er den høye prisen. Rational Rose Enterprise Edition koster ifølge Rational sine hjemmesider 4194 USD, noe som tilsvarer ca 37.000 NOK. Kjøper man lisensen via en forhandler her i Norge ligger prisen på omtrent det dobbelte.

For å gjøre datamodeller enda mer tilgjengelig for Java verdenen har Rational og Java Community Process nylig annonsert Java Request 26 (JSR-26). Dette er en pågående prosess der man ønsker å definere en standard mapping mellom EJB og UML. Denne standarden vil tillate verktøy fra flere leverandører å arbeide sammen via et felles format for overføring av modeller. Den setter en standard for hvordan EJB blir modellert og representert i UML.

Drøfting

Rational Rose er et kraftig verktøy som antakeligvis vil score høyt på evalueringen. Ulempene er den høye prisen, samt at man får masse ekstra som man kanskje ikke har behov for. Det er selvfølgelig mulig at du får bruk for dette senere, og kan i så måte være en forhåndsinvestering.

7.4.4 SoftModeler 3

Utgitt av: **Softera**
Web:
http://www.softera.com/
http://www.softera.com/products.htm

Produktbeskrivelse

Et helt verktøy for hele systemutviklingsprosessen rettet kun mot utvikling innen Java.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✔
Reengeneering av eksisterende EJB komponenter	✔
Deployment på applikasjonsserver	○
Kodegenerering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistance i entity beans	○
EJB 2.0 – Bean Managed Persistance i entity beans	○
EJB 2.0 - Foretningslogikk i sessions beans	○
COM komponenter	✘
GUI	✔
SQL databasescript	✔

PRIS: kr 8800 (Enterprise Edition)

Andre egenskaper

SoftModeler 3 inneholder redskap for hele den objektorienterte utviklingsprosseen. Programmet spenner over use-case, domeneanalyse, sekvensdiagrammer, tilstandsdiagrammer og klassediagrammer . I tillegg finnes løsninger som animasjon av sekvensdiagrammer, deteksjon av deadlocks i ”multi thread” systemer og simuleringsmuligheter. Støtter dessuten samtidighet ved å låse de ulike diagrammene isteden for hele prosjektet og lagrer diagrammene som bilder.

Drøfting

Virker som et veldig solid og godt produkt for hele OOAD fasen. Finnes dessverre ikke i nåværende form med alle de egenskapene som dette prosjektet trenger, men mesteparten av disse er planlagt og vil komme i neste versjon. Konsekvensen av dette er at denne versjonen ikke vil passe til prosjektet, men man bør ta en ny evaluering når neste versjon lanseres.

7.4.5 StructureBuilder

Utgitt av: **Webgain**
Web:
http://www.webgain.com
http://www.webgain.com/products/structure_builder/

Produktbeskrivelse

Et helt verktøy for hele systemutviklingsprosessen rettet kun mot utvikling innen EJB systemer.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✔
Reengeneering av eksisterende EJB komponenter	✔
Deployment på applikasjonsserver	✔

Kodegenerering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistance i entity beans	✔
EJB 2.0 – Bean Managed Persistance i entity beans	✔
EJB 2.0 - Foretningslogikk i sessions beans	✔
COM komponenter	✘
GUI	✔
SQL databasescript	✘

PRIS: kr 23.000 (Enterprise Edition)

Andre egenskaper

Har use-case diagram, tilstandsdiagram, sekvensdiagram, aktivitetsdiagram og aktivitetsdiagram som alle leder opp mot den automatiske genereringen av EJB-kode. Synkroniserer senere kode og modell slik at modellen alltid gjenspeiler den faktiske tilstanden til koden.

Drøfting

Virker som et bra konstruert program som støtter hele systemutviklingsfasen, og dermed en del mer enn hva vårt prosjekt etterspør. Innehar veldig mange av de egenskapene som det stilles krav om i vårt prosjekt, men mangler støtte for andre arkitekturer. Dette medfører at produktet er lite fleksibelt, da det kun støtter EJB-prosjekter.

7.4.6 Together ControlCenter 5

Utgitt av: **Togethersoft**
Web:
http://www.togethersoft.com
http://www.togethersoft.com/products/controlcenter/index.jsp

Produktbeskrivelse

Togehter ControlCenter er Togethersoft sitt hovedprogram. Programmet genererer ønsket kode, etter å ha gitt sin input som for eksempel UML eller XMI.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✔
Reengeneering av ekstisterende EJB komponenter	✘
Deployment på applikasjonsserver	✔

Kodegenerering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistance i entity beans	✔
EJB 2.0 – Bean Managed Persistance i entity beans	✔
EJB 2.0 - Foretningslogikk i sessions beans	✔
COM komponenter	✔
GUI	✔
SQL databasescript	✔

PRIS: kr 53.000

Andre egenskaper

- ?? Samtidig utvikling for Java, C++ og IDL
- ?? Platform-nøytralt format
- ?? God brukervennlighet med intuitive skjermbilder
- ?? Datamodellering:
  - Entitet-relasjons-diagram editor
  - DBMS konfigureringsekspert
- ?? XML struktur editor
- ?? JSP/HTML editor
- ?? Databasemapping
- ?? Versjonskontroll

Programmet inneholder i tillegg en rekke andre egenskaper som kan være nyttig i en utviklingsprosess.

Drøfting

ControlCenter har full støtte for EJB 2.0, og hånderer dette enkelt og greit. Det er mulig å lage JSP-sider for å enkelt teste ut komponentene man lager. Inputen til programmet er UML, eller man kan importere XMI, DTD eller fra Rational Rose. Datagrunnlag kan man importere direkte fra database. Programmet virker fleksibelt og har svært mange funksjoner for å få til den outputen som er ønskelig.

ControlCenter er en stor løsning, da både med tanke på hva den kan utføre ytelseskrav og pris. Velger man å innvistere i en slik løsning får man imidlertid et program som inneholder det meste av påkrevd funksjonalitet for å kunne utføre en modellering til kode prosess.

7.5 Genereringsløsninger

7.5.1 Avantis Unisuite

Utgitt av: **Avantis**
Web:
http://www.avantis.de
http://www.avantis.de/products/avantis/umlbridge.htm

Produktbeskrivelse

Avantis Unisuite består av tilleggsmoduler for noen utvalgte modelleringsverktøy, og tilbyr kodegenerering ut ifra UML-modeller. Det finnes foreløpig tre UML-broer: en for EJB, en for Java og en for CORBA. Her følger en gjennomgang av Avantis Unisuite for EJB.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av eksisterende EJB komponenter	✗
Deployment på applikasjonsserver	✗
Kode generering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistance i entity beans	✓
EJB 2.0 – Bean Managed Persistance i entity beans	✓
EJB 2.0 – Foretningslogikk i sessions beans	✗
COM komponenter	✗
GUI	✗
SQL databasescript	✓

PRIS: ?

Andre egenskaper

Avantis Unisuite for EJB tilbyr en del tjenester som forbedrer koblingen opp mot databasen:

- ?? spørringer via OQL, som overflødiggjør jobbing på SQL-nivå
- ?? mulighet for å spesifisere samtidighetsalgoritme
- ?? konfigurering en cache for databaseaksesser
- ?? transaksjonshåndtering koblet opp mot JTS for å minimalisere kommunikasjon med databasen
- ?? optimalisering for økt hastighet i transaksjonene

Drøfting

Det er god støtte for generering av EJB ut ifra UML. Det som imponerer med produktet er all den ekstra funksjonaliteten som gir store muligheter til å lage et pålitelig og raskt databaseaksesslag. Mye av denne funksjonaliteten er avhengig av at man har en kompatibel applikasjonsserver.

Produktet skal brukes sammen med en UML-editor, og kan bare brukes sammen med noen få utvalgte editorer som Avantis har laget tilleggsmoduler for. En av editorene som kan brukes er Rational Rose, som er svært utbredt og omfattende. Siden Avantis ikke har noe støtte for XMI import, blir man knyttet opp mot bare en liten del av markedet innen UML, og det er en klar mangel.

På databasesiden har produktet god støtte, særlig p.g.a. at JDBC kan brukes. Hvilke databaser, applikasjonsservere og UML editorer som støttes, kan man finne ut mer om på hjemmesiden til firmaet: http://www.avantis.de/products/avantis/unisuite.htm.

7.5.2 CocoBase

Utgitt av: **Thought**
Web:
http://www.thoughtinc.com
http://www.thoughtinc.com/cber_index.html

Produktbeskrivelse

CocoBase tilbyr objekt til relasjonsmapping for utviklere som skriver applikasjoner for J2EE platformen, ved bruk av EJB/JSP/Java klasser/Servlets/etc.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✘
UML-editor	✘
Reengeneering av ekstisterende EJB komponenter	✘
Deployment på applikasjonsserver	✔

Kodegenerering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistance i entity beans	✘
EJB 2.0 – Bean Managed Persistance i entity beans	✘
EJB 2.0 - Foretningslogikk i sessions beans	✘
COM komponenter	✘
GUI	✔
SQL databasescript	✔

PRIS: kr 33.000

Andre Egenskaper

?? Lager persistent kode etter import fra Rational Rose eller Together ControlCenter
?? Bra skalerbarhet

Drøfting

CocoBase baserer seg på å lage kode etter å ha importert en modell fra Together Control Center eller Rational Rose. Programmet genererer så dynamisk persistent kode fra redigerbare templates. Dette er altså et program som bare genererer kode. Det har støtte for EJB 1.1, Servlets og JSP. Det kan også generere EJB-kode for bruk på alle de mest populære J2SE/J2EE EJB aplikasjonsserverne. Dette er et ganske enkelt program, som ikke har så mange funksjoner.

7.5.3 Cool:Joe 2.0

Utgitt av: **Computer Assosiates**
Web:
http://www.ca.com/
http://www.ca.com/products/cool/cooljoe.htm

Produktbeskrivelse

Cool:Joe er et integrert utviklingsmiljø som forenkler kompleksiteten til J2EE utvikling ved å tilby innebygget EJB utviklingshjelp og kodegenerering.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✓
Reengeneering av ekstisterende EJB komponenter	✓
Deployment på applikasjonsserver	✓
Kodegenerering	
EJB kode (home/remote)	✓
EJB 2.0 – Container Managed Persistence i entity beans	✗
EJB 2.0 – Bean Managed Persistence I entity beans	✗
EJB 2.0 - Foretningslogikk i sessions beans	✗
COM komponenter	✗
GUI	✗
SQL databasescript	✓

PRIS: $?

Andre egenskaper

- ?? Task Advisor er et online, steg for steg hjelpeverktøy som hjelper deg med design av EJB applikasjoner.
- ?? Applikasjonsseveruavhengighet gjør at EJBer kan deployes til nesten enhver J2EE kompatibel applikasjonsserver. Dette skjer ved hjelp av en pluginarkitektur. Arkitekturene som støttes er blant annet WebSphere, WebLogic, iPlanet, JRun, og Jboss.
- ?? Genererer HTML og Java Server Pages (JSP) for webklienten.
- ?? Integrasjon med Jasmine II

Cool:Joe 2.0 gir utvilklerne verktøy for utvikling, debugging og ferdigstillelse av J2EE kompatible javabønner som lar seg benytte på alle de ledende serverapplikasjonene. Dette gir utviklerne stor fleksibilitet ved valg av applikasjonsserverprogramvare. Med Cool:Joe 2.0 er det mulig å lage både entitetsbønner og sesjonsbønner med BMP. Den innebygde

UML-editoren støtter UML versjon 1.3 diagrammer. Dette lar utviklerne modellere brukstilfeller, sekvensdiagrammer, komponenttyper, grensesnitt, klassediagram etc. I tillegg til det overforstående har Cool:Joe en rekke avanserte wizards som øker produktiviteten. Disse verktøyene inkluderer Specification To Implementation Wizard, EJB Generation Wizard, Database Wizard, Persistence Generation Wizard, Web Application Wizard, Report Generator Wizard, Test Harness Wizard mfl. [COJ1]

Drøfting

Dessverre støttes kun EJB versjon 1.1 og ikke den nyere 2.0 standarden, noe som gjør Cool:Joe til en mindre aktuell kandidat i vårt prosjekt. Heller ikke XMI er støttet.

7.5.4 Describe

Utgitt av: **Embarcadero Technologies**
Web:
http://www.embarcadero.com
http://www.embarcadero.com/products/…
…/describe/describe_overview.htm

Produktbeskrivelse

Describe er et UML design- og utviklingsverktøy spesielt for de som jobber med Java, C++ og IDL.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✓
Reengeneering av ekstisterende EJB komponenter	✓
Deployment på applikasjonsserver	✓
Kode generering	
EJB kode(home/remote)	✓
EJB 2.0 - Container Managed Persistance i entity beans	✗
EJB 2.0 – Bean Managed Persistance i entity beans	✗
EJB 2.0 - Foretningslogikk i sessions beans	✗
COM komponenter	✓
GUI	✗
SQL databasescript	✗

PRIS: På forespørsel

Andre egenskaper

?? Dette programmet er først og fremst ment å skulle integreres med utviklingsverktøy. I utgangspunktet kan det integreres med Forte fra Sun Microsystems, men det vil etter hvert også komme støtte for JBuilder, Visual Age for Java og Visual Café.
?? Programmet har ”round trip engeneering” som vil si at det er automatisk synkronisering mellom modellen og koden.
?? Det er mulig å importere tilleggsmoduler til programmet. Disse kan være fra produsenten, tredjepartsutviklere, eller man kan lage dem selv. Man kan også legge til egne standarder for kodegenerering.
?? Det er lagt inn god støtte for flere å jobbe sammen over lokalnett eller Internet.
?? Mulighet for å integrere med ER-modelleringsverktøy (ER/Studio).

Drøfting

Describe er etterfølgeren til programmet GDPro som har vunnet flere priser, noe som skulle tyde på at det er et solid verktøy. Tanken med å knytte programmet opp mot kjente utviklingsverktøy som Forte fra Sun Microsystems er i utgangspunktet veldig god i og med at utviklerne fortsatt kan bruke det verktøyet de er vant til. Ulempen er at det er et begrenset antall verktøy som er støttet, og at de som ønsker å bruke andre ikke har særlig nytte av Describe.

Alt i alt så er inntrykket at dette er et godt, men omfattende verktøy. Med støtte for EJB 2.0 så vil det være i nærheten av det som er ønsket. Det største minuset er trolig prisen. Prisen på forgjengeren ligger på flere tusen dollar, og i tillegg kommer prisen på Forte eller et annet verktøy som brukes i tillegg.

7.5.5 EJBGen 1.21

Utgitt av: **Cedric Beust**
Web:
http://www.beust.com/
http://www.beust.com/ejbgen/

EJBGen

Produktbeskrivelse

EJBGen er en kommandobasert EJB 2.0 kodegenerator.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av ekstisterende EJB komponenter	✗
Deployment på applikasjonsserver	✗
Kodegenerering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistence i entity beans	✓
EJB 2.0 – Bean Managed Persistence I entity beans	✓
EJB 2.0 - Foretningslogikk i sessions beans	✓
COM komponenter	✗
GUI	✗
SQL databasescript	✗

PRIS: Gratis

Andre Egenskaper

Siste versjon av EJBGen er per dags dato versjon 1.21. I stedet for å måtte editere og vedlikeholde flere ulike filer (Bønna, Remote og Home klassene) lar EJBGen deg editere kun i en enkeltfil (Bønna) og kommentere denne med Javadoc informasjon. EJBGen vil deretter parse din kode og generere alle de nødvendige filene. Dette er et produkt uten noe brukergrensesnitt, der man må benytte kommandolinja med parametere for å spesifisere oppførselen til generatoren.

Drøfting

EJBGen som kandidat skiller seg ut fordi den er kommandobasert. Om ikke vi velger denne løsningen er det mulig at vi kan benytte deler av denne etersom det er opensource programvare.

7.5.6 EJBQuick

Utgitt av: **Software Associates**
Web:
http://www.software-assoc.com/
http://www.ejbquick.com/

Produktbeskrivelse

Et integrert system som prøver å tette hullet mellom design og deployment. Har ulike komponenter som generer noenlunde ferdige komponenter gitt en Rose-modell eller lignende.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av eksisterende EJB komponenter	✗
Deployment på applikasjonsserver	✓
Kodegenerering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistance i entity beans	✓
EJB 2.0 – Bean Managed Persistance i entity beans	✓
EJB 2.0 – Foretningslogikk i sessions beans	✓
COM komponenter	✓
GUI	✓
SQL databasescript	✗

PRIS: N/A Ikke til salgs i dette øyeblikk (omorganisering av produsentselskap)

Andre egenskaper

EJBQuick består av et knippe komponenter som fungerer som påbygging til Rational Rose eller lignende. De tilbyr tjenester som kodegenerering, stubbgenerering , simuleringsmuligheter, testverktøy og klient generator (JSP, Servlets) osv. Kodegenereringen baserer seg på velkjente patterns og genererer mye dekkende kommentarer i tillegg. Den komponenten som er mest relevant for dette prosjektet er EJBQuickStart.

Drøfting

Dette programmet virker som om det er i utviklingsfasen enda. Konseptet er veldig i trakt med prosjektoppgaven, og kodegenereringen løser alle de oppgavene kunden ønsket. Problemet er at på hjemmesiden virker det som dette firmaet er på veg til å legges ned, da det annonseres etter kjøpere av konsept, program og domene.

7.5.7 EJBX J2EE Code Generator

Utgitt av: **Houston Technology Group**
Web:
http://www.myhtg.com
http://www.myhtg.com/product.html

Produktbeskrivelse

Verktøy for å generere EJB-kode ut i fra en XML-beskrivelse.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✘
UML-editor	✘
Reengeneering av ekstisterende EJB komponenter	✘
Deployment på applikasjonsserver	✔

Kode generering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistance i entity beans	✔
EJB 2.0 – Bean Managed Persistance i entity beans	✘
EJB 2.0 - Foretningslogikk i sessions beans	✘
COM komponenter	✘
GUI	✘
SQL databasescript	✘

PRIS: fra kr 1800

Andre egenskaper

Programmet er designet for å enkelt kunne modifisere hvordan kodegeneratoren oversetter fra XML til generert kode for å kunne optimalisere koden eller ta høyde for eventuelle spesielle behov som måtte være til stede.

Drøfting

EJBX dekker langt på vei de grunnleggende problemstillingene som var utgangspunktet for denne undersøkelsen, men har heller ikke noe mer. Funksjonaliteten er langt under det de store programpakkene kan vise til, men det samme gjelder prisen. Inndataene til programmet er av produsenten tenkt å skulle være generert av databasen som det skal lages bønner for. Men data generert på grunnlag av en datamodell burde gjøre samme nytten.

7.5.8 Forte

Utgitt av: **Sun Microsystems**
Web:
http://www.sun.com/
http://www.sun.com/forte/

Produktbeskrivelse

Forte er en kodegenerator som er beregnet på EJB.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av ekstisterende EJB komponenter	✗
Deployment på applikasjonsserver	✓

Kodegenerering	
EJB kode(home/remote)	✓
EJB 2.0 - Container Managed Persistance i entity beans	✓
EJB 2.0 – Bean Managed Persistance i entity beans	✓
EJB 2.0 - Foretningslogikk i sessions beans	✓
COM komponenter	✗
GUI	✓
SQL databasescript	✓

PRIS: kr 17.500

Andre Egenskaper

?? Kan legge ut på iPlanet Application Server.

Drøfting

Dette programmet er veldig enkelt, men støtter få funksjoner. Det støtter bare JavaBeans og kan bare deploye til iPlanet Application Server. Det blir dermed lite egnet til det formålet som er tenkt i dette prosjektet. Samtidig støtter det ikke import av XMI.

7.5.9 Genova 7.0

Utgitt av: **Genera**
Web:
http://www.genera.no

Produktbeskrivelse

Genova er et sett av verktøy som gir støtte til overgangen fra design til implementasjon og vedlikehold av komponenter. Genova tilbyr, via en tett integrasjon med Rational Rose, generering av databaseskjema, komponentkode og GUI utifra UML.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av eksisterende EJB komponenter	✓
Deployment på applikasjonsserver	✓

Kode generering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistance i entity beans	✓
EJB 2.0 – Bean Managed Persistance i entity beans	✓
EJB 2.0 – Foretningslogikk i sessions beans	✓
COM komponenter	✓
GUI	✓
SQL databasescript	✓

PRIS: 35 000 kr.

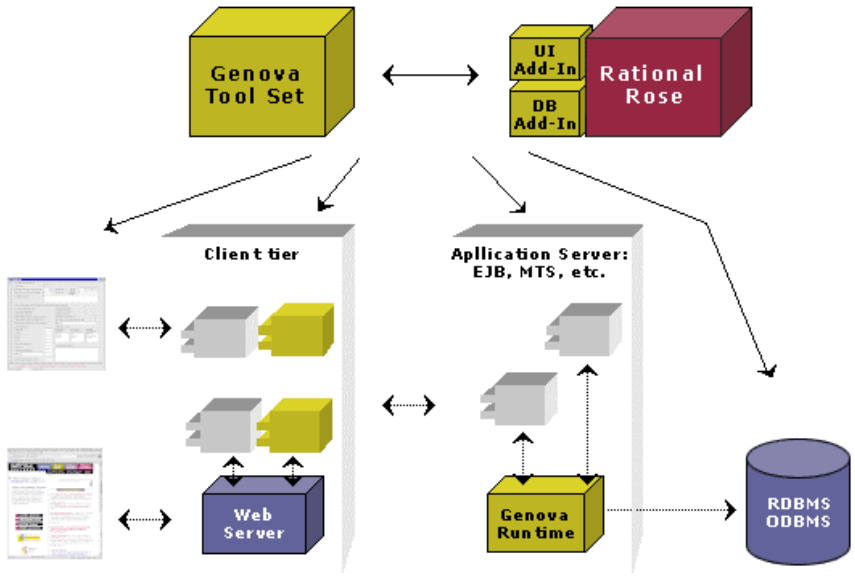
Andre Egenskaper

Ut ifra UML-modeller får man generert kode for mer komplekse databaseoperasjoner og GUI på toppen av applikasjonen, og ikke bare skjelettet av klassene som trengs. Utvikleren skal ikke behøve å bry seg om SQL, når man har valgt hvilken type database som ligger i bunnen av systemet man designer. All kode som tar seg av databasen skal abstraheres bort for utvikleren. Støtte for ulike komponentteknologier, blant annet EJB 2.0, finnes via maler.

Genova legger en del vekt på en inkrementell uvikling med bruk av prototyper. Ut ifra UML-modeller genereres GUI, hovedsaklig ment for prototyper. Hvis man likevel er fornøyd med det autogenererte brukergrensesnittet, vil så og si all manuell koding skje i applikasjonslogikken. GUI i tykk-klient-applikasjoner og websider genereres v.h.a. teknologier som Java Applets, Java Servlets, Javascript, HTML, CSS, ASP, JSP, Visual Basic og C++.

Genova gir altså støtte for generering av kode både for webserveren og klientlaget på den ene siden og applikasjonsserveren, som jobber mot databaseservereren, på den andre siden.

Målet med Genova er at i implementasjonsfasen skal utviklerne kun bry seg om applikasjonslogikken, som ligger mellom dataaksesslaget og presentasjonslaget i systemet. Kodegenereringen skjer ved hjelp av Genova tilleggsmoduler til Rational Rose, slik at man trenger en versjon av Rose for å kunne utnytte Genova fullt ut.



Figur 7.1 - Genovas rolle i utviklingen

Figuren over viser hvilken rolle Genova kan spille i utviklingen av webbaserte løsninger.

Drøfting

Genova er en meget utvidbar løsning, siden det baserer seg på maler for de teknologiene som finnes på markedet. Maler blir distribuert med jevne mellomrom fra Genera, og dette gjør at produktet raskt dekker nye teknologier, hvis firmaet fungerer som det skal. Maler brukes både for komponentteknologiene i dataaksesslaget og skriptspråkene og programmeringsspråkene i presentasjonslaget (GUI). Det finnes blant annet en egen mal for EJB 2.0.

Genova er sterkt integrert med Rational Rose og dermed også UML. UML er de facto standard modelleringsspråk, og Genova tilbyr dessuten oversettelse fra ER til UML. Tilknytningen til Rational Rose er litt mer problematisk. For å kunne utnytte fordelen med kodegenerering utifra UML, må man som sagt ha Rational Rose. UML modeller som Genova bruker må være laget i Rational Rose. Dette er først og fremst en økonomisk problemstilling, siden prisen for Genovas funksjoner kan bli mer enn tredoblet på grunn av Rose, men det går også utover fleksibiliteten. Med XMI i stor framgang som standard

for representasjon av UML blant flere samarbeidende verktøy, virker det litt gammeldags å binde seg opp mot ett produkt, selv om det er det meget omfattende Rational Rose.

Genova User Interface Designer bruker UML for å lage grensesnittmodeller. Dette er en svært nyttig funksjon i utgangspunktet, og alt beror på om resultatet er et bra grensesnitt. Genera påstår at grenssnittet er sterkt knyttet opp mot datagrunnlaget og brukstilfellene, og fokuserer på funksjonalitet, ikke fancy utseende. Dette lover bra.

En arkitektkonsulent ved Oslo Børs [GEN1] skryter av grensesnittet til Genova, og mener produktet er brukervennlig. På grunn av at produktet legger til rette for at utviklerne skal fokusere på applikasjonslogikk, og i stor grad glemme GUI-biblioteker til spesifikke programmeringsspråk, ulike SQL-dialekter og komponentteknologier, kreves det i det hele tatt lite kunnskaper knyttet til implementasjonsrelaterte finesser. Hvis produktet holder det det lover, vil utviklerne virkelig kunne fokusere på design, forretningsdomene og applikasjonslogikk.

7.5.10 LowRoad

Utgitt av: **TallSoftware**
Web:
http://www.faraway.co.uk/tallsoft/
http://www.faraway.co.uk/tallsoft/lowroad/

Produktbeskrivelse

Lowroad er en kodegenereator som er spesielt beregnet på utviklingen av EJB.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av eksisterende EJB komponenter	✗
Deployment på applikasjonsserver	✓

Kodegenerering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistance i entity beans	○
EJB 2.0 – Bean Managed Persistance i entity beans	✓
EJB 2.0 – Foretningslogikk i sessions beans	✓
COM komponenter	✗
GUI	✗
SQL databasescript	✓

PRIS: kr 250

Andre egenskaper

Et program som har sitt utspring fra et enmannsfirma i england. Genererer ferdige BMP entititetsbønner fra et standard XML dokument på et gitt format. Gir også all nødvendig SQL, og alle nødvendige deployment descriptorer for de støttede databasene.

Drøfting

Dette programmet virker litt hjemmesnekret og løser endel, men slett ikke alle problemstillingene i prosjektoppgaven. Konsekvensene av mangel på CMP-støtte er at programmet ikke er egnet for denne oppgaven. Det kan derimot være aktuelt å bruke del løsninger fra dette programmet da prisen er særdeles lav.

7.5.11 Panther for IBM Websphere

Utgitt av: **Prolifics**
Web:
http://www.prolifics.de

Produktbeskrivelse

Panther tilbyr forenklet komponentutvikling basert på UML-modeller i Rational Rose. Det er lagt vekt på kodegenerering og fleksibilitet opp mot EJB, COM osv. Bruksområdet er først og fremst transaksjonsbaserte e-business løsninger.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av eksisterende EJB komponenter	✓
Deployment på applikasjonsserver	✓

Kode generering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistance i entity beans	✓
EJB 2.0 – Bean Managed Persistance i entity beans	✓
EJB 2.0 – Foretningslogikk i sessions beans	✓
COM komponenter	✓
GUI	✗
SQL databasescript	✓

Pris: kr 44.000

Andre Egenskaper

Det er mulig å gjenbruke forretnigslogikk skrevet i C/C++ og i EJB komponenter. COM+/DNA kan brukes sammen med EJB komponenter, slik at utvikleren skal slippe å bry seg om teknologi, og fokusere på forretningslogikk. Det trengs som regel få endringer for å porte fra COM til EJB.

Drøfting

Det som imponerer mest etter et førsteinntrykk av dette produktet, er integrasjonen av forskjellige komponentteknologier. Dette gjør utviklingen lettere og raskere. Undersøkelser lagt fram av Prolifics viser at firmaer som bruker produktet deres oftere blir ferdige til planlagt tid enn andre [PRO1]. Produktet er avhengig av Rational Rose for å generere kode utifra UML, og det er et problem for fleksibiliteten. Fordelen er at produktene samlet kan bli mer brukervennlig - hvis brukerne er godt kjent med Rose.

7.5.12 Pramati Studio

Utgitt av: **Pramati**
Web:
http://www.pramati.com/
http://www.pramati.com/products/studio/overview.htm

Produktbeskrivelse

Pramati Studio forenkler oppgaven med å bygge JSP og EJB komponenter for web og forretningslogikk.

Viktige egenskaper	**Status**
XMI-import	✗
XMI-eksport	✗
UML-editor	✗
Reengeneering av ekstisterende EJB komponenter	✗
Deployment på applikasjonsserver	✓

Kodegenerering	
EJB kode (home/remote)	✓
EJB 2.0 - Container Managed Persistence i entity beans	✗
EJB 2.0 – Bean Managed Persistence I entity beans	✗
EJB 2.0 - Foretningslogikk i sessions beans	✗
COM komponenter	✗
GUI	✗
SQL databasescript	✓

PRIS: ukjent

Andre egenskaper

- ?? Smart Enterprise Class Debugger lar deg finne og fikse feil på både EJB og JSP kode. Man kan sette breakpoints og se at variablene mens programmet kjører.
- ?? Støtte for deployment til WebLogic sin webserver følger med. Utviklere kan skrive egne verktøy som leser Pramati sine XML descriptorer og konverterer dem til server-XML.
- ?? Innebygget database gjør Pramati Studio til et fullstendig utviklingsmiljø.
- ?? Pramati Studio har støtte for ANT, et javabasert make verktøy. Utviklere kan bruke ANT script for å bygge EAR filer fra kildefiler og XML filer.

Drøfting

Studio 2.5 er noe mangelfull fordi den ikke støtter EJB 2.0. Dette vil antageligvis endre seg når det dukker opp en nyere versjon. Dessverre kan man ikke lese om at noe slikt er planlagt. Selskapet Pramati lager i tillegg til Studio serverprogramvare. Denne heter Pramati Server 3.0. og støtter den nye EJB 2.0 standarden.

7.5.13 Software through pictures/UML

Utgitt av: **Aonix**
Web:
http://www.aonix.com/
http://www.aonix.com/content/products/stp/uml.html

Produktbeskrivelse

Software through pictures er et program for design av systemer i UML. Kan generere kode fra modellene.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✔
Reengeneering av ekstisterende EJB komponenter	✔
Deployment på applikasjonsserver	✘

Kodegenerering	
EJB kode (home/remote)	✔
EJB 2.0 - Container Managed Persistance i entity beans	✘
EJB 2.0 – Bean Managed Persistance i entity beans	✘
EJB 2.0 - Foretningslogikk i sessions beans	✘
COM komponenter	✔
GUI	✘
SQL databasescript	✘

PRIS: ukjent

Andre egenskaper

Ni editorer for støtte gjennom hel utviklingsprosess, med use-case, klasseeditor, sekvenseditor, samarbeidseditor, tilstandseditor, aktivitetseditor, stereotypeeditor, komponenteditor og deploymenteditor. Har utstrakt støtte for UML og et flerbrukssystem med delt lager. Bare støtte for EJB 1.1.

Drøfting

I tillegg til å støtte alle typer UML diagrammer og notasjoner, inneholder STP/UML en robust syntaks og semantisk sjekk for hvert diagram og hele modellen. Dette gjør det til en god modelleringseditor. Virker som et enkelt program som har begrenset, men bra funksjonalitet. Da med særlig tyngde mot modelleringsbiten.

7.6 Modelleringsløsninger

7.6.1 Novosoft UML Library

Utgitt av: **Novosoft**
Web: **http://www.novosoft-us.com**

Produktbeskrivelse

Novosoft UML Library er et open-source-bibliotek for å lage UML-verktøy.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✘
Reengeneering av ekstisterende EJB komponenter	✘
Deployment på applikasjonsserver	✘

Kode generering	
EJB kode (home/remote)	✘
EJB 2.0 - Container Managed Persistance i entity beans	✘
EJB 2.0 – Bean Managed Persistance i entity beans	✘
EJB 2.0 - Foretningslogikk i sessions beans	✘
COM komponenter	✘
GUI	✘
SQL databasescript	✘

PRIS: gratis

Andre egenskaper

Produktet er et bibliotek som er ment å kunne inkluderes i andre programmer, og er dermed ikke et selvstendig produkt i seg selv.

Drøfting

Dette produktet er ikke egentlig et verktøy som kan utføre kodegenerering, men et bibliotek som kan representere en fysisk metamodell for UML. Biblioteket kan inkluderes i produkter man lager selv og dra nytte av den vesle funksjonaliteten som ligger her.

7.6.2 Select Component Factory

Utgitt av: **Aonix Inc.**
Web:
www.aonix.com

Produktbeskrivelse

Select Component Factory er en produktgruppe som tilbyr designverktøy, med spesiell vekt på komponentbasert design, generering av kode og en bibliotekfunksjon for komponenter.

Viktige egenskaper	**Status**
XMI-import	✔
XMI-eksport	✔
UML-editor	✔
Reengeneering av eksisterende EJB komponenter	✔
Deployment på applikasjonsserver	✔
Kode generering	
EJB kode (home/remote)	✘
EJB 2.0 - Container Managed Persistance i entity beans	✘
EJB 2.0 – Bean Managed Persistance i entity beans	✘
EJB 2.0 – Foretningslogikk i sessions beans	✘
COM komponenter	✘
GUI	✘
SQL databasescript	✘

PRIS: ukjent

Andre Egenskaper

Utover egenskapene nevnt over, ligger det en del ekstra finesser i beskrivelsen under.

Select Component Factorys fokus på komponentbasert design, og håndtering av komponenter, dekkes av en produktgruppe med 4 integrerte verktøy:

Select Enterprise^{TM}
Miljø for modellering av forretningsmodeller, komponentbaserte modeller, datamodeller og UML. Verktøyet er skalerbart, og støtter derfor omfattende samarbeid i store firma i designfasen. Det er støtte for utveksling av modeller ved hjelp av XMI, både for eksport og import.

Select Component Manager™
Verktøy for å holde orden på ferdig utviklede komponenter og deres spesifikasjoner, for eksempel EJB og COM, i en større organisasjon. Det tilbyr versjonskontroll, enklere publisering av komponenter og gir oversikt og kontroll over avhengigheter mellom komponenter.

Reviewer for Select Enterprise™
Genererer rapporter utifra design, særlig komponentbasert design.

Select Synchronizers (JSync™, CSync™ and VBSync™)
Tilbyr toveis synkronisering mellom design og kode. Generer kode utifra design og oppdaterer design hvis man endrer kode. JSync generer Javakode utifra UML-modeller laget i Select Enterprise, og CSync generer på samme måte C++ kode.

Drøfting

Produktene i Select Component Factory tilbyr en sterk løsning for design, publisering og vedlikehold av komponenter. De er sterkt sammenknyttet og gir derfor gode muligheter for høy kobling mellom utviklingsfasene. Sammenknytningen gjør på den annen side produktene mindre fleksible, siden de hovedsaklig er ment å brukes sammen. Det er derimot gjort vellykkede forsøk med XMI som samarbeidsstandard mellom ulike verktøy, der Select Enterprise ble brukt som modelleringsverktøy. [SEL1].

JSync og de andre kodegeneratorene lager kode utifra UML-modeller, men det er ikke støtte for generering av komponenter, som f.eks. EJB eller COM. Dette er et stort minus, spesielt med tanke på vektleggingen av komponentbasert design.

8 Konstruerte løsninger

Dette kapittelet presenterer de ulike konstruerte løsningene forstudiet har funnet.

8.1 Alternativ 0 – Nåsituasjon

Dette alternativet baserer seg på at man ikke forandrer noe med dagens måte å gjøre ting på. Man fortsetter således med å klippe og lime fra tidligere prosjekter. Dette er en urealistisk løsning siden dagens måte å gjøre ting på har store svakheter.

8.2 Alternativ 1 – Utopi

Den perfekte løsning for problemstillingen finnes ikke som eksisterende løsning og blir heller ikke implementert i denne prosjektoppgaven, men kan være en realitet om ikke så alt for mange år.

Løsningen baserer seg på et system som kan importere UML-modeller på mange ulike måter, evaluere disse og til slutt generere ferdig kode i ønsket format.

Import av modeller kan typisk gjøres ved innscanning av håndtegnede modeller eller konvertering av ulike modellfiler. For konvertering av filer støttes XMI som mellomledd i tillegg til at formatene til alle de største aktørene på markedet støttes direkte for import.

Dersom modellene inneholder svakheter, feil eller inkonsistens varsles dette, og forbedringer foreslåes på grunnlag av etablerte patterns og såkalte ”best practises”. I tillegg stiller løsningen med veivisere som raskt og enkelt produserer ulike varianter av standard systemer slik som for eksempel e-handelsløsninger.

Kodegenerering kan så foretaes til valgfri arkitektur, med eventuelt valgfritt målspråk og valgbare variasjoner på fysisk utplassering. Kodestandard som det genereres etter kan i stor grad tilpasses brukerens ønsker. Alle andre nødvendige script, descriptorer og lignende genereres så automatisk på grunnlag av valgene gjort av brukeren. Alle maler og regler kan endres av brukeren ved hjelp av grafiske verktøy som gjør endringene enkle og intuitive.

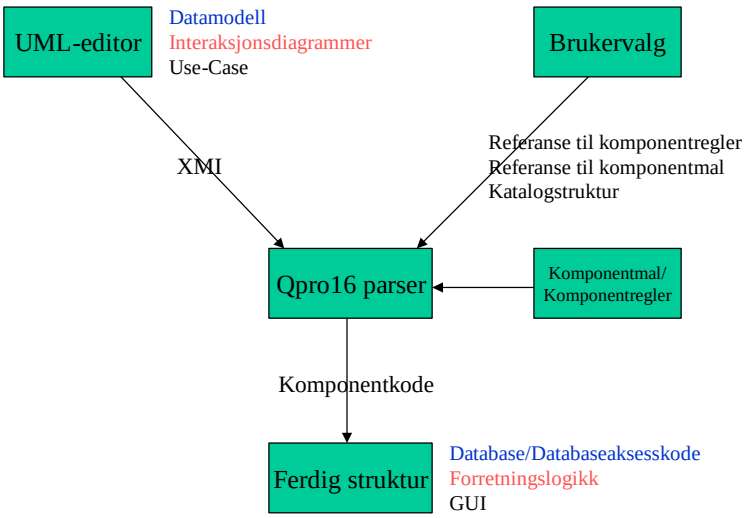
I tillegg stiller systemet med en mengde ferdige brukergrensesnitt maler som automatisk kan genereres på grunnlag av modell eller valg gjort i veivisere. Dette medfører at en prototype kan være oppe å kjøre på veldig kort tid.

8.3 Alternativ 2 – En frittstående parser

Dette alternativet baserer seg på at det skal konstrueres en parser som tar XMI som inndata og har kode som utdata.

8.3.1 Konsept

XMI definisjonene og dokumentene definerer UML-modeller, og koden er basert på disse modellene. Løsningen skal løse alle deler av oppgaven formulert i kapittel 4, Finformulering av oppgaven. Koden skal som kjent være utganspunkt for en komponentbasert løsning, med EJB, COM eller lignende.



Figur 8.1 - Oversikt over systemet.

I figur 8.1 utgjør Qpro16 parseren løsningen som skal konstrueres.

Inndata:

- ?? XMI-representasjon av UML-modeller
- ?? Parametre basert på brukervalg
  - Komponentmal: Bruker velger hvilken komponentteknologi som skal ligge til grunn for den genererte koden. Parameteren er en referanse til en komponentmal, som fastlegger klassestrukturen for den genererte koden.
  - Komponentregler: Referanse til et sett regler, for hva parseren skal gjøre ut ifra forskjellige strukturelle elementer i XMIen.
  - Katalogstruktur: Bruker velger hvilken katalogstruktur som den genererte koden skal ligge på, ved å referere til en fil som beskriver strukturen.

Utdata:

- ?? Databaseskript i SQL for oppretting av databasetabeller og komponenter for databaseaksess. Basert på UML-datamodell.
- ?? Skjelettkode for forretningslaget, som inneholder klasser, metoder og metodekall basert på UML-interaksjonsdiagram.
- ?? Grafisk brukergrensesnitt basert på funksjonalitetet i UML use-case.

8.3.2 XMI/UML

Løsningen er avhengig av å få inn UML-modeller fra et eksternt UML modelleringsverktøy. Verktøyet må kunne eksportere XMI. Det som lagres under ekporteringen må være en XMI DTD som definerer elementene som trengs for å representere UML modellene, og et XMI dokument som beskriver modellene basert på DTDen. Eksempel på dette finnes i kapittel 5.3 XMI. Hvis UML-verktøyet lager generelle DTDer, som for eksempel inneholder en tag for å definere en vilkårlig tabell i en datamodell, må elementene i DTD'ene være kjent for utviklerne og brukes i parseren. Med denne metoden bruker parseren og UML-verktøyet XMI-DTDen som protokoll for overføringen av en modell, mens XMI-dokumentet er selve modellen.

8.3.3 Komponentteknologi

Bruker velger om han vil ha for eksempel EJB eller COM. De forskjellige teknologiene krever ulike klasser og grensesnitt, og det brukes ulike programmeringsspråk i de forskjellige teknologiene. Det eneste som trengs for å generere riktig kode for den ønskede komponentteknologien, er en komponentmal og et sett komponentregler. Komponentmalen og reglene lages av utviklerne, men brukeren velger hvilke som skal brukes. En annen mulighet, som ikke tas opp nærmere her, er at brukeren har mulighet til selv å lage nye maler og regelsett.

Komponentmal

- ?? Følger et bestemt programmeringsspråk.
- ?? Inneholder skjelett av klasser og grensesnitt som skal genereres.
- ?? Inneholder skjelett av tilleggsfiler som skal genereres, for eksempel deployment descriptor i EJB.

Komponentregler

Regler for hva parseren skal gjøre ved tager, definert i DTDen, brukt i XMI-dokumentet. Tagene som er definert i DTDen må være kjent i regelsettet, slik at parseren vet hva den kan støte på i XMI dokumentet.

8.3.4 Eksempel

I dette eksempelet antar vi at brukeren har sendt inn en XMI-modell fra et UML-verktøy, valgt komponentmal og regelsett og katalogstruktur. Komponentmalen og regelsettet som er valgt er tilpasset EJB 2.0.

I XMI-DTDen (se kapittel 5.3 XMI) er det definert et element ved navn ”Table”, som representerer en entitet i en datamodell. Det er også definert et ”Column”-element. XMI dokumentet definerer elementer av typen ”Table”, og hver ”Table” inneholder elementer av typen ”Column”.

I regelsettet er det definert hva som skal gjøres når parseren støter å et ”Table”-element. Da skal det lages en enitetsbønne som tar seg av databaseaksess til denne tabellen. Når parseren er inne i Table-elementet vil den støte på elementer som hører til tabellen, f.eks. attributter. Ved et ”Column”-element skal parseren lage metoder for å hente ut verdier i kolonnen, eller attributtet.

8.4 Alternativ 3 – En tilleggsmodul til en eksisterende løsning

Det konstrueres en tilleggsmodul til en eksisterende løsning som kompletterer denne i henhold til kundens ønsker.

Konstruksjonen blir forholdsvis lik den gitt i alternativ 2, en frittstående parser, bortsett fra at man nå knytter genereringen opp mot et spesielt program og parser dette programmets modellrepresentasjon isteden for XMI før man generer kode.

9 Evaluering

Det er nå samlet inn info om en rekke kodegeneratore og modelleringsverktøy. Disse vil bli evaluert i dette kapittelet. Først kommer litt om hvordan evalueringen skal gjøres, og til slutt resultatene.

9.1 Innledning

Gjennom markedsundersøkelsen ble det funnet et stort antall eksisterende løsninger som kunne utgøre en hel eller delvis løsning for dette prosjektet. Dette medfører at det må utføres en grovsortering for å finne de beste eksisterende løsningene, som så taes med videre i evalueringsprosessen.

Når antallet eksisterende løsninger er redusert betraktelig, vil det gjennomføres en grundig evaluering av disse. Samtidig vil de konstruerte løsningene evalueres på samme måte. Utfallet av denne prosessen vil ligge til grunn for vårt valg av løsning.

9.2 Maler for evaluering

Malene for de ulike evalueringene ble definert og endelig fastlagt i god tid før selve evalueringene. Dette ble gjort for å oppnå en mest mulig objektiv og rettferdig evalueringsprosess. Malene baserer seg i stor grad på kundens ønsker og mål med prosjektet, og malene har vært godkjent av kunden før noen form for evaluering startet.

9.2.1 Malen benyttet i grovsorteringen av de eksisterende løsningene

For å gjøre denne enkel og oversiktlig ble det brukt en vekting av standardtabellen som alle eksisterende løsninger er presentert med. De ulike egenskapene blir vektet i henhold til kundens prioriteringer og preferanser.

En løsning kunne maksimalt oppnå 100 poeng dersom den inneholdt alle de ønskede egenskapene fra kunden. Evalueringen tok i tillegg hensyn til planlagde utvidelser av de eksisterende løsningen. Ønskede egenskaper som var planlagt, men ikke implementert i nåværende versjon, gav halv poeng sum for egenskapen. Evalueringen vektla pris slik at rimelige løsninger ble høyt belønnet.

Målet med å velge denne fremgangsmåten for grovsorteringen er å finne de eksisterende løsningene som innehar akkurat de egenskapene som dette prosjektet etterspør. Disse er da de mest interessante eksisterende løsningene å ta med videre i en grundigere evaluering opp mot de konstruerte løsningene.

Poengfordelingen i standardtabellen som de eksisterende løsningene ble presentert med er fastlagt som:

Viktige egenskaper (max 30 p)	**Eksisterer***	**Er planlagt***
XMI-import	12 p	(6 p)
XMI-eksport	4 p	(2 p)
UML-editor	8 p	(4 p)
Reengeneering av eksisterende EJB komponenter	3 p	(1 p)
Deployment på applikasjonsserver	3 p	(1 p)
Kode generering (max 60 p)		
EJB kode (home/remote)	10 p	(5 p)
EJB 2.0 - Container Managed Persistance i entity beans	22 p	(11 p)
EJB 2.0 – Bean Managed Persistance I entity beans	12 p	(6 p)
EJB 2.0 – Foretningslogikk i Session beans	8 p	(4 p)
COM komponenter	3 p	(1 p)
GUI	2 p	(1 p)
SQL databasescript	3 p	(1 p)

PRIS:	Pris<500 NOK = 10 p	Pris<10 000 NOK = 4 p
(max 10 p)	Pris<1 000 NOK = 9 p	Pris<15 000 NOK = 3 p
	Pris<2 000 NOK = 8 p	Pris<25 000 NOK = 2 p
	Pris<3 000 NOK = 7 p	Pris<50 000 NOK = 1 p
	Pris<5 000 NOK = 6 p	Pris>50 000 NOK = 0 p
	Pris<7 500 NOK = 5 p	

* Enten eller. En løsning kan ikke ha begge, enten eksisterer egenskapen eller så er den planlagt eller ingen av delene.

Tabell 9.1 - Vektleggingen av presentasjonsmatrisen for de eksisterende løsningene

9.2.2 Malen benyttet i den grundige evalueringen

Den grundige evalueringen går mye mer i dybden på løsningene. De eksisterende løsningene prøves ut i praksis, og de konstruerte løsningene gjennomsøkes etter svakheter og forskjeller.

Den grundige evalueringen har som forutsetning at løsningen som skal evalueres har UML-editor eller XMI-import og kan generere kode for entititets- og sessjonsbønner som følger EJB 2.0 spesifikasjonen.

Evalueringen tar spesielt for seg 4 aspekter av hver løsning som vektes forskjellig. Et system som fullt ut oppfyller alle krav i alle aspekter oppnår 30 poeng. I tillegg gis det en poengsum på inntil 10 poeng som beskriver helhetsinntrykket testeren sitter igjen med. Dette medfører at en løsning maks kan oppnå 40 poeng.

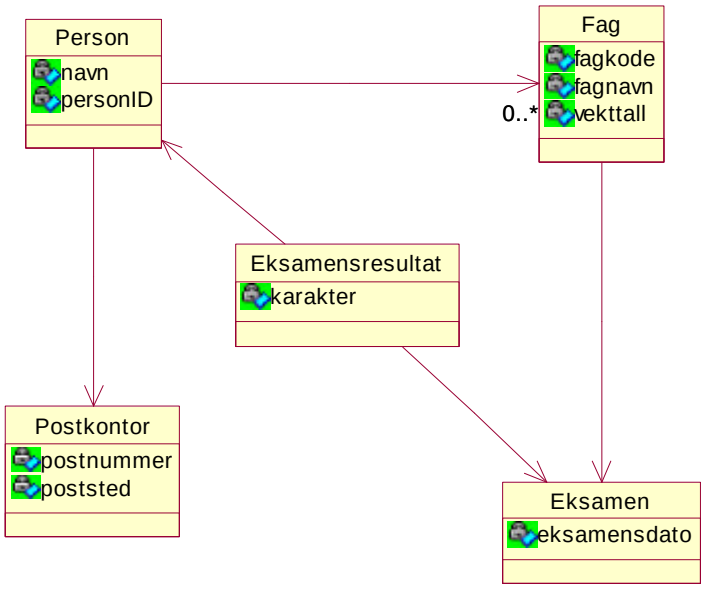
De ulike evaluerings aspektene med vektlegging er:

1. Kvalitet på generert kode	14 poeng
2. SQL script for de mest brukte DBMS	7 poeng
3. Støtte for andre arkitekturer	5 poeng
4. Ressurskrav	4 poeng
5. Helhesinntrykk	10 poeng

Tabell 9.2 - Evalueringsaspekter med poengfordeling

Gjennom evalueringen av aspekt 1, kvalitet på generert kode, skal det legges vekt på elementer som kodestandard, kommentarer og deployment descriptorer. Kodestandarden bør være mest mulig lik standarden gitt av Sun. [GEV1] Kommentarer til koden bør genereres automatisk som Javadoc, og skal bedømmes på standard oppsett og innhold. Automatisk generering av deployment descriptorer er også et viktig moment.
I tillegg tas støtte for andre språk med her.

For å vurdere den genererte koden taes det utgangspunkt i følgende datamodell.



Figur 9.1 - Datamodell som er grunnlag for generert kode

Koden generert av hvert av programmene for entiteten *Person* er lagt ved i vedlegg 5. På bakgrunn av dette kan koden fra de ulike programmene sammenlignes direkte og vurderes opp mot hverandre og standarden fra Sun. [GEV1]

Evalueringen av aspekt 2, SQL-script for de mest brukte DBMS, legger vekt på støtte for databasene MS SQL Server, Sybase, Oracle, DB2, MySql og PostgresSql.

Støtte for andre arkitekturer, aspekt 3, legger primært vekt på COM/COM+, men tar også hensyn til eventuelle andre arkitekturer som er støttet, herunder CORBA og .NET.

Aspekt 4, ressurskrav, legger vekt på systemets krav til CPU, minne og harddiskplass.

I helhetsinntrykket skal det legges vekt på elementer som brukervennlighet, terskel for bruk, layout, tidsforbruk, og praktisk bruk. Samt hvilke konsekvenser innføringen av løsningen vil føre til. I tillegg skal det her legges vekt på fleksibilitet og utvidbarhet.

9.3 Grovsortering av de eksisterende løsningene

Denne sorteringen er først og fremst ment for å sile ut de programmene som ikke har nødvendig funksjonalitet. Denne silingen gjøres kun ut fra produktenes spesifikasjoner, og uten noen form for direkte testing. Totalvurdering av produkter kommer først i neste evalueringsrunde. I tabell 3 er resultatet av grovsorteringen gitt.

Tegnforklaring til resultatsmatrisen:

- ✔ Funksjonaliteten er støttet
- ✘ Funksjonaliteten er ikke støttet
- ○ Funksjonaliteten er planlagt støttet

Produktnavn	XMI imp.	XMI eksp.	UML	Re.eng.	Depl.	EJB 1.1	CMP	BMP	Sess.	COM	GUI	SQL	Pris	SUM
Vekter	12	4	8	3	3	10	22	12	8	3	2	3		
Totalløsninger														
Rational Rose 2001	↘	↘	↘	↘	↘	↘	↘	↘	↘	↘	↘	↘	1	**91**
Together Control Center	↘	↘	↘	✗	↘	↘	↘	↘	↘	↘	↘	↘	0	**87**
StructureBuilder	↘	↘	↘	↘	↘	↘	↘	↘	↘	✗	↘	✗	2	**86**
SoftModeler 3	↘	↘	↘	↘	○	↘	○	○	○	✗	↘	↘	4	**68**
ArcStyler	↘	↘	↘	↘	↘	↘	✗	✗	↘	✗	↘	✗	0	**50**
ObjectIF	✗	✗	↘	✗	✗	↘	✗	✗	✗	↘	✗	↘	0	**24**
Genereringsløsninger														
Genova 7.0	✗	✗	✗	↘	↘	↘	↘	↘	↘	↘	↘	↘	1	**67**
Panther	✗	✗	✗	↘	↘	↘	↘	↘	↘	↘	✗	↘	1	**65**
EjBQuick	✗	✗	✗	↘	↘	↘	↘	↘	↘	↘	↘	✗	0	**63**
EJB Gen 1.21	✗	✗	✗	✗	✗	↘	↘	↘	↘	✗	✗	✗	10	**62**
Forte	✗	✗	✗	✗	↘	↘	↘	↘	↘	✗	↘	↘	0	**60**
EJBX J2EE CG	↘	✗	✗	✗	↘	↘	↘	✗	✗	✗	✗	↘	8	**58**
LowRoad	✗	✗	✗	✗	↘	↘	○	↘	↘	✗	✗	↘	10	**57**
Avantis	✗	✗	✗	✗	✗	↘	↘	↘	✗	✗	✗	↘	0	**47**
Software through pictures	↘	↘	↘	↘	✗	↘	✗	✗	✗	↘	✗	✗	0	**40**
CocoBase	↘	✗	✗	✗	↘	↘	✗	✗	✗	✗	↘	↘	1	**31**
Cool:Joe 2.0	✗	✗	↘	↘	↘	↘	✗	✗	✗	✗	✗	↘	0	**27**
Describe	✗	✗	↘	↘	↘	↘	✗	✗	✗	↘	✗	✗	0	**27**
Pramati Sudio	✗	✗	✗	✗	↘	↘	✗	✗	✗	✗	✗	↘	0	**16**
Modelleringsløsninger														
Select Component Factory	↘	↘	↘	↘	↘	✗	✗	✗	✗	✗	✗	✗	0	**30**
Novosoft UML Libary	↘	↘	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	10	**26**

Tabell 9.3 - Resultatet av grovsortering

9.3.1 Oppsummering

De tre produktene som oppnår høyest poengsum er:

1. Rational Rose, 91 poeng
2. Togehter Control Center, 87 poeng
3. StructureBuilder, 86 poeng

Av disse tre har de to beste alt det som vi evaluerer etter. De har i tillegg ekstrafunksjonalitet som gjør dem til svært gode verktøy, både til å løse den oppgaven dette prosjektet beskriver samt annen modellering og kodegenerering. Det siste programmet har også det meste av den funksjonaliteten vi er ute etter, men mangler støtte for COM og generering av SQL-skript. Da hovedtyngden i prosjektet er mot å kunne generere EJB, taes likevel også dette produktet med i videre evaluering.

Man har også muligheten til å kombinere løsninger fra kategori 2 og 3 som kan utføre hele den ønskede oppgaven. Kravet til en slik løsning er at import av XMI støttes i generatoren, og eksport av XMI støttes av modellatoren. Den beste kombinsajonsløsningen er:

EJBX J2EE CG	58
Select Component Factory	30
SUM	65 (når duplikater er fjernet)

Denne poeng summen er ikke høy nok til å kunne konkurrere med de beste totalløsningene. Derfor vil heller ingen slike løsninger bli tatt med til neste evalueringsrunde.

9.4 Grundig evaluering

De løsningene som er evaluert her tilfredstiller de kravene vi satte fram i kapittel 9.2.2. De skal ha UML-editor eller XMI-import, samt kunne generere kode for entitets- og sesjonsbønner som følger EJB 2.0 spesifikasjonen. Hvert produkt er evaluert opp mot de kriterier som er skissert i Tabell 3 Resultatet av grovsorteringen.

De eksisterende løsningene som er med i den grundige evalueringen er de tre som skilte seg ut i grovsorteringen. Disse løsningene ble opprinnelig presentert i kapittel 7.

De konstruerte løsningene som er med i den grundige evaluering er de to alternativene som er mulig å gjennomføre fra kapittel 8.

Evalueringen er gjort i denne rekkefølgen:

1. Eksisterende løsninger
   - ?? Rational Rose
   - ?? Together Control Center
   - ?? StructureBuilder

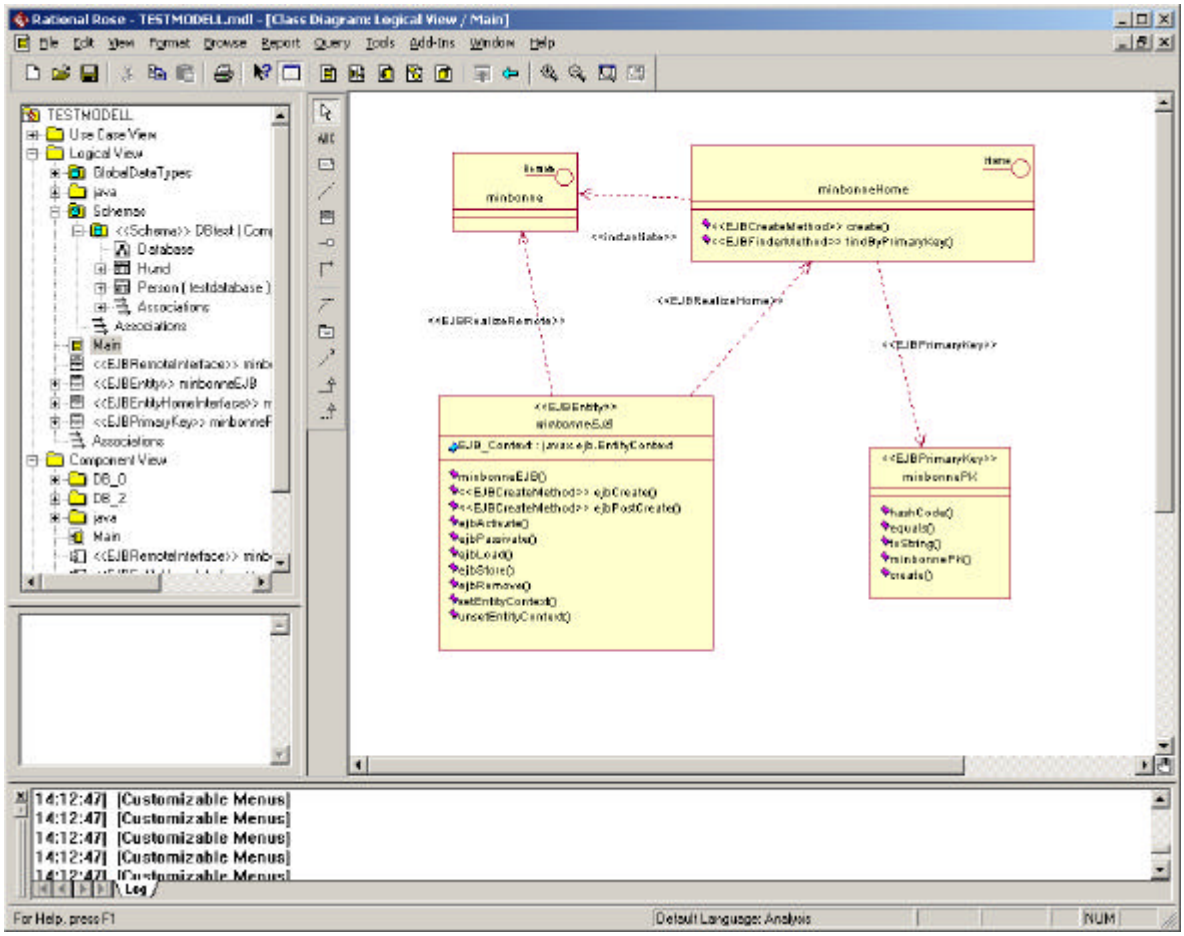
2. Konstruerte løsninger
   ?? En frittstående parser.
   ?? En tilleggsmodul til en eksisterende løsning.

9.4.1 Eksisterende løsninger

Alle de eksisterende løsningene som blir evaluert har vært gjennom grovsorteringen og tilbyr dermed mye av den funksjonalitet som dette prosjektet etterspør.

Rational Rose 2001

Dette er et veldig omfattende program, og de ønskede egenskapene til prosjektet er bare en liten del av alle de tjenestene dette programmet tilbyr. Programmet er så stort at det kan virke litt vanskelig å bruke, men når man først er kommet inn i det går det raskt og effektivt å modellere store systemer. Eksport og import av XMI blir gjort ved hjelp av en gratis tilleggsmodul som er tilgjengelig fra Rational sine hjemmesider.



Figur 9.2 - Hovedskjermbildet i Rational Rose

Kodegenerering
Rose har veldig god kodegenerering som støtter mange språk, herunder Java, EJB, C++ og ADA for å nevne noen. For EJB kan man enkelt konstruere en bønne i modellen ved å gjøre et sett av valg. Programmet støtter både EJB 1.1 og 2.0 og både CMP, BMP og sesjonsbønner. Se for øvrig [ROS4] for mer informasjon.

Når man så har fått opp modellen kan man generere kode fra modellen ved noen få museklikk. Koden som genereres er av veldig god kvalitet, og man kan selv stille inn hvilken kodestil man ønsker. I tillegg kommer Rational Rose med muligheten for automatisk synkronisering slik at enhver endring i koden gjenspeiles i modellen. Resultatet av kodegenereringen av test caset er vedlagt i vedlegg 5, og her ser man de overraskende gode kommentarene som ble automatisk generert til hver metode.

Poengsum: 14 av 14.

Database
Rose støtter mange ulike versjoner av IBM DB2, Oracle, Sybase, MS SQL Server, samt standarden ANSI SQL 92 for automatisk skript generering. Datamodelleringen er vanskelig å komme inn i, men fungerer veldig bra dersom man på forhånd er vant med Rose. En fordel er at samme data modell kan genereres for flere ulike DBMS uten å måtte endres, man endrer kun måldatabase. I Rational Rose har man også prøvd å integrere databasekonstruksjonen tettere opp mot resten av programutviklingen ved å kunne knytte tabeller og lignende til objekter.

Poengsum: 6 av 7. Får trekk for å ikke støtte "open source" databasene MySql og PostgreSql eksplisitt.

Støtte for andre arkitekturer
Støtter CORBA og COM i tillegg til EJB.

Poengsum: 4 av 5. Får trekk for å ikke støtte .NET

Ressurskrav
Rational Rose er stort, og selv om kravet til hardiskplass er oppgitt til 200 MB er Enterprise Edition på ca 400 MB. Rational oppgir også en minimum klokkefrekvens på 200 MHz og minimum prosessorkrav til en vanlig Pentium prosessor.

Poengsum: 3 av 4. Trekk for stor hardiskplass, og litt optimistiske krav fra Rational

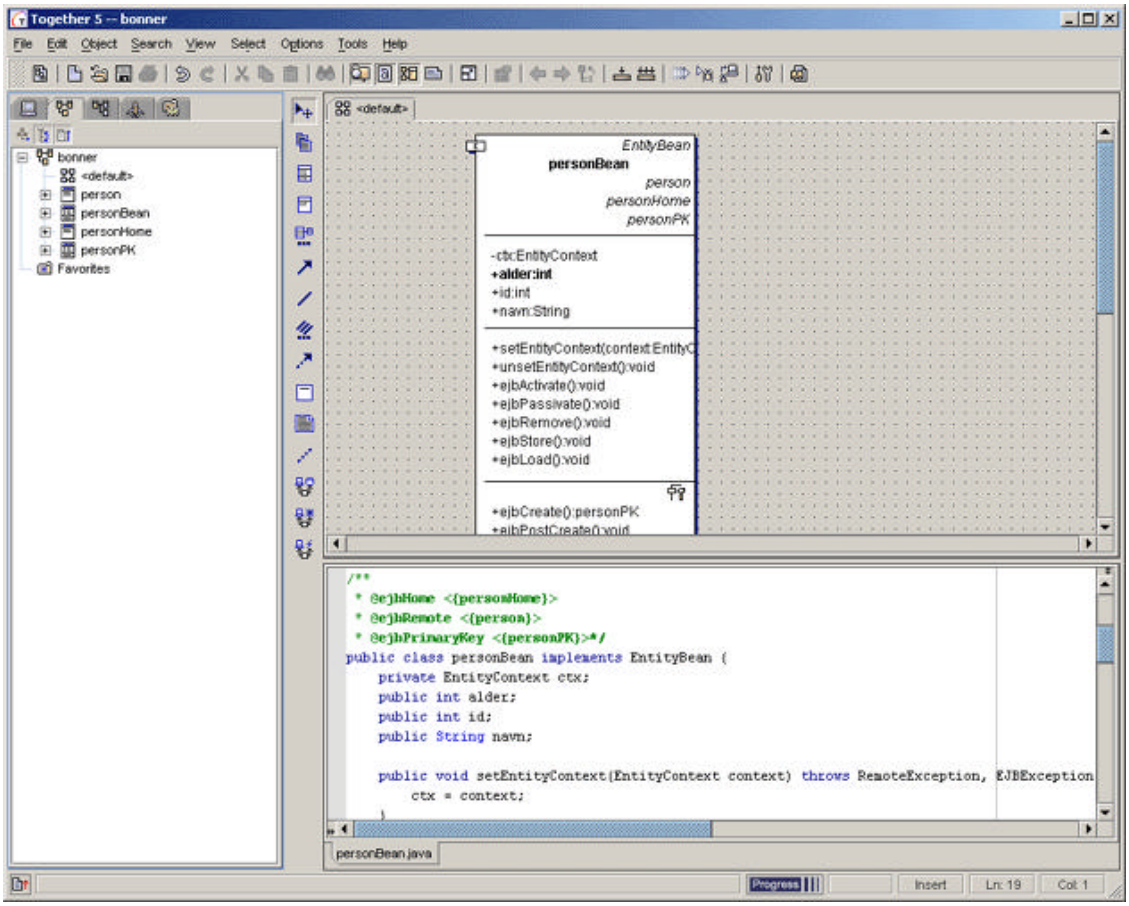
Helhetsinntrykk
Rational Rose er utrolig innholdsrikt og inneholder alle modelleringsegenskaper som behøves ved bruk av UML. Programmet har spesielt god støtte for OOAD og Java [ROS2], men støtter også annen type utvikling helt frem til kodegenerering. Det eneste problemet med Rational Rose er at omfanget medfører at det har blitt et vanskelig program å sette seg inn i, og det kreves lang tid for å beherske verktøyet slik som det er tenkt brukt. Dette medfører at for nye brukere kan Rational Rose bli et hinder i stedet for et hjelpemiddel i utviklings prosessen. I tillegg må det nevnes at prisen er forholdsvis høy, både på programmet og de kursene i bruk av programmet som tilbys.

Poengsum: 6 av 10. Trekkes for høy brukerterskel, og høy pris på produkt og kurs.

Samlet poengsum for Rational Rose: 33 av 40 poeng

Together Control Center v5.0

Together Control Center er et omfattende program for modellering og kodegenerering. Det støtter hele utviklingsprosessen fra modell til kode, og har mange funksjoner for å gjøre denne veien enklest mulig.



Figur 9.3 - Skjermbilde fra Togehter Control Center

Kodegenerering

Man stiller enkelt og greit opp den modellen man vil generere kode fra. Man velger hvilket språk som skal genereres, og så ordner programmet resten. Det er støtte for C++, Java og Visual Basic.

Man kan også skrive inn kode manuelt samtidig som man har oppe modellen. Modellen oppdateres hvis det gjøres endringer i koden og omvendt.

Kvaliteten på den genererte koden er veldig god. Det er ikke mye unødvendig data, og den er oversiktlig. Det følges gitte standarder, slik at koden er lik det meste av annen kode, og dermed lettere å lese.

Resultatet fra kodegenereringen av testmodellen er vedlagt i vedlegg 5, og dette resultatet viser at koden mangler kommentarer, men har innspill av logikk ferdig implementert.

Poengsum: 13 av 14. Får trekk for manglende kommentarer i generert kode.

Database

Man kan fra Control Center automatisk generere SQL-script for å opprette database med de elementer som er ønskelig. Man kan enten modellere databasen i et relasjonsdiagram og så generere skript ut fra det, eller lage skript direkte ut fra en UML-modellen. Hvis man for eksempel holder på å modellere EJB, kan man generere SQL-script ut fra hvilke variabler som er med i entitetsbønnene. Det er støtte for alle de fleste store DBMS.

Poengsum: 6,5 av 7. Får trekk for å ikke støtte databasen PostgreSql eksplisitt.

Støtte for andre arkitekturer

Støtter COM, VB.net og Corba.

Poengsum: 5 av 5.

Ressurskrav

Control Center er ganske tungdrevet. Man bør ha en prosessor på opp mot 1GHz, samt helst 512 Mb minne. Trenger ca 200 Mb harddiskplass.

Poengsum: 1 av 4. Det trekkes mange poeng for at programmet krever så kraftig maskinvare.

Helhetsinntrykk

Det tar ikke lang tid å komme i gang med Control Center. Ved hjelp av veivisere gikk det kjapt å komme inn i gangen i programmet. [TCC1] Brukergrensesnittet er oversiktlig og greit, og det går kjapt å navigere seg rundt. Det er veldig mange funksjoner i programmet, men menyene er bygd opp slik at man ikke går seg bort. Hvis man utvikler EJB er det veldig enkelt og greit å deploye de ferdige bønnene. Man bare velger hvilken server man skal legge de ut på, og så ordner programmet kode som støtter akkurat den serveren.

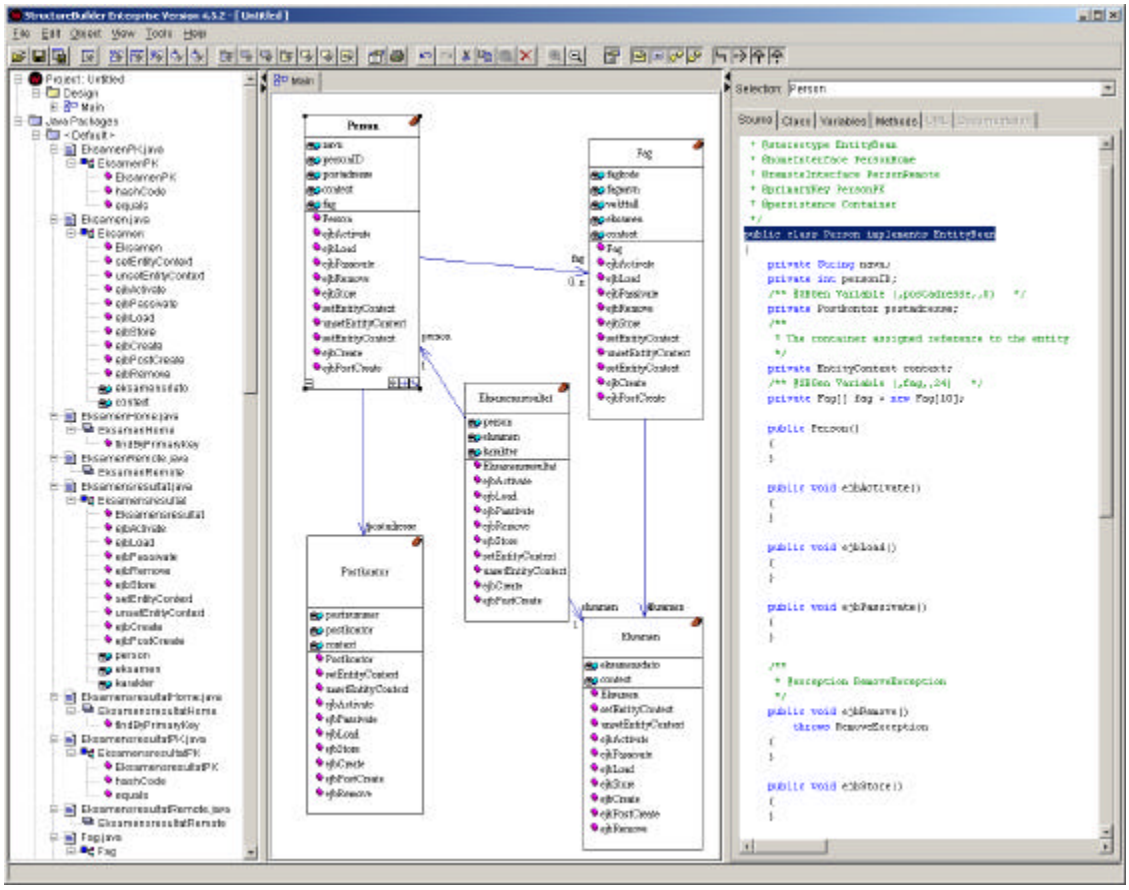
Det som er negativt med programmet er at man ikke kan hente inn allerede utviklet kode. Man kan kun hente inn XMI samt å importere fra Rose. Det vil si at for å kunne endre på eksisterende kode er man avhengig av et annet program. Programmet har også en ganske stiv pris, ca. 55.000 kroner, og det trekker litt ned.

Poengsum: 7 av 10. Trekkes for stiv pris, samt dårlig støtte for reengeneering.

Samlet poengsum Together Control Center: 32,5 av 40 poeng

StructureBuilder

Førsteinntrykket av StructureBuilder er at det er et oversiktlig og enkelt program å komme i gang med. Man kan være i gang med å tegne UML-diagrammer to minutter etter at man starter programmet for første gang. Likevel ser det ut til å være et program med stor funksjonalitet.



Figur 9.4 - Skjermbilde fra StructureBuilder

Kodegenerering

StructureBuilder genererer automatisk java-kode. Koden blir generert etter hvert som man lager modellen slik at koden og modellen alltid er synkronisert. Koden som blir generert har god kvalitet. Den er oversiktlig, og er lett å lese og modifisere.

For å lage EJB trenger man en datamodell. Deretter er det bare noen få museklikk som skal til før man har opprettet alle klasser som trengs. Et stort minus var likevel at den genererte koden ikke kompilerte, og det var lite kommentarer i koden. Koden som ble generert er vedlagt i vedlegg 5.

Poengsum: 12 av 14. Får trekk fordi koden ikke kompilerte og lite kommentarer

Database
StructureBuilder lager ikke databasescript for å opprette databaser.

Poengsum: 0 av 7.

Støtte for andre arkitekturer
StructureBuilder har ikke støtte for andre arkitekturer enn Java/EJB.

Poengsum: 0 av 5.

Ressurskrav
StructureBuilder virker overraskende lettkjørt. Systemkravene er oppgitt til å være en Pentium 133MHz med 32 Mb RAM. Riktignok kreves det at man har Windows NT 4.0 eller Windows 2000, så begrensningene til maskinvare ligger her og ikke i programmet selv. Programmet krever ca. 40 Mb diskplass.

Poengsum: 4 av 4.

Helhetsinntrykk
Det går utrolig fort å komme i gang og få brukt programmet. Menysystemet er oversiktlig og greit å finne fram i. Funksjonaliteten er nok et par hakk under det man får med de større og tyngre programsystemene, men til gjengjeld er også prisen litt lavere. Eksport og import av XMI-filer er gjort på et øyeblikk med et enkelt menyvalg. Hjelpefunksjonen i programmet er laget som websider, og har dermed sterkt begrenset søkefunksjonalitet. Dette kan føre til at det ikke alltid er like lett å finne hjelp når det trengs.

Pengsum: 8 av 10. Får trekk for tungvint hjelpefunksjon

Samlet poengsum StructureBuilder: 24 av 40 poeng

9.4.2 Konstruerte løsninger

Med konstruert løsning menes *egen* løsning, en løsning som må konstrueres i dette prosjektet. De konstruerte løsningene tilbyr fleksibilitet og utvidbarhet i større grad en de eksisterende løsningene, men må til gjengjeld utvikles og vedlikeholdes på brukernes regning. Dette er litt av det velkjente dilemmaet mellom skreddersøm og hyllevare.

Da nåsituasjon og utopi ikke er aktuelle som løsninger for prosjektets videre fremgang taes de ikke med i evalueringen, slik at de to konstruerte løsningene som taes med her er en frittstående parser og en tilleggsmodul til en eksisterende løsning.

En frittstående parser

Denne konstruerte løsningen, som er beskrevet i kapittel 8.3 En frittstående parser, vil bli et enkelt program i forhold til de eksisterende løsningene som er evaluert tidligere. Parseren vil kun tilby kodegenerering på grunnlag av en XMI-modell og noen enkle valg gjort av brukeren. Denne enkle løsningen vil likevel oppfylle alle egenskaper som er etterspurt i kundens krav og kriterier, hvis den konstrueres optimalt.

Kodegenerering

Denne løsningen vil støtte kodegenerering for alle språk som det er definert maler og regler for. Dersom strukturen på malene og reglene er generelle nok vil denne løsningen dermed kunne generere kode for alle språk i henhold til den standard som brukeren har spesifisert.

Poengsum: 13 av 14. Trekk for kompleks spesifisering av regler ved generering i hittil ubrukte språk.

Database

Parseren vil tilby generering av et SQL-skript av modellen. Når det gjelder eksplisitt støtte for ulike DBMS vil dette kunne genereres ved hjelp av maler og regler på samme måte som for kodegenereringen.

Poengsum: 6 av 7. Trekk for at ikke det finnes ferdige maler og regler for de mest brukte DBMS, men at disse må spesifiseres etter behov.

Støtte for andre arkitekturer

Da all generering er regel- og malbasert kan kode for mange ulike arkitekturer genereres av en slik parser. Det eneste minuset er at for hver ny arkitektur må brukeren selv lage regler og maler.

Poengsum: 4 av 5. Trekk for kompleks spesifisering av regler og maler for nye arkitekturer.

Ressurskrav

Parseren er tenkt implementert i Java, og størrelsen på programmet blir veldig liten i forhold til de andre løsningene. Krav til minne og lagringsplass blir minimalt sett i forhold til alle eksisterende løsninger.

Poengsum: 4 av 4.

Helhetsinntrykk
Dersom regel- og malformatet til denne løsningen blir designet generelt nok, samt at støtten for XMI blir optimal vil denne løsningen bli veldig god. Den tilbyr stor fleksibilitet og utvidbarhet, men vil ha et mye mer komplekst grensesnitt enn de ferdige løsningene. Utviklingskostnadene blir lave, i og med at dette prosjektet er gratis, men videreutviklingsutgiftene kan bli forholdsvis store og prosessen kan raskt bli tidkrevende.

En svakhet med denne løsningen er at den er avhengig av en ekstern UML-editor dersom ikke XMI-dokumentet skrives for hånd av en utvikler.

Poengsum: 7 av 10. Trekk for mulig komplisert videreutvikling og bruk, samt avhengighet av en ekstern UML-editor.

Samlet poengsum En frittstående parser: 34

En tilleggsmodul til en eksisterende løsning
Denne konstruerte løsningen, som beskrevet i kapittel 8.4, har mye felles med den forrige løsningen. En tilleggsmodul vil parse et gitt programs modellrepresentasjon isteden for det mer generelle XMI. En slik løsning kan, dersom man velger rett program å skape tilleggsmodulen for, oppfylle alle kundes krav og kriterier.

Kodegenerering
Alle språk vil kunne bli støttet, men man må lage egne maler og regler for hvert enkelt språk. Det vil kunne bli en veldig fleksibel og grei løsning så lenge koblingen mot den eksisterende løsningen fungerer optimalt. Kvaliteten på den genererte koden vil også bli svært god så lenge man følger standarder, og modellen i modelleringsprogrammet støtter alle nødvendige elementer.

Poengsum: 12 av 14. Trekk for at reglene må spesifiseres av brukeren ved generering i hittil ubrukte språk, samt begrensninger som eventuelt følger av den eksisterende løsningen.

Database
Tilleggsmodulen vil tilby generering av et SQL-skript fra modellen i den eksisterende løsningen. Eksplisitt støtte for ulike DBMS kan genereres ved hjelp av maler og regler på samme måte som for kodegenereringen. Siden det som oftest ikke er så kompleks kode som skal til for å generere ønsket database, vil det være kjapt å legge til støtte for nye DBMS.

Poengsum: 5 av 7. Trekk for at ikke det finnes ferdige maler og regler for de mest brukte DBMS, men at disse må spesifiseres etter behov, samt at man er avhengig av en annen løsing.

Støtte for andre arkitekturer

Da all generering er regel- og malbasert kan kode for mange ulike arkitekturer genereres av en slik tilleggsmodul. Antall mulige arkitekturer er delvis begrenset av den eksisterende løsningen man velger å knytte tilleggsmodulen opp mot. I tillegg må brukeren for hver ny arkitektur selv lage regler og maler.

Poengsum: 3 av 5. Trekk for at brukeren selv må spesifiserer regler og maler for nye arkitekturer, samt at tilleggsmodulens fleksibilitet til en viss grad hemmes av den eksisterende løsningen den knyttes mot.

Ressurskrav

Selv om tilleggsmodulen er tenkt implementert i Java vil den være avhengig av den eksisterende løsningen. Dette medfører at ressurskravene i stor grad vil være avhengig av hvilken eksisterende løsning som velges. Man vil imidlertid kunne spare en del tid på selve overgangen fra modellering til kodegenerering siden de to delene blir så tett integrert.

Poengsum: 3 av 4 Trekk for usikkerheten

Helhetsinntrykk

Dersom regel- og malformatet til denne løsningen blir designet generelt nok, samt at en god eksisterende løsning velges vil denne løsningen bli god. Den tilbyr nesten like stor fleksibilitet som parseren, og man slipper å bruke XMI som mellomledd. I denne løsningen kan man modellere og generere kode i samme program.

Utviklingskostnadene blir lave, i og med at dette prosjektet er gratis, men videreutviklingsutgiftene kan bli forholdsvis store og tidkrevende. I tillegg kan det være vanskelig å holde tilleggsmodulen fungerende ved nye versjoner av den eksisterende løsningen. Å binde seg opp mot et enkelt produkt kan derfor være farlig. I ytterste konsekvens stopper oppdateringen av eksisterende løsning, og da kan tilleggsmodulen bli nesten verdiløs. Man vi i alle fall ikke kunne legge til støtte for nye arkitekturer som også krever endringer i modellen.

Poengsum: 7 av 10.

Samlet poengsum En tilleggsmodul til en eksisterende løsning: 31

9.4.3 Oppsummering

Resultatene av den grundige evalueringen ble:

En frittstående parser (konstruert)	34,0 poeng
Rational Rose 2001 (eksisterende)	33,0 poeng
Together Control Center (eksisterende)	32,5 poeng
En tilleggsmodul til en eksisterende løsning (konstruert)	31,0 poeng
StructureBuilder (eksisterende)	24,0 poeng

Den konstruerte løsningen med en frittstående parser kom best ut av den grundige evalueringen og oppnådde hele 34 av 40 mulige poeng. Tett bak følger to av de eksisterende løsningene, henholdsvis Rational Rose 2001 og Together Control Center.

10 Vårt valg

Dette kapittelet konkluderer med et endelig valg av løsning, og inneholder argumentasjonen frem til denne konklusjonen.

10.1 Resultatet av evalueringen

Resultatene fra evalueringen var at den konstruerte løsningen med en frittstående parser fikk mest poeng. Dette var et ventet resultat da en konstruert løsning vil inneholde alle de egenskaper som er ønsket for det problemet den er konstruert for. Det som derimot var uventet var at to eksisterende løsninger kom så nær den konstruerte løsningen. Differansen er faktisk så liten at det i praksis kan sees på som et dødt løp mellom de tre første løsningene.

Dette resultatet medfører at før prosjektet kan konkludere i noe valg må den fundamentale avgjørelsen om noe nytt skal konstrueres eller man skal gå for en eksisterende løsning taes. For å kunne ta denne avgjørelsen må flere aspekter av problemstillingen drøftes grundigere med utgangspunkt i valget mellom konstruert og eksisterende løsning.

10.2 Faktiske kostnader

Med faktiske kostnader menes den totale kostnaden som vil bli konsekvensen dersom en løsning taes i bruk. Her taes det utgangspunkt i at 10 IT konsulenter hos EDB ASA trenger systemet, at et årsverk regnes til 1 000 000 NOK, og en time arbeid er regnet til 500 NOK.

Under evalueringen ble det i stor grad kun tatt hensyn til lisens- og utviklingskostnadene når en løsning ble evaluert på pris. Disse kostnadene kan derimot gi et galt bilde av de faktiske kostnader som en løsning vil medføre. Kostnader relatert til opplæring, brukerstøtte, videreutvikling, oppgraderinger, installasjon, vedlikehold og drift vil også medvirke i de faktiske kostnader. En løsning av denne typen regnes med å ha en praktisk maksimal levetid på om lag 5 år, som da de totale kostnadene kan sprees over.

Felles for begge løsningstypene er installasjonsutgiftene. Disse vil bli forholdsvis små da ingen av løsningene som her er aktuelle ikke krever avanserte installasjoner på server eller lignende, men kun skal kjøres lokalt på en maskin.

En konstruert løsning i dette prosjektet vil ikke ha noen utviklingskostnader og ingen lisenskostnader. Det som derimot ikke kommer helt frem i evaluering er at videreutviklingskostnadene kan bli store for en slik løsning. Med en konstruert løsning må alle utvidelser, endringer, og feilrettinger gjøres av en ansatt hos kunden. Slike oppgaver kan ta mye tid, og spesielt i dette tilfelle hvor koden blir utviklet av eksterne

konsulenter. I dette tilfellet hvor en slik løsning må utvides for hver ny arkitektur eller målspråk er et fornuftig estimat at vedlikehold, oppdatering og feilretting av en konstruert løsning vil kreve om lag et halvt årsverk fra en av IT konsulentene i bedriften. I tillegg må all brukerstøtte også håndteres internt hos kunden, men da alle brukerne av systemet er IT-spesialister er det grunn til å tro at denne kostnaden ikke vil bli så stor og vil derfor inngå i det halve årsverket som allerede er estimert. Kostnaden på opplæring blir heller ikke så stor for en frittstående parser da funksjonaliteten den tilbyr er enkel og brukeren sine muligheter er få. Dette medfører at tiden benyttet til selvstudium for hver ansatt er ubetydelig.

Dette gir grunnlag for følgende regnestykke for de faktiske kostnadene med en konstruert løsning som den frittstående parseren over 5 år:

Lisenskostnader		**0,-**
Vedlikehold, brukerstøtte	*0,5 årsverk i fem år blir 2,5 årsverk*	**2 500 000,-**
Kursing		**0,-**
Oppdateringer		**0,-**
Faktisk kostnad		**2 500 000,-**

Alle eksisterende løsninger som her er aktuelle vil ha store lisensutgifter knytt til seg. Rational Rose for eksempel koster ca. 40.000 kroner per node låst lisens og da er et års support med på kjøpet [ROS5], mens prisen for Together Control Center er enda høyere. Det som ikke kommer frem i evalueringen er derimot at videreutviklingskostnadene da blir små. Nye versjoner av den eksisterende løsningen, feilrettinger og eventuelle tilleggsmoduler utvikles av leverandøren og kan installeres gratis eller mot en liten kompensasjon. Opplæring i bruk av de eksisterende løsningene er nødvendig i dette tilfelle, da omfanget av begge de eksisterende alternativene er så stort. Slike kurs er kostbare, men effektive, og et grunnleggende kurs i Rational Rose ligger på ca. 35.000 kroner for å få en instruktør til å komme til bedriften å holde det.[ROS6] Brukerstøtte i tillegg til det første året som følger med hver lisens regnes som unødvendig i og med at brukerne er godt kvalifiserte på IT systemer fra før. Resten av programmet læres ved hjelp av selvstudium, der vi rekner med at hver ansatt vil bruke en uke totalt med kurset på å sette seg inni programmet.

Dette gir grunnlag for følgende regnestykke for de faktiske kostnaden med å ta i bruk Rational Rose som en eksisterende løsning over 5 år:

Lisenskostnader	*10 lisenser a kroner 40 000,-*	**400 000,-**
Vedlikehold, brukerstøtte		**0,-**
Kursing		**35 000,-**
Oppdateringer		**200 000,-**
Selvstudium	*40 timer per lisens utgjør 400 timer*	**200 000,-**
Faktisk kostnad		**835 000,-**

På grunnlag av disse utregningene ser man at den i utgangspunktet billige konstruerte løsningen viser seg å være 4 ganger så dyr som den eksisterende løsningen.

10.3 Utvidbarhet

Med utvidbarhet menes det de muligheter en løsning innehar for legge til ny funksjonalitet eller utvide eksisterende.

En egenprodusert løsning vil medføre større utvidbarhet enn en eksisterende løsning. Ved behov for oppdateringer eller utvidelser vil det med en egenprodusert løsning kunne settes i gang arbeid med dette straks behovet melder seg.

Med en eksisterende løsning vil avhengigheten opp mot leverandøren bli veldig sterk. Behovet for oppdateringer eller utvidelser vil ikke kunne tilfredstilles før leverandøren gir ut oppdateringer eller nye versjoner. Dersom leverandøren skulle slutte å satse på denne løsningen eller bli nødt til å legge ned driften vil det medføre at nødvendige oppdateringer ikke lenger er tilgjengelig. Dette vil da medføre at den eksisterende løsningen vil gradvis bli utdatert og man må investere andre eksisterende løsninger for å få dekket sine nye behov.

10.4 Kundens ønsker

Til tross for at resultatene peker i retning av at man burde satse på et eksisterende system ønsker kunden av forskjellige grunner å få konstruert sitt eget.

For kundens del var et av de viktigste aspektene ved prosjektet å se hva man kan oppnå med et system som bygger på regler og maler. Å ha muligheten til å definere egne regler og maler medfører stor fleksibilitet sammenlignet med å måtte forholde seg til begrensningene i ferdigdefinerte templates som gjerne leveres med et verktøy.

Generering av Enterprise JavaBeans og databaseskjema er en oppgave som ligner på andre prosesser som kan automatiseres. Kunden ser derfor for seg at et slikt system etter hvert kan utvides til også å omfatte andre prosesser.

Det er ikke ønskelig for kunden å binde seg opp mot en enkelt leverandør. Kunden jobber ofte på mindre prosjekter for sine kunder der verktøyvalgene vil variere. Det kan være at bruk av store systemer ikke er ønskelig på disse prosjektene. Selv om kunden har lisenser på et system som Rational Rose eller Together Control Center så kan ikke disse brukes når de er ute på oppdrag for andre. Da kreves det at denne bedriften allerede har lisenser, eller at det kjøpes nye lisenser i forbindelse med prosjektet. Kostnadene for lisenser kan fort blir uforholdsmessig store dersom systemet skal brukes bare i ett enkelt prosjekt. Ved å ha et egenutviklet system vil dette være noe kunden kan ha med seg på alle prosjekter uten at det medfører ekstrautgifter. Ved at systemet baserer seg på inndata i form av XMI så vil det kunne brukes i kombinasjon med flere av de vanligste modelleringsverktøyene

som finnes, og man vil langt på vei være uavhengig av hvilke verktøyvalg som ellers blir gjort.

10.5 Konklusjon

På grunnlag av de evalueringer og drøftninger som har blitt gjennomført, har det kommet frem at den rimeligste og beste løsningen er å gå til innkjøp av Rational Rose 2001 eller Together Control Center. Innføringen av en av disse løsningene vil oppfylle alle krav og kriterier kunden stilte til en lavere kostnad enn en konstruert løsning. I tillegg blir tiden før løsningen kan taes i bruk betraktelig lavere ved innføring av en eksisterende løsning, enn ved konstruksjon av en ny.

Dersom dette forstudiet hadde tatt plass i det reelle næringslivet ville derfor konklusjonen vært å argumentere hos kunden for å ta i bruk en av de eksisterende løsningene. Kundens ønsker om å undersøke en malbasert teknologi kunne så oppfylles som et internt prosjekt der konsulenter uten oppdrag for øyeblikket kunne bidra. I den videre prosessen ville så de eksisterende løsningene bli undersøkt nærmere, før et valg mellom de to gjenstående ville blitt gjort. Dette valget ville så blitt etterfulgt av forhandlinger med leverandør om pris og betingelser. Omgivelsene hos kunden måtte så klargjøres for den valgte løsningen før installasjonen tok plass, og løsningen kunne settes i produksjon.

Siden dette forstudiet ikke hører hjemme i det reelle næringslivet, men er en del av et fag ved NTNU, vil derimot konklusjonen bli å anbefale den konstruerte løsningen. Dette kan grunngis med at kunden ikke legger vekt på kostnadsaspektet ved prosjektet og ønsker en undersøkelse av mulighetene for en generell malbasert kodegenerator. For kunden er det et stort poeng å få teste ut hvilke muligheter som ligger i en løsning basert på maler, og at de gjerne ønsker å være litt i forkant av de store verktøyene når det kommer nye teknologier.

Konklusjonen til forstudiet blir dermed at den konstruert løsningen med en frittstående parser, som er beskrevet i kapittel 8.3, velges som grunnlag for den videre løsningen i prosjektet.

11 Litteraturliste

[COJ1] Cool:Joe Managing eBusiness Information : Computer Assosiates (2001)

[COM1] Undervisningsmateriell fra fag EECE 812 ved
Electrical Engineering - University of South Carolina
http://www.ece.sc.edu/classes/fall99/eece812/Docs/IntroToCom.htm
http://www.ece.sc.edu/classes/fall99/eece812/Docs/ComDetailsPart1.htm
http://www.ece.sc.edu/classes/fall99/eece812/Docs/ComDetailsPart2.htm

[COM2] Component Object Model (COM), DCOM and Related Capabilities
Software Engineering Institute (SEI) - Carnegie Mellon University
http://www.sei.cmu.edu/str/descriptions/com.html

[COM3] The Component Object Model: A technical Overview
Sara Williams and Charlie Kindel, Developer Relations Group
Microsoft Corporation
http://msdn.microsoft.com/library/default.asp?URL=/library/techart/msdn_comppr.htm

[GEN1] "Nordmenn i verdensteten", Yngve Vogt, Computerworld, 16. januar 2001

[GEV1] Code Conventions for the Java™ Programming Language
Revised April 20, 1999
Sun Microsystems, Inc
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

[OIF1] ObjectiF The tool for developing object-oriented and component-based software systems : Microtool (2000)

[PRO1] http://www.prolifics.de/downloads/panther-websphere.pdf

[ROS1] Analysis, Modeling, and Design Tools Market Forecast and Analysis, 2001-2005 : Rikki Kirzner (2001)

[ROS2] Enterprise Java and Rational Rose, Khawar Ahmed, Loïc Julien (2001)
http://www.rational.com/products/whitepapers/java_whitepaper.jsp

[ROS3] Model Driven Development with UML : Rational (2001)

[ROS4] Developing J2EE Applications with the UML and Rational Rose,
Khawar Ahmed, Rational (2001)
http://www.rational.com/products/whitepapers/tp197.jsp

[ROS5] Rationale Software Shop – Rational Rose 2100 Enterprise Edition
http://www.rational.com/shop/catalog/rose.jsp?ecomm_prod_no=13

[ROS6] Rationale Education – Course Catalog – Fundamentals of Rational Rose
http://www.rational.com/university/description/40.jsp

[SEL1] "XMI Opens Application Interchange", S.Brodsky, IBM
http://www-4.ibm.com/software/ad/standards/xmiwhite0399.pdf,
30.03.1999

[TCC1] Veivisere til oppstartshjelp med Together ControlCenter
http://www.togethersoft.com/products/controlcenter/features.jsp

[UML1] OMG Unified Modelling Language Spesification v.1.3
ftp://ftp.omg.org/pub/docs/ad/01-02-13.pdf

12 Indekser

12.1 Figurliste

12.2 Tabelliste

13 Vedlegg

13.1 Vedlegg 1 – XML og DTD

Den følgende intruduksjonen til XML er på engelsk, og innholdet er hentet fra Sun sine javasider, [XML1] http://java.sun.com/xml/jaxp-1.1/docs/tutorial/overview/1_xml.html og [XML2] http://java.sun.com/xml/jaxp-1.1/docs/tutorial/sax/5a_dtd.html.

Først kommer en introduksjon til XML, og så en del om å lage DTD.

What Is XML?

XML is a text-based markup language that is fast becoming the standard for data interchange on the Web. As with HTML, you identify data using tags (identifiers enclosed in angle brackets, like this: <...>). Collectively, the tags are known as "markup". But unlike HTML, XML tags *identify* the data, rather than specifying how to display it.

Where an HTML tag says something like "display this data in bold font" (`<b>...</b>`), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example: `<message>...</message>`).

Note:
Since identifying the data gives you some sense of what *means* (how to interpret it, what you should do with it), XML is sometimes described as a mechanism for specifying the *semantics* (meaning) of the data.

Here is an example of some XML data you might use for a messaging application:

```
<message>
    <to>you@yourAddress.com</to>
    <from>me@myAddress.com</from>
    <subject>XML Is Really Cool</subject>
    <text>
       How many ways is XML cool? Let me count the ways...
    </text>
</message>
```

The tags in this example identify the message as a whole, the destination and sender addresses, the subject, and the text of the message. As in HTML, the `<to>` tag has a matching end tag: `</to>`. The data between the tag and and its matching end tag defines an element of the XML data. Note, too, that the content of the `<to>` tag is entirely contained within the scope of the `<message>..</message>` tag. It is this ability for one tag to contain others that gives XML its ability to represent hierarchical data structures Once again, as with HTML, whitespace is essentially irrelevant, so you can format the data for readability and yet still process it easily with a program. Unlike HTML, however, in XML you could easily search a data set for messages containing "cool" in the subject,

because the XML tags identify the content of the data, rather than specifying its representation.

Tags and Attributes

Tags can also contain attributes -- additional information included as part of the tag itself, within the tag's angle brackets. The following example shows an email message structure that uses attributes for the "to", "from", and "subject" fields:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Cool">
    <text>
       How many ways is XML cool? Let me count the ways...
    </text>
</message>
```

Empty Tags

One really big difference between XML and HTML is that an XML document is always constrained to be well formed. There are several rules that determine when a document is well-formed, but one of the most important is that every tag has a closing tag. So, in XML, the `</to>` tag is not optional. The `<to>` element is never terminated by any tag other than `</to>`.

Sometimes, though, it makes sense to have a tag that stands by itself. For example, you might want to add a "flag" tag that marks message as important. A tag like that doesn't enclose any content, so it's known as an "empty tag". You can create an empty tag by ending it with `/>` instead of `>`. For example, the following message contains such a tag:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
         subject="XML Is Really Cool">
    <flag/>
    <text>
       How many ways is XML cool? Let me count the ways...
    </text>
</message>
```

You can control which tags are allowed to be empty by creating a Document Type Definition, or DTD.

Comments in XML Files

XML comments look just like HTML comments:

```
<!-- This is a comment -->
```

The XML Prolog

To complete this journeyman's introduction to XML, note that an XML file always starts with a prolog. The minimal prolog contains a declaration that identifies the document as an XML document, like this:

```
<?xml version="1.0"?>
```

The declaration may also contain additional information, like this:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

version
: Identifies the version of the XML markup language used in the data. This attribute is not optional.

encoding
: Identifies the character set used to encode the data. "ISO-8859-1" is "Latin-1" the Western European and English language character set. (The default is compressed Unicode: UTF-8.)

standalone
: Tells whether or not this document references an external entity or an external data type specification (see below). If there are no external references, then "yes" is appropriate

XML glossary

A glossary that contains words used in XML could be found at: [XML3]
http://java.sun.com/xml/jaxp-1.1/docs/tutorial/attribute

Creating a Document Type Definition (DTD)

After the XML declaration, the document prolog can include a DTD, which lets you specify the kinds of tags that can be included in your XML document. In addition to telling a validating parser which tags are valid, and in what arrangements, a DTD tells both validating and nonvalidating parsers where text is expected, which lets the parser determine whether the whitespace it sees is significant or ignorable.

Basic DTD Definitions

To begin learning about DTD definitions, let's for example start by telling the parser where whitespace is ignorable.

Start by creating a file named for example `slideshow.dtd`**.** Enter an XML declaration and a comment to identify the file, as shown below:

```
<?xml version='1.0' encoding='utf-8'?>

<!-- DTD for a simple "slide show". -->
```

Next, add the text highlight below to specify that a `slideshow` element contains `slide` elements and nothing else:

```
<!-- DTD for a simple "slide show". -->

<!ELEMENT slideshow (slide+)>
```

As you can see, the DTD tag starts with `<!` followed by the tag name (`ELEMENT`). After the tag name comes the name of the element that is being defined (`slideshow`) and, in parentheses, one or more items that indicate the valid contents for that element. In this case, the notation says that a `slideshow` consists of one or more `slide` elements.

Without the plus sign, the definition would be saying that a `slideshow` consists of a single `slide` element. Here are the qualifiers you can add to an element definition:

Qualifier	*Name*	*Meaning*
?	Question Mark	Optional (zero or one)
*****	Asterisk	Zero or more
+	Plus Sign	One or more

You can include multiple elements inside the parentheses in a comma separated list, and use a qualifier on each element to indicate how many instances of that element may occur. The comma-separated list tells which elements are valid and the order they can occur in.

You can also nest parentheses to group multiple items. For an example, after defining an `image` element (coming up shortly), you could declare that every `image` element must be paired with a `title` element in a slide by specifying `((image, title)+)`. Here, the plus sign applies to the `image/title` pair to indicate that one or more pairs of the specified items can occur.

Defining Text and Nested Elements

Now that you have told the parser something about where *not* to expect text, let's see how to tell it where text *can* occur. Add the text highlighted below to define the `slide`, `title`, `item`, and `list` elements:

```
<!ELEMENT slideshow (slide+)>
<!ELEMENT slide (title, item*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
```

The first line you added says that a slide consists of a `title` followed by zero or more `item` elements. Nothing new there. The next line says that a title consists entirely of *parsed character data* (`PCDATA`). That's known as "text" in most parts of the country, but in XML-speak it's called "parsed character data". (That distinguishes it from `CDATA` sections, which contain character data that is not parsed.) The "#" that precedes `PCDATA` indicates that what follows is a special word, rather than an element name.

The last line introduces the vertical bar (`|`), which indicates an *or* condition. In this case, either `PCDATA` or an `item` can occur. The asterisk at the end says that either one can occur zero or more times in succession. The result of this specification is known as a mixed-content model, because any number of `item` elements can be interspersed with the text. Such models must always be defined with `#PCDATA` specified first, some number of alternate items divided by vertical bars (`|`), and an asterisk (`*`) at the end.

Limitations of DTDs

It would be nice if we could specify that an `item` contains either text, or text followed by one or more list items. But that kind of specification turns out to be hard to achieve in a DTD. For example, you might be tempted to define an `item` like this:

```
<!ELEMENT item (#PCDATA | (#PCDATA, item+)) >
```

That would certainly be accurate, but as soon as the parser sees `#PCDATA` and the vertical bar, it requires the remaining definition to conform to the mixed-content model. This specification doesn't, so you get can error that says: `Illegal mixed content model for 'item'. Found &#x28; ...`, where the hex character 28 is the angle bracket the ends the definition.

Trying to double-define the item element doesn't work, either. A specification like this:

```
<!ELEMENT item (#PCDATA) >
<!ELEMENT item (#PCDATA, item+) >
```

produces a "duplicate definition" warning when the validating parser runs. The second definition is, in fact, ignored. So it seems that defining a mixed content model (which allows `item` elements to be interspersed in text) is about as good as we can do.

In addition to the limitations of the mixed content model mentioned above, there is no way to further qualify the kind of text that can occur where `PCDATA` has been specified. Should it contain only numbers? Should be in a date format, or possibly a monetary format? There is no way to say in the context of a DTD.

Finally, note that the DTD offers no sense of hierarchy. The definition for the `title` element applies equally to a `slide` title and to an `item` title. When we expand the DTD to allow HTML-style markup in addition to plain text, it would make sense to restrict the size of an `item` title compared to a `slide` title, for example. But the only way to do that would be to give one of them a different name, such as "`item-title`". The bottom line is that the lack of hierarchy in the DTD forces you to introduce a "hyphenation hierarchy" (or its equivalent) in your namespace. All of these limitations are fundamental motivations behind the development of schema-specification standards.

Special Element Values in the DTD

Rather than specifying a parenthesized list of elements, the element definition could use one of two special values: `ANY` or `EMPTY`. The `ANY` specification says that the element may contain any other defined element, or `PCDATA`. Such a specification is usually used for the root element of a general-purpose XML document such as you might create with a word processor. Textual elements could occur in any order in such a document, so specifying `ANY` makes sense.

The `EMPTY` specification says that the element contains no contents. So the DTD for email messages that let you "flag" the message with `<flag/>` might have a line like this in the DTD:

```
<!ELEMENT flag EMPTY>
```

Referencing the DTD

In this case, the DTD definition is in a separate file from the XML document. That means you have to reference it from the XML document, which makes the DTD file part of the external subset of the full Document Type Definition (DTD) for the XML file. As you'll see later on, you can also include parts of the DTD within the document. Such definitions constitute the local subset of the DTD.

To reference the DTD file you just created, add the line highlighted below to your `xml` file:

```
<!--  A SAMPLE set of slides  -->
<!DOCTYPE slideshow SYSTEM "slideshow.dtd">
```

Again, the DTD tag starts with "`<!`". In this case, the tag name, `DOCTYPE`, says that the document is a `slideshow`, which means that the document consists of the `slideshow` element and everything within it.

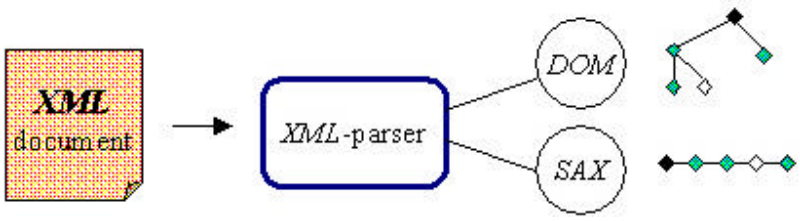
The `DOCTYPE` tag occurs after the XML declaration and before the root element. The `SYSTEM` identifier specifies the location of the DTD file. Since it does not start with a prefix like `http:/` or `file:/`, the path is relative to the location of the XML document. Remember the `setDocumentLocator` method? The parser is using that information to find the DTD file, just as your application would to find a file relative to the XML document. A `PUBLIC` identifier could also be used to specify the DTD file using a unique name -- but the parser would have to be able to resolve it

The `DOCTYPE` specification could also contain DTD definitions within the XML document, rather than referring to an external DTD file. Such definitions would be contained in square brackets, like this:.

```
<!DOCTYPE slideshow SYSTEM "slideshow1.dtd" [
  ...local subset definitions here...
]>
```

13.2 Vedlegg 2 – XML-parsing

Figuren under viser parsingen av et XML-dokument. Det er to forskjellige API'er som er definert: *DOM* (Document Object Model) som genererer et hierarkisk parsetre og *SAX* (Simple API for XML) som prosesserer et dokument hendelsesbasert uten å generere et tre. DOM er veldig nyttig for å navigere i et dokument, mens SAX kan prosessere veldig store dokumenter uten å ta opp mye minne.



Figur 1: parsing av *XML*

Document Object Model (DOM)

DOM er et platform- og språk-nøytral interfacfe som vil tillate programmer og script å dynamisk aksesserer og oppdatere innholdet, strukturen og stilen på dokumenter. Dokumentet kan bli prosessert videre og resultatene kan bli innlemmet i den prestenterte siden.

DOM bygger opp et tre av et dokument. Dette treet representerer så alle elementene(objektene) i dokumentet som noder eller blad. Alle objektene kan da bli aksessert direkte, uten å måtte søke gjennom dokumentet sekvensielt. En ulempe med dette er at hele treet lagres i minnet, og derfor er størrelsen på minnet en begrensning for hvor stort dokument som kan prosesseres.

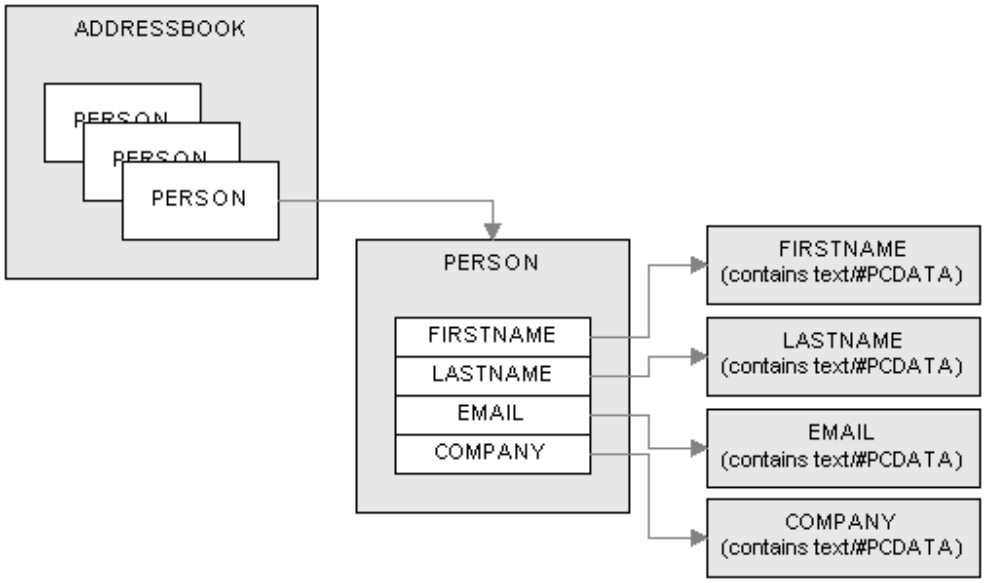
Et eksempel på hvordan man bruker en slik parser kan du se i eksempelet under. Eksempelet viser bruk av sun sin XML-parser

```
import com.sun.xml.tree.*;
import org.w3c.dom.*;

try {
    String url =
        "http://beanfactory.com/xml/AddressBook.xml";
        //URL henviser til filen som skal parses
    Document doc =
        XmlDocumentBuilder.createXmlDocument(url);
}
catch(Exception e){}
```

Her er en figur som viser hvordan et dokument blir ”delt opp” gjennom DOM-parsing.

DTD-delen av XML-dokumentet:

```
<?xml version="1.0"?>
<!DOCTYPE ADDRESSBOOK [
<!ELEMENT ADDRESSBOOK (PERSON)*>
<!ELEMENT PERSON (LASTNAME, FIRSTNAME, COMPANY, EMAIL)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT COMPANY (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
]>
```



Figur 2: Illustrasjon av DTDen

Man kan så kjøre diverse metoder for å manipulere dataene på en enkel måte.

The Simple API for XML (SAX)

SAX er et standart inrerface for hendelses-basert XML parsing. SAX tillater, slik som DOM, å aksessere XML-dokumenter, men trenger ikke å bygge opp et helt tre av dokumentet fordi det er hendelsesbasert. Dokumentet blir prosessert sekvensielt og treff på søkte mønster produserer en hendelse som han bli brukt videre. Dette betyr at SAX kan benyttes til svært store dokumenter, uten å ta opp mye resurser. Det er spesielt nå man ikke trenger å parse hele XML-filen at denne metoden bli svært effektiv.

Her følger et ekempel på en java SAX-parser. Denne er fra Sun.

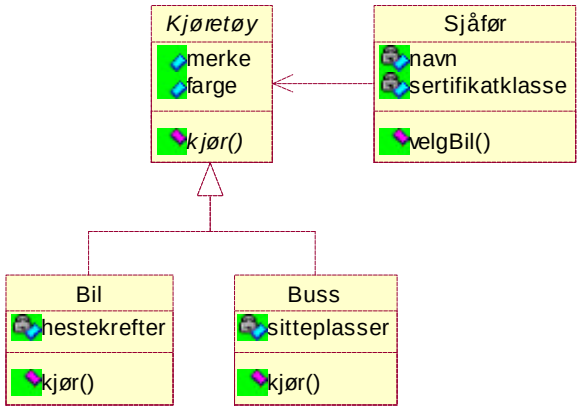
```
public static void main(String argv[])

    {
        if (argv.length != 1) {
            System.err.println("Usage: cmd filename");
            System.exit(1);
        }

        // Use an instance of ourselves as the SAX event handler
        DefaultHandler handler = new Echo();

        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse( new File(argv[0]), handler );

        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }
```

13.3 Vedlegg 3 – UML-diagrammer

Her er en kort gjennomgang av de ulike diagramtypene i UML med et lite eksempel til hver type.

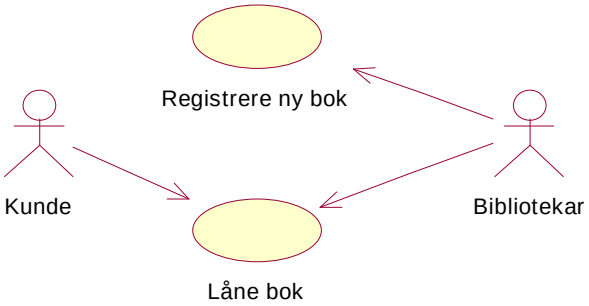
Klassediagram

Et klassediagram viser klassene og grensesnittene et system består av, samt relasjonene og samspillet mellom dem.



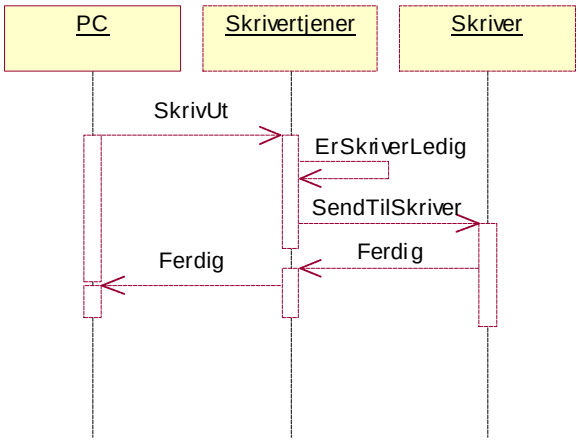
Brukstilfellediagram

Et brukstilfellediagram er en dynamisk beskrivelse av hva et system skal gjøre, men ikke hvordan. Det blir ofte brukt for å modellere kravene til et system.



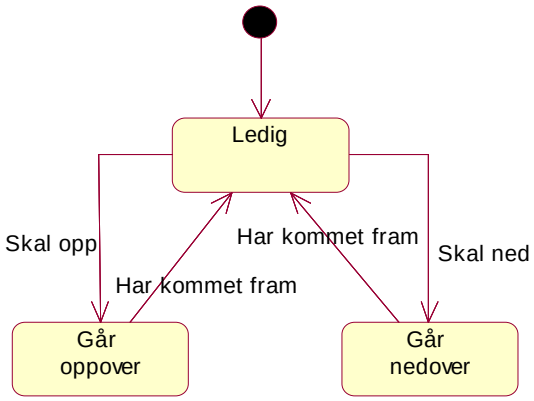
Sekvensdiagram

Et sekvensdiagram viser tidsrekkefølgen for meldinger mellom objekter, og dermed hvordan de samarbeider.



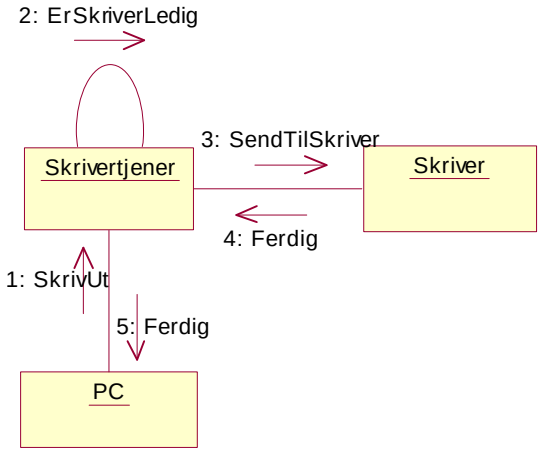
Tilstandsdiagram

Et tilstandsdiagram viser en tilstandsmaskin og legger vekt på hva som skal til for å gå fra en tilstand til en annen.



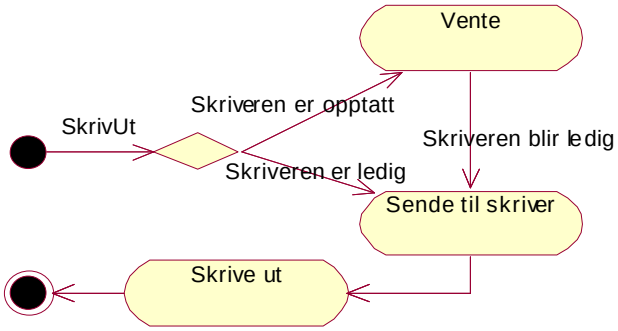
Samarbeidsdiagram

Et samarbeidsdiagram viser organiseringen av objekter og flyten av meldinger i en interaksjon.



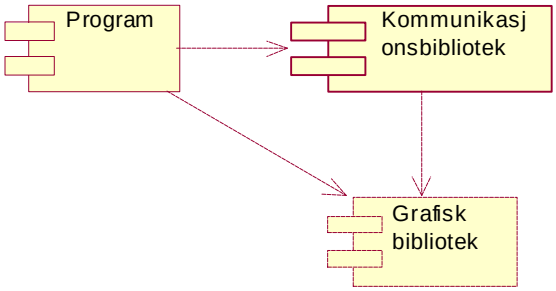
Aktivitetsdiagram

Et aktivitetsdiagram viser flyten fra aktivitet til aktivitet. En aktivitet er en abstrahering på høyt nivå, og vil gjerne inneholde flere enkelthandlinger.



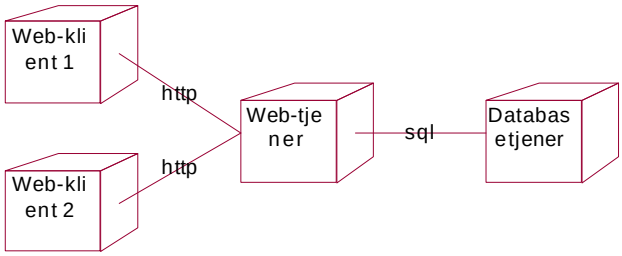
Komponentdiagram

Et komponentdiagram viser et relasjoner mellom programvarekomponenter. Komponenter i UML kan for eksempel være filer, grensesnitt, klasser og lignende som samles i pakker. Pakker kan også nøstes innenfor andre pakker.



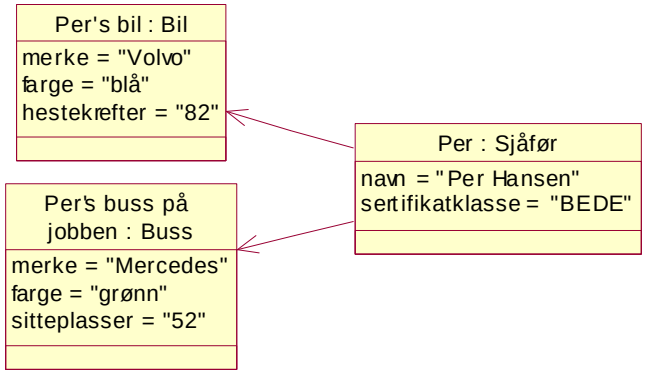
Utplasseringsdiagram

Et utplasseringsdiagram viser konfigurasjonen av eksekverbare prosesseringsnoder og komponentene som inngår.



Objektdiagram

Et objektdiagram er en statisk diagramtype som viser objekter og deres relasjoner på et gitt tidspunkt. Objekter vil gjerne være instanser av klasser.



13.4 Vedlegg 4 – Rational Unified Process (RUP)

RUP definerer følgende fire prosjektfaser:

1. **Inception** (forberedelsesfasen, forprosjekt) utgjør normalt om lag 10% av tiden det tar å gjennomføre hele prosjektet, men bare 5% av arbeidsinnsatsen. Fasen leder frem til en milepæl, "Life-Cycle Objective", der man har definert prosjektets omfang og mål, utviklet en forretningsmodell og formulert en generell kravspesifikasjon.

2. **Elaboration** (bearbeidelsesfasen) krever om lag 30% av tiden og 20% av arbeidsinnsatsen. Fasen kan deles inn i flere iterasjoner, der hver avsluttes med en mindre milepæl med detaljerte kravspesifikasjoner, analyse og design for en avgrenset del av systemet. De mest kritiske deler av systemet bearbeides først. Tidlige alfaversjoner produseres. Fasen leder frem til en milepæl, "Life-Cycle Architecture", der systemets tekniske arkitektur er definert og utprøvd.

3. **Construction** (konstruksjonsfasen) krever om lag 50% av tiden og 65% av arbeidsinnsatsen. Fasen kan deles inn i flere iterasjoner, som først og fremst omfatter programmering og hver avsluttes med en ny alfaversjon av systemet. Fasen leder frem til en milepæl, "Initial Operational Capability", der all vesentlig funksjonalitet er implementert og første betaversjon levert.

4. **Transition** (avslutningsfasen) krever om lag 10% av tiden og 10% arbeidsinnsatsen. Fasen kan deles inn i flere iterasjoner, som hver avsluttes med en ny betaversjon av systemet. Fasen leder frem til en milepæl, "Product Release", der systemet er levert og satt i drift.
Hver iterasjon omfatter seks arbeidsprosesser ("workflows") i større eller mindre omfang. Den klassiske fossefallmodellen foreskriver at man først må utarbeide kravspesifikasjon for hele systemet under ett, deretter gjøre analyse og design, og deretter programmere og teste. Man kan si at en iterativ prosess benytter vannfallsmodellen suksessivt på mindre delsystemer istedenfor å ta hele systemet under ett. Fordelen er at man tidligere blir klar over kritiske elementer, og dessuten kan lære og dermed forbedre sine arbeidsmetoder underveis.

De arbeidsprosessene som inngår i selve utviklingsarbeidet er:

A. **Forretningsmodellering** ("Business Modeling"). I samarbeid med kunden modelleres den organisasjon og den forretningsdrift som systemet skal operere innenfor.

B. **Kravspesifikasjon** ("Requirements"). I samarbeid med kunden beskrives funksjonelle krav ved hjelp av brukstilfeller ("Use-Cases"), scenarier og storyboards. Tekniske og andre tilleggskrav formuleres. Beskrivelse av testscenarier.

C. **Analyse og design**. Vi modellerer funksjonalitet og beskriver en teknisk løsning for systemet ved hjelp av klasse- og sekvensdiagrammer i Rational Rose.

D. **Implementasjon**. Programmering i JBuilder, Visual Studio eller andre utviklingsverktøy. Vi søker i størst mulig grad å gjenbruke tilgjengelige komponenter fra tidligere prosjekter og fra solide leverandører, fremfor å oppfinne hjulet på nytt hver gang. Systemet vil utvikles mot det aktuelle driftsmiljø - i forhold til operativsystemer (Solaris, MVS, Linux, Windows), databaser (Oracle, Sybase, SQL Server, MySQL), applikasjonsservere (BEA WebLogic, IBM WebSphere, Borland Application Server, JBoss) og andre systemressurser.

E. **Test**. Utprøving av alfa- og betaversjoner i et testmiljø.

F. **Installasjon** (”Deployment”). Installasjon og konfigurasjon av pilotversjoner og ferdig versjon ute hos kunde.

13.5 Vedlegg 5 – Kodeeksempler

Her følger koden som ble generert i den grundige testen av de eksisterende løsningene i kapittel 9.3. De kommer i samme rekkefølge som produktene ble testet, dvs først Ratinal Rose, så Together Control Center og til slutt Structure Builder. Innen hvert av produktene er rekkefølgen på den genererte koden implementasjons klassen, remote interfacet, home interfacet, deploymet descriptor og tilslutt eventuelt andre genererte koder.

RATIONAL ROSE

Implementasjonen av Person bønnen

PersonEJB.java:

```
//
// -- Java Code Generation Process --

// Import Statements
import java.rmi.RemoteException;
import javax.ejb.*;

public abstract class PersonEJB implements javax.ejb.EntityBean
{
    /*
       Attributes declaration
    */
    private int navn;
    private int personID;
    public javax.ejb.EntityContext EJB_Context;
    public EksamensresultatEJB theEksamensresultatEJB;
    public Postkontor thePostkontor;
    public Fag theFag;

    /**
     * @roseuid 3BB9BCBA01B7
     * @J2EE_METHOD  --  PersonEJB
     */
    public PersonEJB    ()
    {

    }

    /**
     * @roseuid 3BB9BCC900EF
     * @J2EE_METHOD  --  ejbActivate
     * A container invokes this method when the instance is taken out of the pool of
available
     * instances to become associated with a specific EJB object. This method transitions
     * the instance to the ready state. This method executes in an unspecified transaction
     * context.
     */
    public void ejbActivate    ()
    {

    }




/**
```

```
     * @roseuid 3BB9BCC90103
     * @J2EE_METHOD  --  ejbPassivate
     * A container invokes this method on an instance before the instance becomes
disassociated
     * with a specific EJB object. After this method completes, the container will place
     * the instance into the pool of available instances. This method executes in an
unspecified
     * transaction context.
     */
    public void ejbPassivate    ()
    {

    }

    /**
     * @roseuid 3BB9BCC90117
     * @J2EE_METHOD  --  ejbLoad
     * A container invokes this method to instruct the instance to synchronize its state
     * by loading it from the underlying database. This method always executes in the
transaction
     * context determined by the value of the transaction attribute in the deployment
descriptor.
     */
    public void ejbLoad    ()
    {

    }

    /**
     * @roseuid 3BB9BCC9012B
     * @J2EE_METHOD  --  ejbStore
     * A container invokes this method to instruct the instance to synchronize its state
     * by storing it to the underlying database. This method always executes in the
transaction
     * context determined by the value of the transaction attribute in the deployment
descriptor.
     */
    public void ejbStore    ()
    {

    }

    /**
     * @roseuid 3BB9BCC90149
     * @J2EE_METHOD  --  ejbRemove
     * A container invokes this method before it removes the EJB object that is currently
     * associated with the instance. It is invoked when a client invokes a remove
operation
     * on the enterprise Bean's home or remote interface. It transitions the instance from
     * the ready state to the pool of available instances. It is called in the transaction
     * context of the remove operation.
     */
    public void ejbRemove    () throws javax.ejb.RemoveException
    {

    }

    /**
     * @roseuid 3BB9BCC9015D
     * @J2EE_METHOD  --  setEntityContext
     * Set the associated entity context. The container invokes this method on an instance
     * after the instance has been created. This method is called in an unspecified
transaction
     * context.
     */
    public void setEntityContext    (javax.ejb.EntityContext ctx)
    {

    }

    /**
```

```
     * @roseuid 3BB9BCC90171
     * @J2EE_METHOD  --  unsetEntityContext
     * Unset the associated entity context. The container calls this method before
removing
     * the instance. This is the last method that the container invokes on the instance.
     * The Java garbage collector will  invoke the finalize() method on the instance. It
     * is called in an unspecified transaction context.
     */
    public void unsetEntityContext    ()
    {

    }

    /**
     * @roseuid 3BB9BD7D02C6
     * @J2EE_METHOD  --  ejbCreate
     * Matching method of the create(...) method of the bean's home interface. The
container
     * invokes an ejbCreate method to create an entity object. It executes in the
transaction
     * context determined by the transaction attribute of the matching create(...) method.
     */

    /**
     * @since 02.10.2001
     */
    public PersonPK ejbCreate    () throws javax.ejb.CreateException
    {
    return null;
    }

    /**
     * @roseuid 3BB9BD7D02E4
     * @J2EE_METHOD  --  ejbPostCreate
     * Matching method of ejbCreate. The container invokes the matching ejbPostCreate
method
     * on an instance after it invokes the ejbCreate method with the same arguments. It
     * executes in the same transaction context as that of the matching ejbCreate method.
     */

    /**
     * @since 02.10.2001
     */
    public void ejbPostCreate    () throws javax.ejb.CreateException
    {

    }
}
```

Remote Interfacet

Person.java

```
//
// -- Java Code Generation Process --

// Import Statements
import java.rmi.RemoteException;
import javax.ejb.*;

public interface Person extends javax.ejb.EJBObject
{
}
```

Home Interfacet

PersonHome.java:

```
//
// -- Java Code Generation Process --

// Import Statements
import java.rmi.RemoteException;
import javax.ejb.*;

public interface PersonHome extends javax.ejb.EJBHome
{

    /**
     * @roseuid 3BB9BCC901C2
     * @J2EE_METHOD  --  findByPrimaryKey
     * Called by the client to find an EJB bean instance, usually find by primary key.
     */
    public Person findByPrimaryKey    (PersonPK primaryKey)
                throws java.rmi.RemoteException, javax.ejb.FinderException;

    /**
     * @roseuid 3BB9BCC90398
     * @J2EE_METHOD  --  create
     * Called by the client to create an EJB bean instance. It requires a matching pair in
     * the bean class, i.e. ejbCreate(...).
     */
    public Person create    ()
                throws java.rmi.RemoteException, javax.ejb.CreateException;
}
```

Primary Key class

PersonPK.java:

```
//
// -- Java Code Generation Process --

// Import Statements
import java.io.Serializable;

public class PersonPK implements java.io.Serializable
{

    /**
     * @roseuid 3BB9BCC90081
     * @J2EE_METHOD  --  hashCode
     * Referencing to object that represetns the entity object.
     */
    public int hashCode    ()
    {
return 0;
    }

    /**
     * @roseuid 3BB9BCC9008B
     * @J2EE_METHOD  --  equals
     * Method that compares two entity object references -- since the Java
Object.equals(Object
     * obj) method is unspecified.
     */
    public boolean equals    ()
    {
return true;
    }

    /**
     * @roseuid 3BB9BCC9008C
     * @J2EE_METHOD  --  toString
     * Own toString() method of a bean's PK class.
```

```
     */
    public java.lang.String toString    ()
    {
return null;
    }

    /**
     * @roseuid 3BBAD5B80184
     * @J2EE_METHOD  --  PersonPK
     */
    public PersonPK    ()
    {

    }
}
```

Deployment Descriptor

PersonEJB.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">

<ejb-jar>
 <enterprise-beans>
  <entity>
   <ejb-name>PersonEJB</ejb-name>
   <home>PersonHome</home>
   <remote>Person</remote>
   <ejb-class>PersonEJB</ejb-class>
   <persistence-type>Container</persistence-type>
   <prim-key-class>PersonPK</prim-key-class>
   <reentrant>False</reentrant>
  </entity>
 </enterprise-beans>
</ejb-jar>
```

Together ControlCenter

Implementasjonen av Person bønnen.

PersonBean.java

```
/* Generated by Together */
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.CreateException;
import java.sql.SQLException;
import javax.ejb.FinderException;

/**
 * @ejbHome <{personHome}>
 * @ejbRemote <{person}>
 * @ejbPrimaryKey <{personPK}>*/
public class personBean implements EntityBean {
    private EntityContext ctx;
    public int personID;
    public String navn;

    public void setEntityContext(EntityContext context) throws RemoteException,
EJBException {
        ctx = context;
    }

    public void unsetEntityContext() throws RemoteException, EJBException {
        ctx = null;
    }

    public void ejbActivate() throws RemoteException, EJBException {
    }

    public void ejbPassivate() throws RemoteException, EJBException {
    }

    public void ejbRemove() throws RemoteException, EJBException {
    }

    public void ejbStore() throws RemoteException, EJBException {
    }

    public void ejbLoad() throws RemoteException, EJBException {
    }

    public personPK ejbCreate() throws CreateException, EJBException, RemoteException,
SQLException {
        // Write your code here
        return null;
    }

    public void ejbPostCreate() throws CreateException, EJBException, RemoteException,
SQLException {
            // Write your code here
        }

    public personPK ejbFindByPrimaryKey(personPK pk) throws FinderException, EJBException
{
            // Write your code here
            return null;
        }

    public int getPersonID(){ return personID; }

    public void setPersonID(int param){ this.personID = param; }
}
```

Remote interface

Person.java:

```
/* Generated by Together */
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import javax.ejb.EJBException;

public interface person extends EJBObject {
    int getPersonID() throws RemoteException, EJBException;

    void setPersonID(int param) throws RemoteException, EJBException;

    /** @link dependency */
    /*#personBean lnkEntity1Bean;*/
}
```

Home interface

PersonHome.java:

```
/* Generated by Together */
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.EJBException;
import java.rmi.RemoteException;
import java.sql.SQLException;
import javax.ejb.FinderException;

public interface personHome extends EJBHome {
person create() throws CreateException, EJBException, RemoteException, SQLException;

    person findByPrimaryKey(personPK pk) throws FinderException, RemoteException,
EJBException;

    /** @link dependency */
    /*#personBean lnkEntity1Bean;*/
}
```

Primary Key class

PersonPK.java:

```
/* Generated by Together */
import java.io.Serializable;

public class personPK implements Serializable {
    public personPK() {
    }

    public boolean equals(Object o){
            if (o instanceof personPK) {
                personPK otherKey = (personPK)o;
                return (personID == otherKey.personID);
            } else
                return false;
        }

    public int hashCode(){
            return String.valueOf(personID).hashCode();
        }

    /** @link dependency */
    /*#personBean lnkEntity1Bean;*/
    public int personID;
}
```

Annen kode

SQL-kode for opprettelse av tabell i database(MS SQL):

```
CREATE TABLE %SCHEMA_NAME%."person"
(
"personID" int identity,
"navn" varchar(128),
  CONSTRAINT "PK_personBean" PRIMARY KEY ("personID")
)
```

StructureBuilder

Implementasjonen av Person bønnen.

Person.java

```
import javax.ejb.EntityContext;
import javax.ejb.EntityBean;

/**
 * @stereotype EntityBean
 * @homeInterface PersonHome
 * @remoteInterface PersonRemote
 * @primaryKey PersonPK
 * @persistence Container
 */
public class Person implements EntityBean
{
        private String navn;
        private int personID;
        /** @SBGen Variable (,postadresse,,0)   */
        private Postkontor postadresse;
        /**
         * The container assigned reference to the entity
         */
        private EntityContext context;
        /** @SBGen Variable (,fag,,24)   */
        private Fag[] fag = new Fag[10];

        public Person()
        {
        }

        public void ejbActivate()
        {
        }

        public void ejbLoad()
        {
        }

        public void ejbPassivate()
        {
        }

        /**
         * @exception RemoveException
         */
        public void ejbRemove()
                throws RemoveException
        {
        }

        public void ejbStore()
        {
        }

        /**
         * @param p0
         */
        public void setEntityContext(EntityContext p0)
        {
        }

        public void unsetEntityContext()
        {
        }

        /**
```

```
         * Sets the context of the bean
         * @param ec
         */
        public void setEntityContext(EntityContext ec)
        {
                context = ec;
                // to do: code goes here.
        }

        /**
         * This method is invoked when a client invokes the matching create()
         * on the home interface. Initialize the fields that will be used
         * used by the container to create a record in the database
         *
         * @return Returns the primary key of the new entity
         */
        public PersonPK ejbCreate()
        {
                return null;
        }

        /**
         * The container invokes this method after invoking the ejbCreate
         * method with the same arguments.
         */
        public void ejbPostCreate()
        {
        }
}
```

Remote interface

PersonRemote.java:

```
import javax.ejb.EJBObject;

public interface PersonRemote extends EJBObject
{
}
```

Home interface

PersonHome.java:

```
import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.rmi.RemoteException;

public interface PersonHome extends EJBHome
{
        /**
         * @return
         */
        public PersonRemote findByPrimaryKey(PersonPK key)
                throws FinderException, RemoteException;
}
```

Primary Key class

PersonPK.java:

```
import java.io.Serializable;

public class PersonPK implements Serializable
{
        public PersonPK()
        {
        }
```

```
	/**
	 * @return
	 * @SBGen Method hashcode
	 */
	public int hashCode()
	{
		// SBgen: Generate hashcode
		return 0;
		// SBgen: End hashcode
	}

	/**
	 * @param that
	 * @return
	 * @SBGen Method compare class using
	 */
	public boolean equals(Object that)
	{
		// SBgen: Compare members
		return(this == that);
		// SBgen: End compare
	}
}
```

13.6 Vedlegg 6 – XMI

Her er noen eksempler på hvordan XMI kan se ut.

XMI DTD som datamodell

XMI DTDen som genereres utifra en UML datamodell beskriver entitetene og deres attributter. Her tar vi utgangspunkt i at UML datamodellen ikke inneholder metoder.

Under er en XMI DTD og korresponderende UML:

```
<!ELEMENT Auto (Auto.Make, Auto.Model, Auto.Year,
                Auto.Color, Auto.Price,
                XMI.extension*)? >
<!ATTLIST Auto %XMI.element.att; %XMI.link.att;>
<!ELEMENT Auto.Make (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Model (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Year (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Color (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Price (#PCDATA | XMI.reference)* >
```

Et XMI dokument som bruker XMI DTD'en ovenfor beskriver entitets*instanser*.

Eksempel på XMI dokument baser på XMI DTD'en:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI SYSTEM "auto.dtd">
<XMI xmi.version="1.0" >
<XMI.header>
<XMI.documentation>
An example of an auto.
</XMI.documentation>
</XMI.header>
<XMI.content>
<Auto>
<Auto.Make>Ford</Auto.Make>
<Auto.Model>Mustang</Auto.Model>
<Auto.Year>99</Auto.Year>
<Auto.Color>blue</Auto.Color>
<Auto.Price>25000</Auto.Price>
</Auto>
</XMI.content>
</XMI>
```

[IBM XMI]

XMI dokument som datamodell

XMI DTD'en beskriver hvilke elementer og strukturer som finnes i modellen. Under er et eksempel på en DTD som definerer hvilke elementer som inngår i en datamodell:

```
<!ELEMENT RelationalDatabase (RelationalDatabase.name, XMI.extension*,
RelationalDatabase.tables*)? >
<!ELEMENT RelationalDatabase.name (#PCDATA | XMI.reference)* >
<!ELEMENT RelationalDatabase.tables (Table)* >
<!ELEMENT Table (Table.name, XMI.extension*, Table.columns*)? >
<!ELEMENT Table.name (#PCDATA | XMI.reference)* >
<!ELEMENT Table.columns (Column)* >
<!ELEMENT Column (Column.name, XMI.extension*)? >
<!ELEMENT Column.name (#PCDATA | XMI.reference)* >
```

En datamodell med én tabell vil da se slik ut i et XMI dokument:

```
<RelationalDatabase>
<RelationalDatabase.name>accountsDB</RelationalDatabase>
<RelationalDatabase.tables>
<Table>
<Table.name>customers</Table.name>
<Table.columns>
<Column><Column.name>customerId</Column.name></Column>
<Column><Column.name>customerName</Column.name></Column>
<Column><Column.name>balance</Column.name></Column>
</Table.columns>
</Table>
</RelationalDatabase.tables>
</RelationalDatabase>
```

Kundestyrt prosjekt

Høst 2001

Kravspesifikasjon

Versjon 1.14

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
24.09.2001	Opprettet og lagt inn innholdsfortegnelse	0.01	Odd Christian
28.09.2001	Gjort ferdig innledningen og alle småinnledningene.	0.02	Odd Christian
01.10.2001	La til funksjonelle og ikke-funksjonelle krav.	0.03	Atle
01.10.2001	Lagt til Brukstilfeller	0.04	Odd Christian
01.10.2001	Oppdatert Krav	0.05	Atle
02.10.2001	Lagt til Sekvensdiagram	0.06	Odd Christian
03.10.2001	Lagt til Testplan	0.07	Atle
03.10.2001	Oppdatert Testplan	0.08	Atle
03.10.2001	Fikset på Sekvensdiagram og småting	0.1	Odd Christian
04.10.2001	Oppdatert Testplan	0.11	Atle
04.10.2001	Lagt inn DFD osv.	0.20	Odd Christian
04.10.2001	Oppdatert Testplan	0.21	Atle
04.10.2001	Beskrivelse av dataflytdiagrammer	0.22	Atle
04.10.2001	Ny versjon av usecasebeskrivelse og mer tekst på oppgave og start på estimering	0.23	Odd Christian
04.10.2001	Integrasjon i bedriften	0.24	Atle
04.10.2001	Oppdaterte DFD	0.25	Odd Christian
04.10.2001	Diverse oppfiksing	0.30	Atle
05.10.2001	Klargjøring	0.31	Odd Christian
09.10.2001	Oppdatering av krav og testplan.	0.40	Atle
09.10.2001	Finpussing, Captions	0.41	Atle
09.10.2001	Forbedringer i forhold til forståelighet og sammenheng	0.42	Odd Christian
17.10.2001	Småfiksing	0.43	Atle
17.10.2001		0.44	Odd Christian
18.10.2001	Korrekturlesning	0.5	Ole Kristian
18.10.2001	Korrekturlesning	0.6	Martin
19.10.2001	Oppdaterte brukstilfeller og relevant informasjon, innføring av figurliste.	0.8	Martin
22.10.2001	Gjennomgang av layout	1.0	Ole Kristian
06.11.2001	Gjennomlesning	1.01	Martin
09.11.2001	Nye estimat og oppdatering av estimatkap.	1.10	Odd Christian
09.11.2001	Nytt estimat	1.11	Atle
13.11.2001	Korrekturlesing	1.14	Atle

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dokumentet skal være en kontrakt mellom leverandøren, som er prosjektgruppen, og kunden. Kontrakten skal fastslå hva som skal lages, med all funksjonalitet som kunden vil ha, og i hvilke omgivelser produktet skal fungere.

1.2 Avgrensning

Kravspesifikasjonen skal dekke alle krav til en mulig løsning. Kravene er spesielt rettet mot ny konstruert løsning, beskrevet i forstudiet [FOR], kapittel 8.3, da dette var konklusjonen på forstudiet.

1.3 Spesielle definisjoner

Se glossar dokumentet [GLO].

1.4 Dokumentreferanser

[GLO] Glossar, QPRO 2001
[FOR] Forstudiet, QPRO 2001

1.5 Dokumentoversikt

Hele forstudiet bør leses før kravspesifikasjonen, men spesielt viktig er kapittel 2, 3, 4 og 8. Dette dokumentet begynner med en overordnet beskrivelse av hva som skal lages i kapittel 2. Bruken av produktet visualiseres deretter med brukstilfellediagram i kapittel 3, som gir en oversikt over brukerens interaksjon med systemet. Sekvensen i interaksjonen er beskrevet i kapittel 4.

Den viktigste delen av dokumentet er kapittel 5, som vil fungere som kontrakten mellom utviklingsteamet og kunden. Her listes det opp alle funksjonelle og ikke-funksjonelle krav.

For å kunne integrere produktet i organisasjonen til kunden og synliggjøre en del krav som settes til organisasjonen ved bruk av produktet, gås det nærmere inn på de prosesser i bedriften som blir direkte eller indirekte affektert av produktet. Dette skjer i kapittel 6.

Validering av produktet skal skje mot slutten av prosjektet, og da trengs en testplan knyttet opp mot kravene. Testplanen er i kapittel 7. Til slutt skal tidsbruken for resten av prosjektet estimeres, ved hjelp av brukstilfeller og omgivelsesfaktorer. Dette gjøres i kapittel 8.

2 Oppgaven

Det skal lages et system som gjør starten på implementasjonsfasen i et systemutviklingsprosjekt raskere. Systemet skal generere kode ut fra modeller som er laget i analyse- og designfasen. Her er en grov oversikt over hva som skal produseres:

Høynivå konstruksjon	Konstruksjonen skal ligge til rette for å ta inn datamodeller, forretningsmodeller og brukstilfellemodeller, og generere databaseskript, dataaksesskomponenter, forretningslag og grafisk brukergrensesnitt.
Detaljert konstruksjon	Konstruksjonen skal spesifisere en løsning som tar inn datamodeller og forretningsmodeller, og genererer databaseskript, databaseaksesskomponenter og klasser med forretningslogikk. Designet skal være generelt nok til å kunne håndtere forskjellige komponentteknologier, som for eksempel COM og EJB 2.0.
Implementasjon	En demoversjon som tar inn en datamodell og genererer databaseskript og databaseaksesskomponenter for entitetene i datamodellen. Implementasjonen skal dekke en dialekt av SQL for databaseskjemagenerering og Container Managed Entity EJB for databaseaksess. Det skal planlegges hvordan UML interaksjonsdiagrammer kan tas inn og 'Session Beans' genereres. De skal være et utgangspunkt for forretningslogikken.

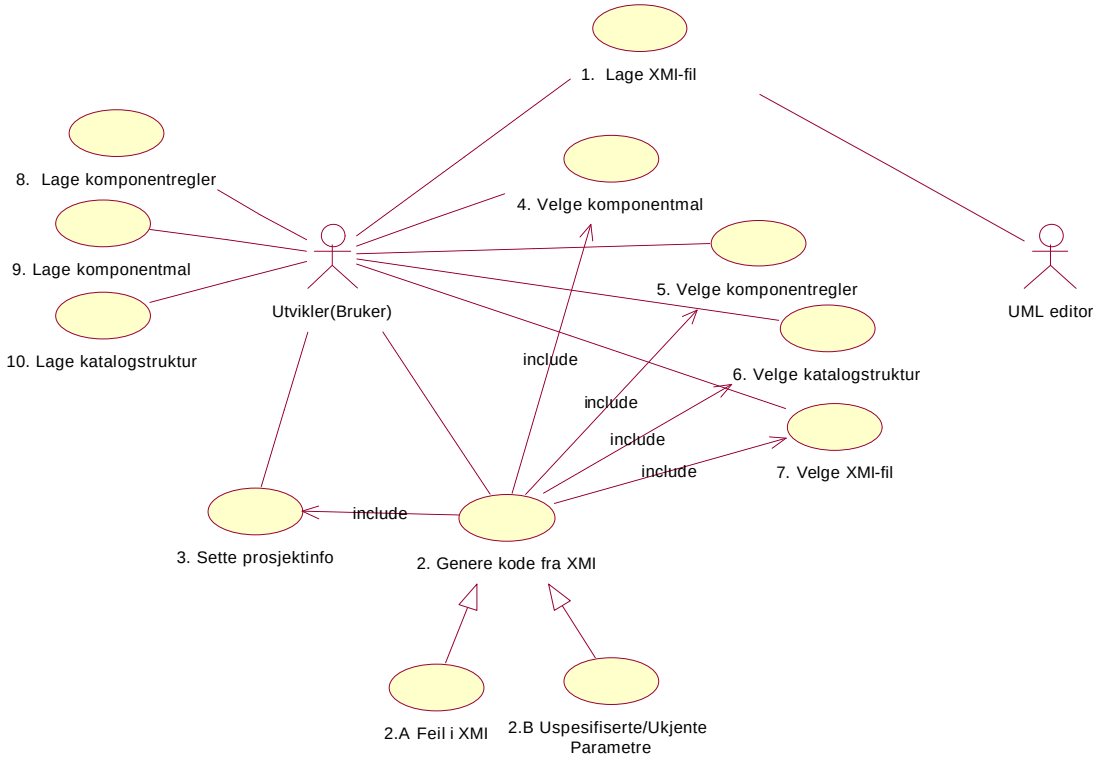
I tillegg skal det tas høyde for at støtte for nye komponentteknologier skal kunne lages av systemets brukere. I kapittel 8.3, En frittstående parser, i forstudiet [FOR] er støtten for nye teknologier samlet i 3 spesifiseringer: komponentmal, komponentregler og katalogstruktur. Under utvikling av systemet vil det lages støtte for utvalgte komponentteknologier, som for eksempel EJB. Det er likevel viktig at det er enkelt å utvide systemet til å støtte nye teknologier som dukker opp senere i produktets levetid uten at det trengs særlig kunnskap om hvordan produktet er bygd opp.

Siden produktet skal generere kode ut fra UML-modeller, er det ønskelig at UML-modellene ikke er nødt til å komme fra et bestemt verktøy. Produktet må da ta inn en generell representasjon av UML, for eksempel XMI. I resten av dokumentet refererer vi konsekvent til XMI, selv om det i teorien kan være andre generelle representasjoner av UML.

3 Brukstilfeller

Brukstilfellediagrammet gir en oversikt over brukerens interaksjon med systemet. Innholdet i modellen beskrives deretter tekstlig.

3.1 Diagram



Figur 3.1 – Brukstillfelldiagram

3.2 Aktører

Utvikler
Utvikleren er brukeren av produktet, og jobber i et systemutviklingsprosjekt. Jobben til utvikleren er å programmere ut fra et design.

UML editor
UML editoren er et eksternt system som kan eksportere XMI representasjon av UML.

3.3 Tekstlige beskrivelser

Hvert brukstilfelle beskrives ved hjelp av en standard tabell som inneholder alle relevante opplysninger. Tabellen som benyttes ser ut som:

Brukstilfelle	***NR. NAVN***
Beskrivelse	*En kort beskrivelse.*
Aktør	*Alle deltakende aktører.*
Trigger	*Den situasjonen som setter i gang brukstilfelle.*
Pre-betingelser	*Betingelser som må være oppfylt for at brukstilfellet kan settes i gang.*
Post-betingelser	*Betingelser som er oppfylt etter at brukstilfeller er fullført.*
Normal hendelsesflyt	*En kort nummerert liste over den normale hendelsesflyten.*
Variasjoner	*Mulige variasjoner som kan oppstå av den normale hendelsesflyten.*
Relatert informasjon	*En referanse til relaterte brukstilfeller*
Prioritet	*Brukstilfellet prioritet.*

Hvert brukstilfelle har en prioritet. Prioriteten er kritisk, høy, middels eller lav. Disse prioritetene samsvarer med de som blir gitt i kravlisten i kapittel 5.

Prioritetsklasser:
Kritisk – Brukstilfeller som det må designes funksjonalitet for og denne funksjonaliteten må implementeres i produktet.
Høy – Brukstilfeller som det må designes funksjonalitet for og denne funksjonaliteten bør implementeres i produktet.
Middels – Brukstilfeller som det bør designes funksjonalitet for, men som ikke nødvendigvis skal implementeres i produktet.
Lav – Brukstilfeller som det ikke behøver å ta spesielt høyde for i design- eller implementasjonsfasen. Dette er utvidelser av programmet som kan taes med om tiden strekker til.

Brukstilfelle	**1. Lage XMI-fil**
Beskrivelse	Utvikler bruker UML editor til å lage XMI av UML modellen han ønsker å generere kode for.
Aktør	Utvikler og UML editor
Trigger	Utvikler ønsker å lage XMI
Pre-betingelser	Det finnes UML modeller i et verktøy som kan lagre som XMI
Post-betingelser	En XMI-representasjon av en UML-modell lagret som en fil.
Normal hendelsesflyt	1. Utvikler åpner UML editor 2. Utvikler lagrer en UML modell som en XMI fil.
Variasjoner	-
Relatert informasjon	-
Prioritet	Middels

Brukstilfelle	**2. Generere kode fra XMI**
Beskrivelse	Det viktigste brukstilfellet, som omfatter den eneste funksjonaliteten som produserer noe. Dette brukstilfellet er enten direkte eller indirekte avhengig av alle de andre brukstilfellene.
Aktør	Utvikler
Trigger	Utvikler har satt i gang parseprogrammet
Pre-betingelser	ProsjektInfo er satt, komponentmal, komponentregler og katalogstruktur er tilgjengelig. Det er en XMI-fil tilgjengelig
Post-betingelser	Kode er generert og lagt ut i angitt katalogstruktur
Normal hendelsesflyt	1. Utvikler setter i gang parseren med XMIen som skal brukes 2. Utvikler setter opp prosjektinfoen 3. Utvikler velger komponentmal 4. Utvikler velger komponentregler 5. Utvikler refererer til katalogstruktur 6. Systemet sjekker parametrene 7. Systemet bruker XMI'en og genererer kode
Variasjoner	2.A Feil i XMI Det er noe galt med XMIen, og parseren gi melding til brukeren og avslutter prosessen. 2.B Uspesifiserte/Ukjente Parametere Noen av parametrene er ikke gitt eller er gitt feil. Parseren vil da gi melding til brukeren og avslutter prosessen.
Relatert informasjon	Brukstilfelle 3. Sette prosjektinfo Brukstilfelle 4. Velge komponentmal Brukstilfelle 4. Velge komponentregler Brukstilfelle 6. Velge katalogstruktur Brukstilfelle 7. Velge XMI-fil
Prioritet	Kritisk

Brukstilfelle	**3. Sette prosjektinfo**
Beskrivelse	Brukeren angir prosjektinformasjon som senere automatisk vil brukes når parseprosessen settes i gang. Prosjektinformasjonen kan inneholde prosjektnavn, ansvarlig person for UML, ansvarlig person for kode, dato for generering, databasesystem.
Aktør	Utvikler
Trigger	Utvikler har fått klarsignal for å kjøre parseprosess
Pre-betingelser	Utvikler har klar all prosjektinformasjon
Post-betingelser	Systemet har adgang til prosjektinformasjonen
Normal hendelsesflyt	1. Systemet gir mulighet til å gi inn informasjon 2. Utvikler legger inn all nødvendig informasjon
Variasjoner	-
Relatert informasjon	Brukstilfelle 2. Generere kode fra XMI
Prioritet	Middels

Brukstilfelle	**4. Velge komponentmal**
Beskrivelse	Brukeren bestemmer hvilken komponentmal som skal brukes i kodegenereringen.
Aktør	Utvikler
Trigger	Parseprosessen er klar til å ta imot referanseparameter for mal
Pre-betingelser	Komponentmal tilgjengelig
Post-betingelser	Parseprogrammet vet hvilken mal som skal brukes
Normal hendelsesflyt	1. Systemet gir bruker mulighet til å velge komponentmal 2. Utvikler velger en komponentmal som ligger tilgjengelig
Variasjoner	-
Relatert informasjon	Brukstilfelle 2. Generere kode fra XMI
Prioritet	Høy

Brukstilfelle	**5. Velge komponentregler**
Beskrivelse	Brukeren bestemmer hvilket komponentregelsett som skal brukes i kodegenereringen.
Aktør	Utvikler
Trigger	Parseprosessen er klar til å ta imot referanseparameter for regelsett
Pre-betingelser	Komponentregelsett tilgjengelig
Post-betingelser	Parseprogrammet vet hvilket sett som skal brukes
Normal hendelsesflyt	1. Systemet gir bruker mulighet til å velge komponentregelsett 2. Utvikler velger et komponentregelsett som ligger tilgjengelig
Variasjoner	-
Relatert informasjon	Brukstilfelle 2. Generere kode fra XMI
Prioritet	Høy

Brukstilfelle	**6. Velge katalogstruktur**
Beskrivelse	Brukeren bestemmer hvilken katalogstruktur som skal brukes i kodegenereringen.
Aktør	Utvikler
Trigger	Parseprosessen er klar til å ta imot referanseparameter for katalogstruktur
Pre-betingelser	Katalogstrukturspesifikasjon tilgjengelig
Post-betingelser	Parseprogrammet vet hvilken struktur som skal brukes
Normal hendelsesflyt	1. Systemet gir bruker mulighet til å velge katalogstruktur 2. Utvikler velger en katalogstruktur som ligger tilgjengelig
Variasjoner	-
Relatert informasjon	Brukstilfelle 2. Generere kode fra XMI
Prioritet	Høy

Brukstilfelle	**7. Velge XMI-fil**
Beskrivelse	Brukeren bestemmer hvilken XMI-fil som skal brukes i kodegenereringen.
Aktør	Utvikler
Trigger	Parseprosessen er klar til å ta imot referanse til XMI-filen
Pre-betingelser	XMI-filen tilgjengelig
Post-betingelser	Parseprogrammet vet hvilken XMI-fil som skal brukes
Normal hendelsesflyt	1. Systemet gir bruker mulighet til å velge XMI-fil 2. Utvikler velger ønsket XMI-fil
Variasjoner	-
Relatert informasjon	Brukstilfelle 2. Generere kode fra XMI
Prioritet	Høy

Brukstilfelle	**8. Lage komponentmal**
Beskrivelse	Brukeren setter opp skjelettet for komponentene som skal lages i kodegenereringen. Komponentmalen skal senere kunne refereres til, på samme måte som for komponentteknologier systemet allerede støtter.
Aktør	Utvikler
Trigger	Utvikler skal bruke parseren, men trenger støtte for ny komponentteknologi som den genererte koden skal følge.
Pre-betingelser	
Post-betingelser	Det finnes en komponentmal klar til å bli referert til i parseprosessen
Normal hendelsesflyt	-
Variasjoner	-
Relatert informasjon	-
Prioritet	Lav

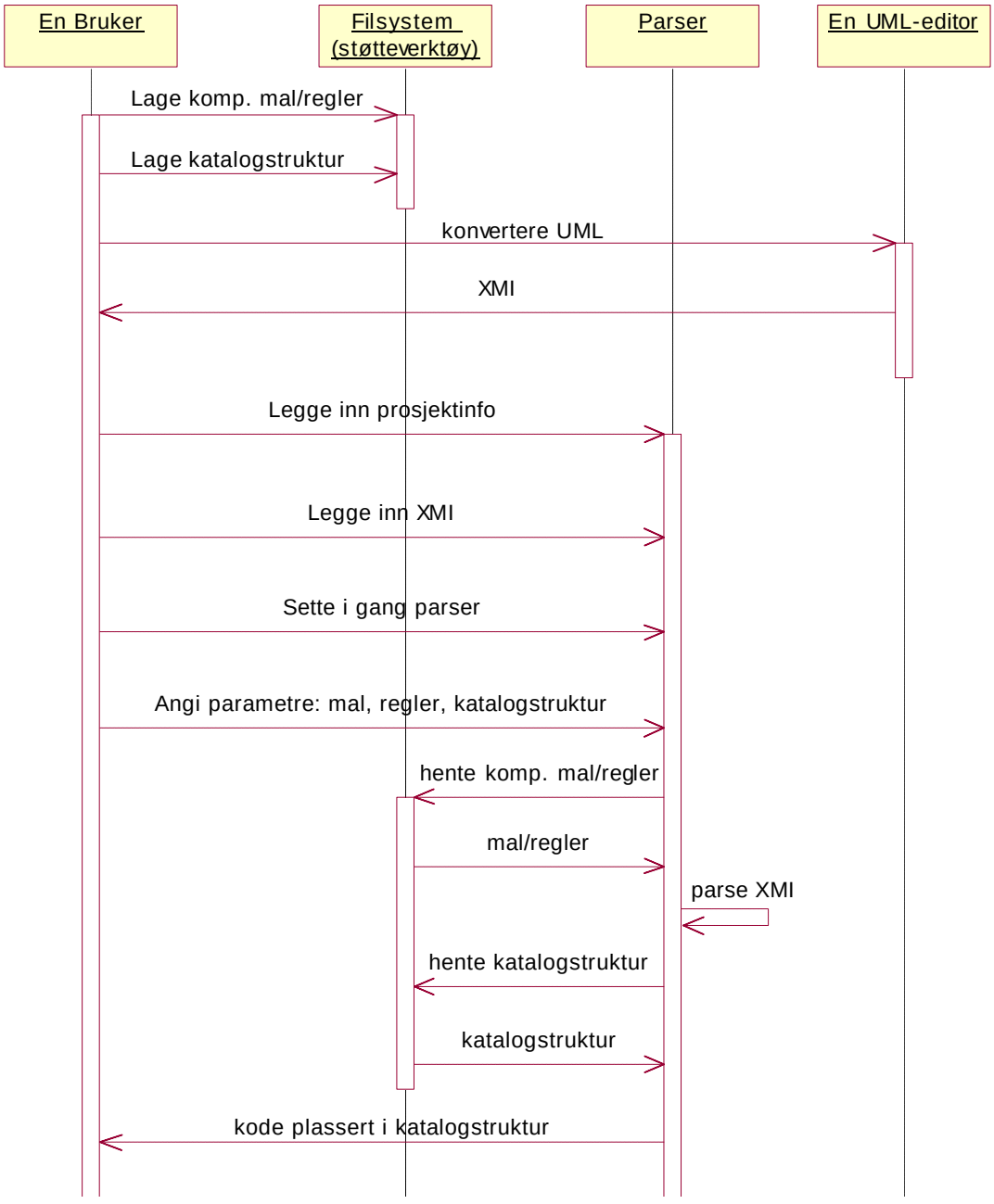
Brukstilfelle	**9. Lage komponentregler**
Beskrivelse	Brukeren lager et sett av regler for hva slags kode som skal genereres ut fra elementer i XMI. Komponentreglene skal senere kunne refereres til på samme måte som for komponentteknologier systemet allerede støtter.
Aktør	Utvikler
Trigger	Utvikler skal bruke parseren, men trenger støtte for ny komponentteknologi som den genererte koden skal følge.
Pre-betingelser	
Post-betingelser	Det finnes et regelsett klar til å bli referert til i parseprosessen
Normal hendelsesflyt	-
Variasjoner	-
Relatert informasjon	-
Prioritet	Lav

Brukstilfelle	**10. Lage katalogstruktur**
Beskrivelse	Brukeren lager en katalogstruktur som skal brukes når koden genereres og legges i filer på disk. Katalogstrukturen skal senere kunne refereres til når parsingen settes i gang.
Aktør	Utvikler
Trigger	Utvikler skal bruke parseren, men trenger støtte for ny komponentteknologi som den genererte koden skal følge.
Pre-betingelser	
Post-betingelser	Det finnes en katalogstruktur klar til å bli referert til i parseprosessen
Normal hendelsesflyt	-
Variasjoner	-
Relatert informasjon	-
Prioritet	Lav

Det er to brukstilfeller hver for komponentmal, sett av regler og for katalogstruktur for å gi muligheten til å skille viktig funksjonalitet fra mindre viktig funksjonalitet. En ting er å velge komponentmaler, regler og katalogstruktur for eksisterende komponentteknologier. En annen ting er å lage nye maler osv. for teknologier etter hvert som de dukker opp.

4 Sekvens

Sekvensdiagram brukes for å illustrere sekvensen i interaksjonen mellom bruker, system og eksterne system. Diagrammet beskriver normal flyt. Forutsetningen her er at brukeren har behov for å lage støtte for en ny komponentteknologi før parsingen settes i gang.



Figur 4.1 – Overordnet sekvensdiagram

’En Bruker’ er ’Utvikler/Bruker’ fra brukstilfellediagrammet i kapittel 3, Brukstilfeller. ’Parser’ og ’Filsystem/Støtteverktøy’ utgjør systemet som brukeren interagerer med i brukstilfellediagrammet, og ’En UML-editor’ tilsvarer ’UML-editor’ i brukstilfellediagrammet.

Etter at Brukeren har bestemt seg for hvilken komponentteknologi ’Parser’ skal generere kode i bruker han et verktøy i systemet for utvikling av ny komponentmal og kodegenereringsregler. Enten ved hjelp av et verktøy eller i en flat fil i filsystemet lager ’En Bruker’ en katalogstruktur som skal brukes av ’Parser’ når kode genereres og legges ut. Fra en tidligere fase har man en designspesifikasjon klar i form av UML-modeller. Disse UML-modellene konverteres til XMI i et UML-verktøy.

’Parser’ får prosjektinfo og XMI, og deretter setter ’En Bruker’ i gang parseprosessen. Brukeren angir de nødvendige parametere for at Parseren skal kunne generere kode ut ifra XMIen: referanser til komponentmal, sett av komponentregler og katalogstruktur. Parseren bruker disse referansene for å finne de tre nødvendige parametrene, generere kode og legge koden i angitt katalogstruktur.

5 Kravliste

Kravlisten utgjør kontrakten mellom utviklerteam og kunde. Nedenfor følger en nummerert liste over krav og deres målekriterier. Kravene er kategorisert i funksjonelle og ikke-funksjonelle krav. Alle kravene er dessuten prioritert i en av prioritetsklassene kritisk, høy, middels eller lav. Disse klassene står forklart nedenfor.

Prioritetsklasser:

Kritisk – Krav som det må designes funksjonalitet for og denne funksjonaliteten må implementeres i produktet.
Høy – Krav som det må designes funksjonalitet for og denne funksjonaliteten bør implementeres i produktet.
Middels – Krav som det bør designes funksjonalitet for, men som ikke nødvendigvis skal implementeres i produktet.
Lav – Krav som det ikke behøver å ta spesielt høyde for i design- eller implementasjonsfasen. Dette er utvidelser av programmet som kan taes med om tiden strekker til.

5.1 Funksjonelle krav

De funksjonelle kravene sier hva brukeren skal kunne gjøre med produktet og hva produktet skal kunne produsere. Kravene er nært knyttet til brukstilfeller og vi har derfor valgt å ha dette som en egen kolonne for å få koblet kravene til brukstilfellene. Kolonnen for volum forsøker å gi et anslag på hvor mye arbeid dette kravet utgjør.

5.1.1 Krav til inndata

ID	Beskrivelse	Prioritet	Volum	Brukstilfelle
Inndata Krav 1	Inndata skal være en generell representasjon av en UML modell representert i XMI.	Kritisk	10%	1,2
Inndata Krav 2	Programmet skal kunne ta inndata ved oppstart for minimum dette: - katalog hvor kildekode skal legges til - hvilken arkitektur koden skal genereres for	Høy	2%	4,5,6,7
Inndata Krav 3	Programmet skal ha muligheter for å sette generelle parametere slik som prosjektnavn.	Middels	2%	3

ID	Beskrivelse	Prioritet	Volum	Brukstilfelle
Inndata Krav 4	Ved feil i inndata skal det genereres en beskrivende feilmelding som sier noe om hvilken linje det gjelder og hva som er problemet.	Middels	4%	2A,2B (alternativ flyt)
Inndata Krav 5	Programmet skal spesielt kunne støtte import av XMI fra Rational Rose.	Kritisk	6%	1,2
Inndata Krav 6	Programmet skal spesielt kunne støtte import av XMI fra Select.	Middels	6%	1,2
Inndata Krav 7	Programmet skal ikke stille så store og komplekse krav til XMI-inndata formatet at brukeren selv ikke kan lage gyldige dokumenter.	Middels	5%	2

5.1.2 Krav til parse- og genereringsprosess

ID	Beskrivelse	Prioritet	Volum	Brukstilfelle
Parse Krav 1	Programmet skal kunne generere kode for EJB versjon 2.0.	Kritisk	6%	1 (og 4,5,6)
Parse Krav 2	Man skal også kunne konfigurere programmet til å kunne generere kode for andre tilsvarende teknologier og fremtidige versjoner av disse.	Middels	15%	4,5,6
Parse Krav 3	Programmet skal kunne generere kode for COM i Visual Basic.	Lav	2%	1 (og 4,5,6)
Parse Krav 4	Brukeren skal enkelt kunne lage sine egne regler og maler uten kunnskap om hvordan systemet er laget.	Lav	5%	8,9,10
Parse Krav 5	Programmet skal automatisk generere ulike SQL dialekter avhengig av databaseprogramvaren som benyttes.	Lav	5%	2
Parse Krav 6	Det skal være mulig å generere kun et subsett, det vil si kun SQL, data aksessobjekter, forretningsobjekter eller GUI, av den totale koden.	Middels	3%	2,3

5.1.3 Krav til utdata

ID	Beskrivelse	Prioritet	Volum	Brukstilfelle
Utdata Krav 1	Programmet skal lage SQL-script for opprettelse av databasen.	Kritisk	3%	2
Utdata Krav 2	Programmet skal lage aksessobjekter mot tabellene i databasen.	Kritisk	3%	2
Utdata Krav 3	Programmet skal lage aksessobjekter for de ulike forretningsprosessene.	Middels	5%	2
Utdata Krav 4	Programmet skal lage brukergrensesnitt for de ulike forretningsprosessene.	Lav	10%	2
Utdata Krav 5	Programmet skal generere SQL kode støttet av MS SQL server.	Kritisk	2%	2
Utdata Krav 6	Programmet skal kunne generere SQL kode for andre databasesystemer implementert ved hjelp av templates.	Middels	6%	2

5.2 Ikke-funksjonelle krav

Ikke-funksjonelle krav er alle krav som ikke har med systemets funksjonalitet å gjøre, men handler mer om omgivelsene produktet skal fungere i.

5.2.1 Krav til leveranser

ID	Beskrivelse	Prioritet
Lever Krav 1	Ferdig utviklet produkt leveres EDB ASA senest 15. november 2001.	Kritisk

5.2.2 Krav til maskinvare og programvare

ID	Beskrivelse	Prioritet
Maskin Krav 1	Programmet skal kunne kjøre på en 350 MHz med 128 MB RAM eller bedre.	Høy
Maskin Krav 2	Programmet skal kunne kjøres i et MS Windows miljø.	Kritisk
Maskin Krav 3	Programmet skal unngå maskinvareavhengighet og støtte andre operativsystemer enn MS Windows.	Lav

5.2.3 Krav til tidsforbruk

ID	Beskrivelse	Prioritet
Tid Krav 1	Tidsforbruket for en typisk modell (10 data aksessobjekter og 10 brukstilfeller) skal ikke være enn 5 minutter å generere all kode.	Høy
Tid Krav 2	Et subsett av koden skal ikke isolert sett bruke mer enn to minutter på å genereres.	Middels

5.2.4 Krav til utviklingsmiljø

ID	Beskrivelse	Prioritet
Miljø Krav 1	Programmet skal utvikles i Java (Suns JDK 1.3).	Høy
Miljø Krav 2	Tredjeparts pakker som ønskes brukt må godkjennes av oppdragsgiver.	Kritisk

5.2.5 Krav til programkode og dokumentasjon

ID	Beskrivelse	Prioritet
Kode Krav 1	Koden til programmet skal skrives etter Suns anbefalinger.	Høy
Kode Krav 2	Programmet skal konstrueres i fornuftige klasser for å øke vedlikeholdbarheten.	Høy
Kode Krav 3	All kildekode til programmet skal kommenteres i henhold til JavaDoc standarden.	Høy
Kode Krav 4	Leveransen skal inneholde en enkel beskrivelse av hvordan systemet installeres og gjøres klar til bruk. I tillegg skal det følge med en enkel brukerdokumentasjon som beskriver de ulike funksjoner og hvordan disse kan utføres.	Middels
Kode Krav 5	Systemdokumentasjonen til programmet skal genereres vha. JavaDoc.	Høy

5.3 Krav til brukergrensesnitt

Det eksisterer ikke noe krav til et grafisk brukergrensesnitt for generatoren som vi skal implementere, men et eller annet brukergrensesnitt må vi nødvendigvis ha. I første omgang tenker vi oss et kommandobasert verktøy der alle parametere gis inn som tekstlige beskrivelser og referanser. En senere mulig utvidelse vil naturlig nok være å lage et lite grafisk brukergrensesnitt, men dette blir uansett av lav prioritet.

5.4 Andre krav

Pålitelighet – Hvis det skjer noe uforutsett skal programmet kunne håndtere dette og levere en forståelig feilmelding til brukeren, før programmet avsluttes.

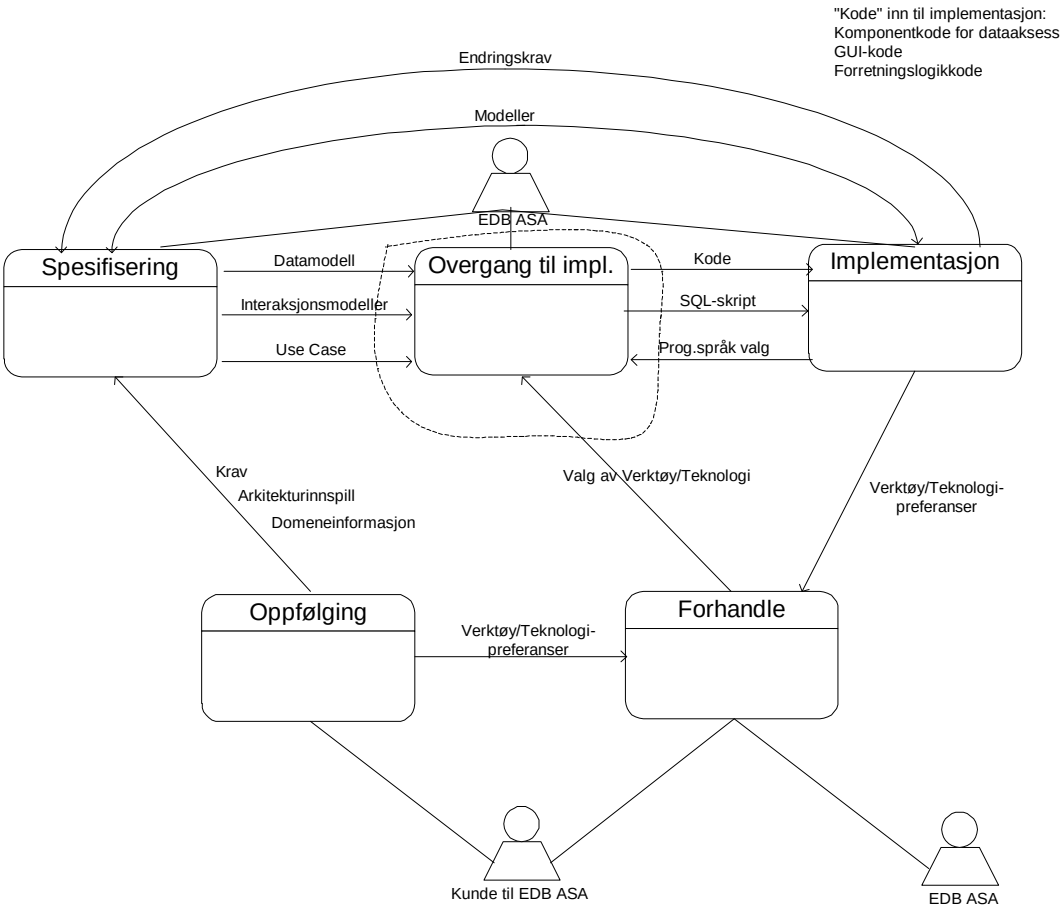
Vedlikeholdbarhet – Programmet skal implementeres med fyldige kommentarer i kildekoden slik at det blir enkelt å endre på koden senere dersom dette blir nødvendig.

Utvidbarhet – Selv om ikke alt implementeres i første omgang skal programmet ta høyde for senere utvidelser i henhold til de funksjonelle kravene slik at dette kan utføres uten å måtte redesigne hovedarkitekturen fullstendig.

6 Integrasjon i bedriftens dataflyt

EDB ASA har funnet ut at det i oppstart av EJB prosjekter tar relativt lang tid å få generert en kodeskjellett manuelt. En automatisk generering av kode vil føre til innspart tid. Dette kan bli et konkurransefortrinn ovenfor konkurrenter ved at EDB ASA kan tilby raskere leveringstider og lavere totalpris (Les kapitel 2 Nåsituasjon i forstudiet [FOR]). Dette kapittelet tar for seg hvordan produktet kan brukes i prosjekter hos EDB ASA.

6.1 Dataflyt med produktets omgivelser

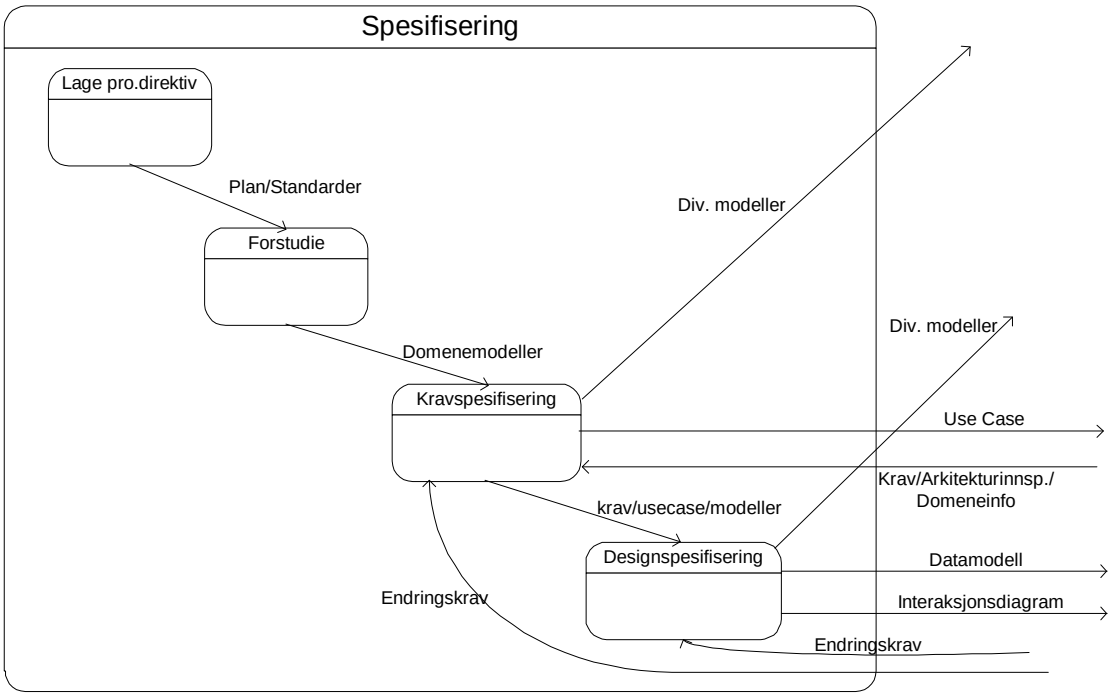


Figur 6.1 – Førsteordens dataflytdiagram (DFD)

Førsteordens dataflytdiagram (DFD) har vi valgt å dele opp i 5 prosesser. Dette er ’Spesifisering’, ’Overgang til implementasjon’, ’Implementasjon’, ’Oppfølging’ og ’Forhandle’. Systemet som skal utvikles befinner seg innenfor den prikkede linjen. Det er kun aktuelt å ta med to aktører, nemlig EDB ASA og deres kunde. ’Spesifisering’-prosessen mottar krav, arkitekturinnspill og domeneinformasjon via oppfølging fra kunden til EDB ASA. Spesifikasjonen sendes videre i form av datamodeller,

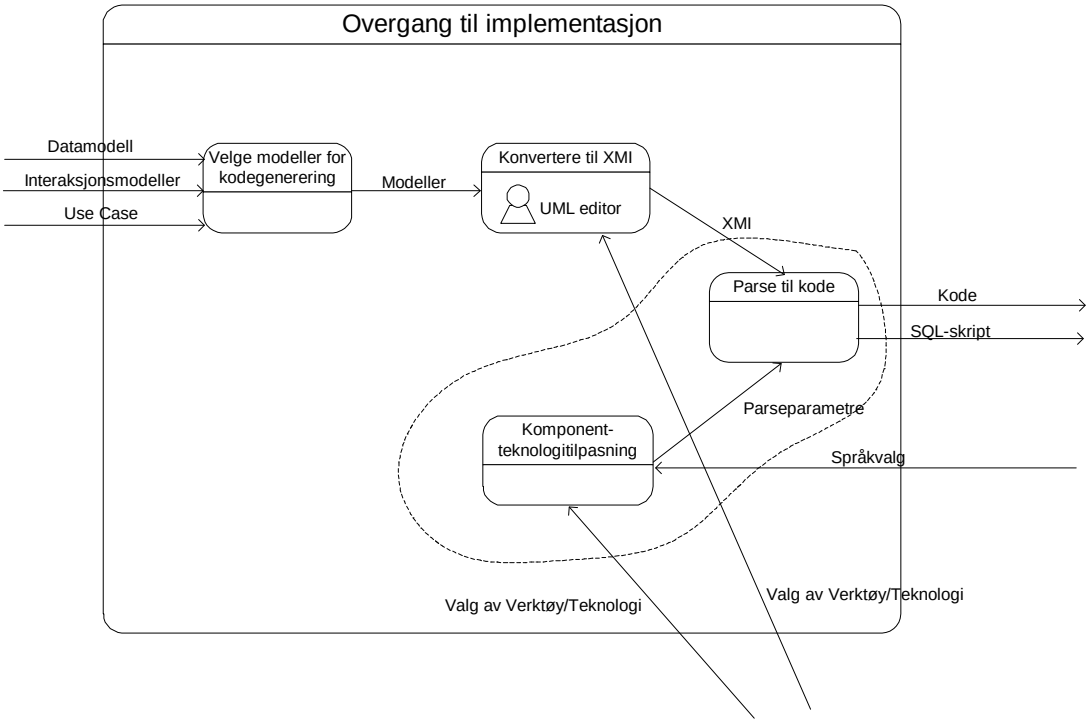
interaksjonsmodeller og brukstilfeller til 'Overgang til implementasjon'. Denne fasen mottar valg av verktøy og teknologi fra forhandlinger mellom EDB ASA og kunden deres. Kundens oppfølgingsprosess og starten av implementasjonsprosessen resulterer blant annet i preferanser fra begge parter innen verktøyvalg og teknologivalg. Dette er utgangspunkt for forhandlingene. Implementasjonsfasen mottar kode og SQL script fra overgang til implementasjon og modeller fra spesifikasjonsfasen, slik at implementasjonen kan fortsette. Eventuelle endringer fører til endringskrav. Disse endringskravene vil igjen føre til at spesifikasjonene må endres til å møte de ny kravene. Denne dataflyten fører til en løkke i diagrammet, noe som egentlig ikke er lov i DFD-notasjonen, men er tatt med fordi endringer i design er et viktig aspekt.

Vi har valgt å dekomponere de tre viktigste boksene fra førsteordens dataflytdiagram (DFD). Dette er spesifikasjon, overgang til implementasjon og implementasjon.



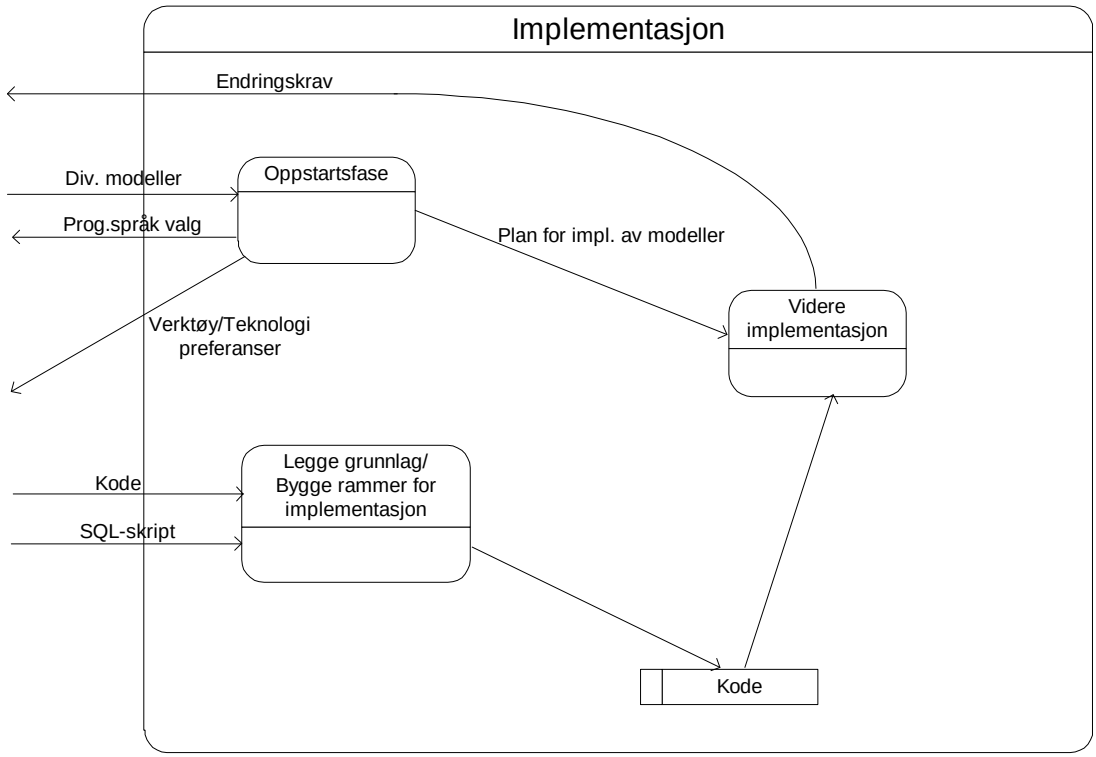
Figur 6.2 – Andreordens dataflytdiagram (DFD) for dekomponering av 'Spesifisering'

Spesifikasjonen starter med utarbeidelse av prosjektdirektiv. Planer og standarder går videre inn i et forstudie der det skal gjøres et grundig forarbeid for prosjektet. Forarbeidet resulterer i domenemodeller og til slutt kravspesifikasjon. Det er her det blir laget brukstilfeller som er viktige i overgangen til implementasjonsfasen. Krav, brukstilfeller og modeller fra kravspesifiseringen går videre til en designspesifisering som resulterer datamodeller og interaksjonsdiagram.



Figur 6.3 – Andreordens dataflytdiagram (DFD) for dekomponering av 'Overgang til implementasjon'

Systemet som skal utvikles befinner seg innenfor den prikkede linjen. Overgang til implementasjonsfasen får inn datamodeller, interaksjonsmodeller og brukstilfeller fra spesifikasjonsfasen. Disse brukes til å velge modeller for kodegenerering. Modellene må konverteres til XMI slik at disse kan bli parset av vårt produkt. XMI og parseparametere er avhengig av komponentteknolgitilpasning. Hvilken komponentteknologi som benyttes avhenger av språkvalg. Denne parsingsprosessen resulterer i kode og SQL-skript som sendes videre til implementasjon.



Figur 6.4 – Andreordens dataflytdiagram (DFD) for dekomponering av 'Implementasjon'

Implementasjonsprosessen består av en oppstartsfase der det kommer inn nye modeller som brukes til videre implementasjon. Den videre implementasjonen bygger på den komponentkoden og SQL koden som kommer inn fra parseren. Denne kan eventuelt tenkes lagret i et kodelager. I implementasjonen kan det dukke opp behov for å endre på designet, slik at spesifiseringsfasen forlenges på bakgrunn av korresponderende endringskrav.

6.2 Integrasjon

Systemet som skal utvikles vil brukes i 'Overgang til Implementasjon', i figur 6.1, Førsteordens DFD. Denne prosessen må ha inn modeller fra 'Spesifisering'-prosessen og noen teknologiske valg fra 'Implementasjon'-prosessen og 'Forhandle'-prosessen. Som vi ser av figur 6.3, Andreordens DFD for 'Overgang til Implementasjon', er alle de nevnte inndata til denne prosessen nødvendige for at produktet skal få gjort jobben sin.

Produktet må ha inn utvetydige og komplette modeller og rammebetingelser, hvis produktet skal generere meningsfull kode for resten av implementasjonsfasen. Dette stiller klare krav til omgivelsene (se kapittel 2 og 3 i forstudiet [FOR]):

1. Spesifiseringen må gjennomføres på en så god måte at datamodellen inneholder alle nødvendige entiteter og relasjoner, brukstilfellemodellen inneholder alle brukstilfeller og så videre.
2. Dialogen med kunden i ’Forhandle’-prosessen må være såpass tett at avgjørelser innen teknologivalg, for eksempel foretrukket komponentteknologi, og bruk av verktøy, for eksempel UML-verktøy, er avgjort før man går i gang med ’Overgang til implementasjon’.
3. ’Implementasjon’-prosessen må dessuten ha kommet såpass godt i gang at det er avgjort hvilket programmeringsspråk og hvilken kodestandard som skal brukes.

Dataflyten som det vises til i punkt 1 over, mellom ’Spesifisering’ og ’Overgang til Implementasjon’ fører også til krav på produktet. I inndata krav 4 i kapittel 5.1.1, Krav til inndata, stilles det krav til at modell-inputen, i form av XMI, skal valideres når den gjennomgår parseprosessen. Hvor detaljert denne valideringen skal være, er ikke spesifisert. Ideelt sett skal produktet si fra hvis for eksempel en datamodell ikke inneholder nok relasjoner til at alle entiter kan nås. Dette aspektet ved kravet kan også dekkes av validering i en UML-editor. Minimum er at produktet sier fra hvis XMI-en mangler noe elementært, som for eksempel alle entitetene.

I figur 6.3, Andreordens DFD for ’Overgang til implementasjon’, ser vi at produktet er involvert i to prosesser: ’Komponentteknologittilpasning’ og ’Parse til kode’.

Produktet må tilby en enkel måte for brukeren å sørge for, i ’Komponentteknologi-tilpasning’, at kode kan genereres innen en teknologi som kunden og EDB ASA er blir enige om. Dette forbedrer håndteringen av punkt 2 og tas hånd om av blant annet krav 4 i kapittel 5.1.2, Parse krav. Språkvalget i ’Implemtasjon’-prosessen, nevnt i punkt 3, legger også føringer på ’Komponentteknologitilpasning’-prosessen.

’Parse til kode’ er den andre prosessen produktet er involvert i, og den trenger en XMI-fil, som kan være laget i et UML-verktøy. Kunden til EDB ASA kan ha ønsker og krav knyttet til hvilke verktøy som brukes, og dette må være avklart før prosessen ’Overgang til Implementasjon’ starter.

7 Testplan

Testplanen beskriver tester som skal validere systemet i forhold til kravene. Dette må betraktes som en akseptansetest
der vi skal forsøke å avgjøre om kravene fremsatt fra kunde er oppfylt i det som eventuelt er blitt implementert.

7.1 Teststrategi

Testbeskrivelsen nedenfor inneholder en punktliste med sjekkpunkter. Disse kan brukes til test eller evaluering av produktet. Ettersom vi enda ikke vet helt hvordan produktet vil bli seende ut blir dette en slags svart boks test der vi forsøker å koble de ulike testmomentene opp mot kravene. Hvert sjekkpunkt inneholder derfor referanser til hvilke krav som testes. Test av et ferdig produkt eller en prototyp vil dermed foregå på den måten at vi går igjennom sjekklista og forsøker å besvare spørsmålene. Ofte er svaret kun ja eller nei, men en fyldigere kommentar legges til der dette er nødvendig.

7.2 Testbeskrivelse

Test mot inndata krav

Test ID	**Beskrivelse**	**Mål**	**Krav ID**
Inndata Test 1	Kan programmet ta en generell UML modell som inndata?	Produktet passerer om det kan ta en XMI beskrivelse for en UML modell som inndata, enten denne er manuelt skrevet inn eller eksportert fra et eksisterende programvareprodukt. Om dette oppfylles vil kravet om generelle parametere også gå igjennom ettersom dette kan defineres ved hjelp av XMIen.	Inndata Krav 1, 3
Inndata Test 2	Kan programmet velge i hvilke kataloger kildekoden som genereres skal legges i?	Produktet passerer om det legger ut ulike deler av resultatkoden i henhold til det som defineres før kjøring, for eksempel ved kommandolinjeparametere.	Inndata Krav 2

Test ID	Beskrivelse	Mål	Krav ID
Inndata Test 3	Kan programmet velge hvilken arkitektur det skal genereres kode for?	Produktet passerer om det er mulig å velge å generere ikke bare EJB kode, men også kode for andre tilsvarende arkitekturer.	Inndata Krav 2
Inndata Test 4	Dukker det opp en feilmelding i programmet hvis inndata er av et uforståelig format?	Produktet passerer om det ved valg av inndatafiler som opplagt ikke skal bli taklet dukker opp beskrivende feilmelding som sier at inndata er av feil format.	Inndata Krav 4
Inndata Test 5	Kan programmet importere XMI fra Rose?	Produktet passerer om en enkel forhåndsdefinert UML modell generert i Rose og eksportert til XMI lar seg bruke som XMI.-inndata til kodegeneratoren.	Inndata Krav 5
Inndata Test 6	Kan programmet importere XMI fra Select?	Produktet passerer om den samme forhåndsdefinerte UML modellen som i punktet ovenfor generert i Select og eksportert til XMI lar seg bruke som XMI-input til kodegeneratoren.	Inndata Krav 6
Inndata Test 7	Er det mulig innen en rimelig tidsfrist å manuelt endre eller skape inndata til kodegeneratoren ?	Produktet passerer om det er slik at den XMI-inputen som kreves av kodegeneratoren er godt forståelig slik at man kan gå manuelt inn i denne koden hvis det skulle være nødvendig.	Inndata Krav 7

Test mot parse- og genereringsprosess krav

Test ID	Beskrivelse	Mål	Krav ID
Parse Test 1	Kan programmet generere kode for EJB 2.0?	Produktet passerer om man kan velge å generere EJB 2.0 kode og at dette valget gir korrekt utdata. Dette innebærer at EJB koden som blir generert lar seg kompilere uten feil.	Parse Krav 1
Parse Test 2	Kan programmet generere kode for COM språk?	Produktet passerer om man kan velge å generere kode for eksempelvis Visual Basic og at koden som produseres lar seg kompilere uten feil.	Parse Krav 2, 3
Parse Test 3	Tar det under 5 minutter å generere all kode for en normal modell med 10 data aksessobjekter og 10 brukstilfeller?	Produktet passerer om det for en forhåndsdefinert UML modell med de gitte spesifikasjoner eksportert til XMI ikke tar over 5 minutter å generere all koden.	Tid Krav 1
Parse Test 4	Tar det under 2 minutter å generere et subsett (SQL, data aksessobjekter, forretningsobjekter, GUI)?	Produktet passerer om det for en forhåndsdefinert UML modell med de samme spesifikasjoner som i punktet over eksportert til XMI ikke tar over 2 minutter å generere et subsett av koden.	Tid Krav 2, Parse Krav 6
Parse Test 5	Kan programmet automatisk generere ulike SQL dialekter for ulike databasesystemer?	Produktet passerer om det er mulig å generere SQL dialekter for Oracle, MySQL og MS SQL. Dessuten så må man se at den koden som er blitt generert virkelig fungerer.	Parse Krav 5, Utdata Krav 6
Parse Test 6	Er det mulig å enkelt kunne lage sine egne regler og maler?	Produktet passerer om det er tatt høyde for i produktet at man kan lage sine egne regler ved hjelp av maler og at utdataene ved bruk av disse malene blir som forventet.	Parse Krav 4

Test mot utdata krav

Test ID	Beskrivelse	Mål	Krav ID
Utdata Test 1	Kan programmet generere SQL kode som oppretter databasen?	Produktet passerer om det er mulig å opprette SQL databasen automatisk med kode i stedet for å måtte taste denne inn manuelt.	Utdata Krav 1
Utdata Test 2	Kan programmet generere aksessobjekter mot tabellene i databasen?	Produktet passerer om det er mulig å generere EJB entitetsbønner.	Utdata Krav 2
Utdata Test 3	Kan programmet generere aksessobjekter for forretningsprosesser?	Produktet passerer om det er mulig å generere EJB sesjonsbønner.	Utdata Krav 3
Utdata Test 4	Kan programmet generere brukergrensesnitt for forretningsprosessene?	Produktet passerer om det er mulig å generere brukergrensesnitt for EJB sesjonsbønnene.	Utdata Krav 4
Utdata Test 5	Kan programmet generere SQL kode støttet av MS SQL server?	Produktet passerer om SQL koden som genereres er kompatibel med MS SQL server. Det må altså testes både at koden utfører operasjoner på serveren, og at disse operasjonene blir utført på en korrekt måte.	Utdata Krav 5

Test mot maskinvare- og programvare krav

Test ID	Beskrivelse	Mål	Krav ID
Maskin Test 1	Kan programmet kjøres på en 350 MHz Pentium med 256 MB minne?	Produktet testes på en maskin med nettopp disse spesifikasjonene. Dette er avgjørende for målingene av tid.	Maskin Krav 1
Maskin Test 2	Kan programmet kjøres i et MS Windows operativsystem?	Produktet passerer om alle krav med kritisk og høy prioritet seg kjøre i et MS Windows operativsystem. Produktet passerer om alle krav med kritisk og høy prioritet passerer.	Maskin Krav 2

Test ID	Beskrivelse	Mål	Krav ID
Maskin Test 3	Kan programmet kjøres i et UNIX/Linux operativsystem?	Dette kravet testes på samme maskin som ovenfor, men denne omstartes om til FreeBSD Linux. Produktet passerer om alle krav med kritisk og høy prioritet passerer.	Maskin Krav 3

Test mot utviklingsmiljø krav

Test ID	Beskrivelse	Mål	Krav ID
Miljø Test 1	Er programmet utviklet i henhold til Sun JDK 1.3?	Produktet passerer om det følger hovedtrekkene i Sun JDK 1.3.	Miljø Krav 1, Kode Krav 1
Miljø Test 2	Er det blitt benyttet tredjeparts pakker?	Produktet passerer om det ikke er blitt benyttet tredjeparts pakker hvis ikke disse er blitt godkjent av kunden.	Miljø Krav 2

Test mot generatorkode og dokumentasjon krav

Test ID	Beskrivelse	Mål	Krav ID
Kode Test 1	Er klassene fornuftige?	Produktet passerer om den generatoren er inndelt i fornuftige klasser. Dette må diskuteres med kunde.	Kode Krav 2
Kode Test 2	Er generatorkoden kommentert med JavaDoc?	Produktet passerer om all den koden som generatoren består av er godt kommentert med JavaDoc i henhold til Sun sin standard.	Kode Krav 3
Kode Test 3	Følger det med en installasjons- og brukerdokumentasjon?	Produktet passerer om det følger med dokumentasjon som gjør en utenforstående i stand til å installere og bruke produktet.	Kode Krav 4

7.3 Kriterier for godkjenning og underkjenning

Når test er gjennomført vil resultatene bli tatt opp med gruppa. Produktet vil da bli godkjent eller underkjent avhengig av hvor mange testmomenter som ikke passerte testen og prioriteten på de som ikke passerte testen. Testmomenter med kritisk eller høy prioritet må passere testen før leveranse til kunde. En test kan stoppes helt dersom det avdekkes feil som umuliggjør gjennomføringen av testen. Alle testresultater skal rapporteres til kvalitets- og testansvarlig som har ansvaret for å offentliggjøre disse på prosjektmøtet.

8 Estimering av tidsbruk

Ut ifra brukstilfeller og omgivelsesfaktorer estimeres tidsbruk i design og implementasjonsfasen.

8.1 Full versjon

Estimerer tidsbruk i konstruksjons- og implementasjonsfasen, gitt at vi skal implementere alle brukstilfeller.

Kompleksitet

Aktører	**Kompleksitet**
Utvikler	Gjennomsnittlig
UML editor	Enkelt

Brukstilfelle	
1 Lage XMI	Enkelt
2 Generere kode fra XMI	Gjennomsnittlig
3 Sette prosjektinfo	Enkelt
4 Velge komponentmal	Enkelt
5 Velge komponentregler	Enkelt
6 Velge katalogstruktur	Enkelt
7 Velge XMI	Enkelt
8 Lage komponentmal	Gjennomsnittlig
9 Lage komponentregler	Gjennomsnittlig
10 Lage katalogstruktur	Gjennomsnittlig

Estimering
Estimert tidsforbruk, design- og implementasjonsfase: 829,28 timer
Se Vedlegg 1 for Excel regneark med brukstilfelle estimering av tidsforbruket.

Kompleksiteten av brukstilfellene er anslått ut ifra hvor komplisert interaksjonen mellom bruker og system er. Systemet i dette tilfellet vil ha et svært enkelt grensesnitt, men operasjonene som skal utføres i parseprosessen er ikke enkle. Derfor kan estimatet være langt mindre enn det reelle tidsforbruket. Sammenhengen mellom maler/regler, som gjør produktet fleksibelt i forhold til. teknologi, og parseprosessen er en faktor som kompliserer utviklingen i stor grad, og vil sannsynligvis føre til større tidsbruk enn det som her er estimert.

8.2 Skrellet versjon

Estimerer tidsbruk i konstruksjons- og implementasjonsfasen, gitt at vi skal implementere alle brukstilfeller unntatt brukstilfelle 8,9 og 10 (’Lage komponentmal’, ’Lage komponentregler’ og ’Lage katalogstruktur’).
Estimert tidsforbruk, design- og implementasjonsfase: 488,48 timer
Som i kapittel 8.1, Full versjon, blir det sannsynligvis også her et underestimat grunnet kompleksiteten til parseprosessen.

9 Figurliste

10 Vedlegg

10.1 Vedlegg 1 – Brukstilfelle Estimering (Full versjon)

Enter values in this column

Actors Weight Factors	Description	Weight	Enter NActors_i	Weighted Value	Comment
Simple_Actor	Ulvikler	1	1	1	
Average_Actor	UML Editor	2	1	2	
Complex_Actor		3	0	0	
Total Actor Weight				3	

Use Cases Weight Factors (Bases on the number of transactions in a use case)		Weight	Enter NUsecases_i	Weighted Value	Comment
Simple_Use_Case	3 or fewer transactions	5	6	30	
Average_Use_Case	4 to 7 transactions	10	4	40	
Complex_Use_Case	more than 7 transactions	15	0	0	
Transaction Based Factors				70	
Unadjusted Use Case Points				73	

Technical Weight Factors	Rating Scale is 0 to 5	TWeight_i	Enter TRating_i	Weighted Value	Reason
T1 Distributed System	0 =not important 5 =essential	2	0	0	
T2 Response or throughput performance objectives	0 =not important 5 =essential	1	1	1	
T3 End-user efficiency (online)	0 =not important 5 =essential	1	0	0	
T4 Complex internal processing	0 =not important 5 =essential	1	4	4	
T5 Code must be reusable	0 =not important 5 =essential	1	4	4	
T6 Easy to install	0 =not important 5 =essential	0,5	2	1	
T7 Easy to use	0 =not important 5 =essential	0,5	2	1	
T8 Portable	0 =not important 5 =essential	2	3	6	
T9 Easy to change	0 =not important 5 =essential	1	3	3	
T10 Concurrent	0 =not important 5 =essential	1	0	0	
T11 Includes special security features	0 =not important 5 =essential	1	0	0	
T12 Provides direct access for third parties	0 =not important 5 =essential	1	0	0	
T13 Special user training facilities are required	0 =not important 5 =essential	1	0	0	
TFactor				20	
Technical Factor (TCF)	.6 + (.01*TFactor)			0,8	

Environmental Factors for Team and Weights	Rating Scale is 0 to 5	EWeight_j	Enter ERating_j	Weighted Value	Reason	Experience Stability
F1 Familiar with the Rational Unified Process	0 = no experience, 3=average, 5=expert	1,5	5	7,5		0
F2 Application experience	0 = no experience, 3=average, 5=expert	0,5	1	0,5		1
F3 Object-Oriented Experience	0 = no experience, 3=average, 5=expert	-1	5	5		0
F4 Lead analyst capability	0 = no experience, 3=average, 5=expert	0,5	4	2		1
F5 Motivation	0=no motivation, 3=average, 5=high	1	5	5		0
F6 Stable requirements	0=extremly unstable, 5=unchanging	2	4	8		0
F7 Part-time workers	0=no part time , 5=all part time	-1	4	-4		0
F8 Difficult programming language	0=easy language, 3=average,5= difficult	-1	1	-1		0
EFactor				23		2
Environmental Factor (EF)	1.4+(-0.03*EFactor)			0,71		
Use Case Points	UUCP			41,464		
Person-hours per use case point	EperUUCP			20		
Estimated person-hours in project	Effort in person-hours			829,28		

Kundestyrt prosjekt

Høst 2001

Konstruksjon

Versjon 1.16

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
09.10.2001	Dokumentet opprettet	0.01	Ole Kristian
17.10.2001	Førsteutkast modultest, integrasjonstest etc.	0.02	Atle
19.10.2001	Startet på regler og maler	0.03	Magnus
23.10.2001	Førsteutkast kravoppfyllelse	0.06	Odd Christian
24.10.2001	Oppdatering av regler og maler	0.07	Magnus
24.10.2001	Overordnet systemstrukture	0.10	Ole Kristian
24.10.2001	Detaljert systembeskrivelse, Bevis av konsept	0.20	Martin
24.10.2001	Sekvensdiagrammer	0.21	Odd Christian
25.10.2001	Viktige valg, Første utkast på kontrollflyt	0.25	Martin
25.10.2001	Intern kommunikasjon	0.26	Odd Christian
25.10.2001	Stor forbedring av testkapittelet	0.30	Atle
25.10.2001	Innledning, oppdatert valg, figurer	0.35	Martin
25.10.2001	Noe oppdatering av regler og maler	0.36	Magnus
25.10.2001	Konsept lagt til	0.37	Ole Kristian
26.10.2001	Oppdateringer av ansvar ved modultest	0.38	Atle
30.10.2001	Oppdatert overordnet systembeskrivelse	0.45	Martin
01.11.2001	Oppdatert kravoppfyllelse og test	0.50	Atle
01.11.2001	Detaljert beskrivelse oppdatert	0.60	Martin
02.11.2001	Oppdatert mulige utvidelser	0.61	Magnus
02.11.2001	Regler & Maler, Vedlegg, DTD	0.70	Atle
02.11.2001	Intern kommunikasjon	0.80	Martin
04.11.2001	Figurtekst, første gjennomlesning	1.00	Martin
05.11.2001	Testlogg & Endringslogg	1.01	Atle
06.11.2001	La til vedlegg om kodeinspeksjon	1.02	Atle
07.11.2001	Endret litt i regler og maler	1.03	Magnus
07.11.2001	Fjernet kryss fra kravoppfyllelse	1.04	Atle
07.11.2001	Småkorrigeringer fra veilederne	1.05	Martin
09.11.2001	Endringer av regler og maler	1.07	Magnus
09.11.2001	Endringer av regler og maler	1.10	Odd Christian
10.11.2001	Retting av detaljfeil i konstruksjon	1.11	Martin
10.11.2001	Fikse ferdig regler	1.12	Odd Christian
13.11.2001	Korrekturlesing	1.13	Atle
13.11.2001	Topptekst/bunntekst	1.14	Ole Kristian
13.11.2001	Layout i regel og mal delen	1.16	Martin

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dette dokumentet er resultatet av konstruksjonsfasen i prosjektet og har dermed som målsetting å gjøre overgangen fra kravspesifikasjon til implementasjon enklest mulig.

1.2 Avgrensning

Dette dokumentet vil kun konstruere et system som oppfyller de krav som er gitt i Kravspesifikasjonen [KRA] for systemet valgt i forstudiet [FOR], og vil kun beskrive konstruksjonen ned til et rimelig detaljeringsnivå.

1.3 Spesielle definisjoner

Se glossar [GLO].

1.4 Dokumentreferanser

[FOR] Forstudiet, Qpro16 2001
[KRA] Kravspesifikasjonen, Qpro16 2001
[GLO] Glossar, Qpro16 2001
Alle eksterne dokumentreferanser er oppgitt i kapittel 13 – Litteraturliste

1.5 Dokumentoversikt

Dette dokumentet beskriver konstruksjonen av et system som skal oppfylle alle krav gitt i kravspesifikasjonen, og starter dermed med å definere konseptet i det systemet som skal konstrueres i kapittel 2. I kapittel 3 redegjøres det for viktige valg som ble tatt igjennom prosessen og hvordan de spiller inn på den resulterende konstruksjonen.

I kapittel 4 defineres konstruksjonen av et regel og mal formatet som gir den ønskede funksjonalitet. Kapittel 5 gir en overordnet system beskrivelse av et system som bygger på disse reglene og malene, mens kapittel 6 går i dybden på systemet og beskriver det i detalj. De dynamiske aspektene av systemet blir dekket gjennom kapittel 7 der kontrollflyten gjennom systemet beskrives.

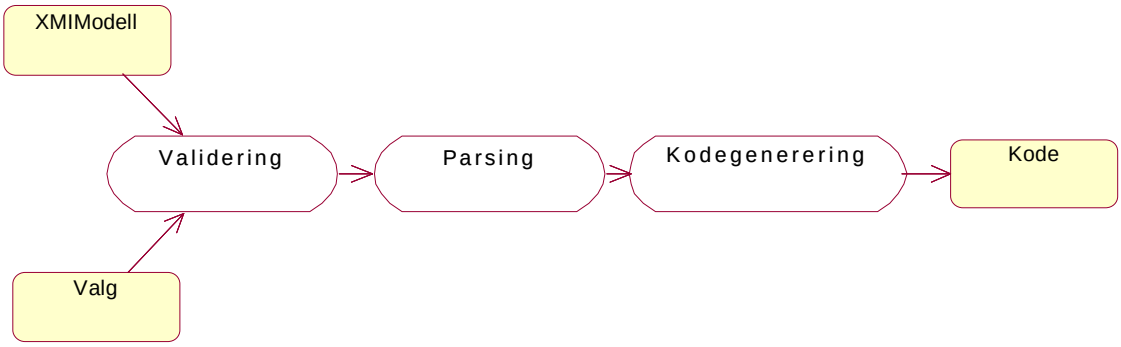
Kapittel 8 viser hvordan det konstruerte systemet oppfyller de ulike kravene, og hvilke deler av systemet som er ansvarlig for å oppfylle de. Kapittel 9 inneholder et implementert bevis av konseptet for systemet.

Kapittel 10 definerer modultester, integrasjonstest og regresjonstester for systemet samt andre risikoreduserende tiltak som skal brukes i implementasjonsfasen. Til slutt oppsummeres det hele i kapittel 11 med å peke ut mulige utvidelse av systemer.

2 Konsept

Systemet som skal konstrueres har som formål å forenkle oppstarten av systemutviklings-prosjekter. Det skal kunne ta inn generelle modeller og automatisk generere kildekode og SQL-skript ut i fra disse. I kravspesifikasjonen [KRA] kapittel 5.1.1, Krav til inndata, er det krav om at modellen skal være en UML-modell representert i XMI. For å oppfylle dette kravet og samtidig ha en enda generell løsning har vi valgt å la systemet støtte alle former for XML som modellrepresentasjon. Så lenge det eksisterer regler for den brukte XML representasjonen kan systemet generere kode fra den. I praksis vil derimot de fleste reglene kun være vinklet kun mot XMI formatet.

Modellrepresentasjonen må parses, og resultatet blir i neste omgang grunnlag for kodegenereringen. Det er også krav om at systemet skal kunne ta innparametere ved oppstart. Før parsingen av XMLen gjennomføres blir modellen og innparametrene validert for å kontrollere at det ikke er alvorlige mangler. At inndataene først blir validert, så parset, og til slutt brukt som grunnlag for kodegenereringen medfører en naturlig tredeling i oppbyggingen av systemet. En valideringsdel, en del som tar seg av generell parsing av XML-filer, og en annen del som genererer nødvendig kode.



Figur 2.1 – Overordnet konsept for systemet

For å generere kildekode ut fra den parsede XMI-modellen brukes et sett med regler og maler som sammen med innparametrene gitt av brukeren forteller hvordan kildekoden skal se ut. Ved oppstart må brukeren velge hvilket sett med regler og maler som skal brukes, og på den måten blir det bestemt hvilken arkitektur det skal genereres kode for. Ved at reglene og malene ligger i egne filer, og ikke er en integrert del av koden, kan brukeren forholdsvis enkelt lage nye regler og maler. Dette vil være nyttig hvis det er ønskelig å få generert kode for en arkitektur som det ennå ikke fins støtte for, eller hvis det er ønskelig å forandre innholdet i en allerede eksisterende arkitektur.

Reglene og malene må ha en så generell struktur at det kan lages regler for alle mulige arkitekturer. Dermed må også kodegeneratoren være generell nok til at alle gyldige regler og maler blir støttet.

3 Viktige valg tatt i konstruksjonsfasen

I dette kapittelet beskrives de valg som ligger til grunn for det systemet som konstrueres og hvilke konsekvenser de medfører.

3.1 Teknologiske valg

Her forklares de viktigste teknologiske valgene som ble tatt tidlig i fasen, og hadde innvirkning på den resulterende konstruksjonen.

3.1.1 Programmeringsspråk

Java 2 Platform, Standard Edition version 1.3.1 [JAV1] er valgt som programmeringsspråk. Dette var et enkelt valg fordi det medførte så mange fordeler:

- ?? Alle gruppens medlemmer har god kjennskap til Java. Tre års studium på NTNU har allerede dekket behovet for opplæring og kode kan derfor produseres umiddelbart.
- ?? Java er plattformuavhengig. Systemet vil enkelt kunne gjøres tilgjengelig for en stor brukermasse uavhengig av hvilket operativsystem de kjører på.
- ?? Gjenbruk av eksisterende programkode kan utnyttes. Det eksisterer mange ferdige pakker som kan benyttes for å løse ulike problemer.

Ulempene er i all hovedsak kompleksiteten av å sette seg inn i eksisterende pakker, samt at plattformuavhengigheten til Java er en sannhet med modifikasjoner.

3.1.2 Inndataformat

I all hovedsak er XMI [XMI1] generelt valgt som inndataformat. Valget her sto mellom XMI eller å definere en egen standard for inndata. Da en egendefinert standard ikke ville oppfylt kravene ble valget enkelt, XMI var eneste reelle alternativ.

XMI har til en viss grad blitt en ”defacto” standard for å representere UML modeller og støttes derfor av de fleste større verktøy. Konsekvensen av dette er at dersom systemet tar XMI som inndata vil man kunne i stor grad kunne bruke kjente UML editorer i samarbeid med kodegeneratoren.

Ulempene med å velge XMI som inndataformat er at de fleste UML editorer har sin egen dialekt av XMI standarden, i tillegg til at det etter hvert begynner å bli litt ulike versjoner av selve standarden. Dette medfører at XMI fra to ulike UML editorer kan se forholdsvis forskjellig ut.

Konsekvensene for systemet er at de viktigste ulike versjoner og dialekter av denne standarden må taes høyde for i regel- og malstrukturen for systemet på lengre sikt, men at man i første omgang kun tar utgangspunkt i XMI 1.0 som inndata.

3.1.3 Lagringsformat

Systemet har to ulike datamengder som skal lagres, regler og maler. Disse dataene er forholdsvis forskjellige i oppbygging, innhold, og bruksmåte. Alternativene her var mange, blant annet flate filer, XML og lagring i en database struktur.

XML [XML1] er valgt som lagringsformat for alle regelrelaterte data. Grunnen til at XML ble valgt var ønske om et regelformat som enkelt kunne endres og utvides av bruker, samt at formatet måtte være enkelt å benytte for kodegeneratoren. Ved å bruke XML oppnåes mye av fleksibiliteten som kreves. Ulike DTDer fastlegger formatet regler kan skrives på, og systemet kan bygges forholdsvis uavhengig av innholdet i reglene bare de er skrevet på rett format.

Flate filer med token er valgt som lagringsformat for malene i systemet. Dette fordi de i oppbygging er enkle og en mal inneholder store deler flat tekst. Bruken av tokens gir i tillegg en måte å relatere innholde malene opp mot ulike regler og gir dermed den frihet som systemet ønsker.

3.1.4 Utdataformat

Som utdata format vil systemet i utgangspunktet generere filer i ASCII kodesett i et UNIX filformat. Dette er standard utdata fra Java programmer.

3.1.5 Tredjeparts moduler

Til å parse XML dokumenter, herunder også XMI dokumenter, taes ferdige pakker fra com.sun og org.w3c i bruk [SUN1]. Disse modulene vil også dekke alle objekter knyttet til parseresultatet, og bruken av dette.

3.2 Ikke-teknologisk valg

Her forklares de viktigste ikke-teknologiske valgene som ble tatt tidlig i fasen og som hadde en implisitt innvirkning på den resulterende konstruksjonen.

3.2.1 Konstruksjonsmetodikk

Som konstruksjonsmetodikk valgte vi å bruke objekt orientert analyse og design (OOAD) basert på brukstilfelle, sekvensdiagram og klassediagram. Dette valget var naturlig da denne metodikken er velkjent av alle gruppens medlemmer og kan da taes i bruk umiddelbart.

En ulempe med dette er at det blir vanskelig å fremstille statiske aspekter av systemet i konstruksjonen. Dette vil gjøre det vanskeligere å modellere regler og malstrukturer med noen av modelltypene som her er brukt.

3.2.2 Konstruksjonsverktøy

Som konstruksjonsverktøy sto valget mellom TogetherSoft og Rational Rose. På grunn av mer tidligere erfaringer med Rational Rose ble dette verktøyet valgt som utviklingsverktøy i konstruksjonsfasen.

Konsekvensen av dette er at alle modeller og lignende i denne konstruksjonen er gjort ved hjelp av Rational Rose, og implementasjonsfasen blir betraktlig enklere å starte når klasseskjelettene blir automatisk generert på grunnlag av konstruksjonsmodellene.

I vedlegg 3 er det vedlagt en tegnforklaring til den notasjonen som Rose innfører i de ulike modellen.

3.2.3 Testmetoder

Alle testmetoder ble strengt bundet til den programvaren som gruppa har til rådighet. Dette innebærer at vi har brukt WebLogic til deploymentserver for bønner og MS SQL Server som databaseserver. Dette betyr imidlertid ikke at det ikke er tatt høyde for bruk av annen serverprogramvare. Endringer i SQL for å få generert databasen i en annen databasearkitektur endres i en egen mal for SQL.

3.2.4 Brukergrensesnitt

Uten at det var krav om det, avgjorde gruppen på et tidlig tidspunkt at det skulle konstrueres et system som hadde muligheten for å ha et grafisk brukergrensesnitt. Selve utformingen av brukergrensesnittet skulle gjøres så enkelt så mulig, slik at minst mulig tid gikk med på denne selvpålagte oppgaven. Brukergrensesnittet blir laget i Java Swing og skal kun ta inn nødvendige parametere enten valgt via dialogbokser eller skrevet inn i tekstfelt.

3.2.5 Navngiving

All metoder, klasser, attributter, pakker og andre momenter er valgt å modeller ved hjelp av engelsk språk. Dette gjelder også GUI og eventuelle feilmeldinger som måtte dukke opp her.

3.2.6 Fremgangsmåte og arbeidsdeling

Fremgangen i denne fasen var i stor grad parallell, men med mange felles sesjoner der arbeidet ble samkjørt. Det første som ble gjort var å etablere en felles oppgaveforståelse, og en veldig overordnet objektmodell som lå til grunne for det videre arbeidet. Deretter ble regel- og malformatet arbeidet frem i parallell med systemkonstruksjonen og bevis av konsept.

Konsekvensen av dette var at fasen gikk forholdsvis raskt, men ble litt uoversiktlig. Det ble også brukt litt arbeid med å samle trådene til slutt.

4 Maler og Regler

Kodegeneratoren skal være bygd opp på en slik måte at man ved å gi regler og maler sammen med XMI som input, skal få generert ferdig kode. I dette kapittelet vil disse reglene og malene blir beskrevet og målet er å gi en full oversikt over oppbyggingen av regelfilen.

Kort definisjon av hva som er ment med regler og maler:

Maler
Malene vil være skjelettet for den ferdige koden. Malen vil ligne mye på den ferdige koden. Forskjellen er at det vil være definert variabler der info fra XMIen må settes inn. Det kan være navn på klasse, metoder, attributter og lignende.

Regler
I reglene vil det i hovedsak bli spesifisert hva og hvordan relevant info i den parsede XMIen skal behandles før denne blir kombinert med malen. Reglene vil definere hvordan generatoren skal bruke malene og den parsede XMIen for å danne den ferdige koden.

4.1 Mulige vinklinger

Under konstruksjonen kom det opp to måter for hvordan reglene og malene kunne benyttes:

4.1.1 Vinkling 1 – fra XMI via reglene til malen

Det kan taes utgangspunkt i det parsede XMI-treet. Man må da først lese gjennom reglene for å finne ut hva som er interessant info i XMI-treet. Man traverserer så gjennom treet og plukker ut info. Denne infoen plasseres på rett plass i malene. Denne måten gjør validering av modellen vanskelig fordi ting kan mangle i treet uten at det blir oppdaget. Det kan selvsagt defineres i reglene hva som er krevd at skal være med i en modell, men dette fører til større kompleksitet.

4.1.2 Vinkling 2 – fra XMI via mal til regler

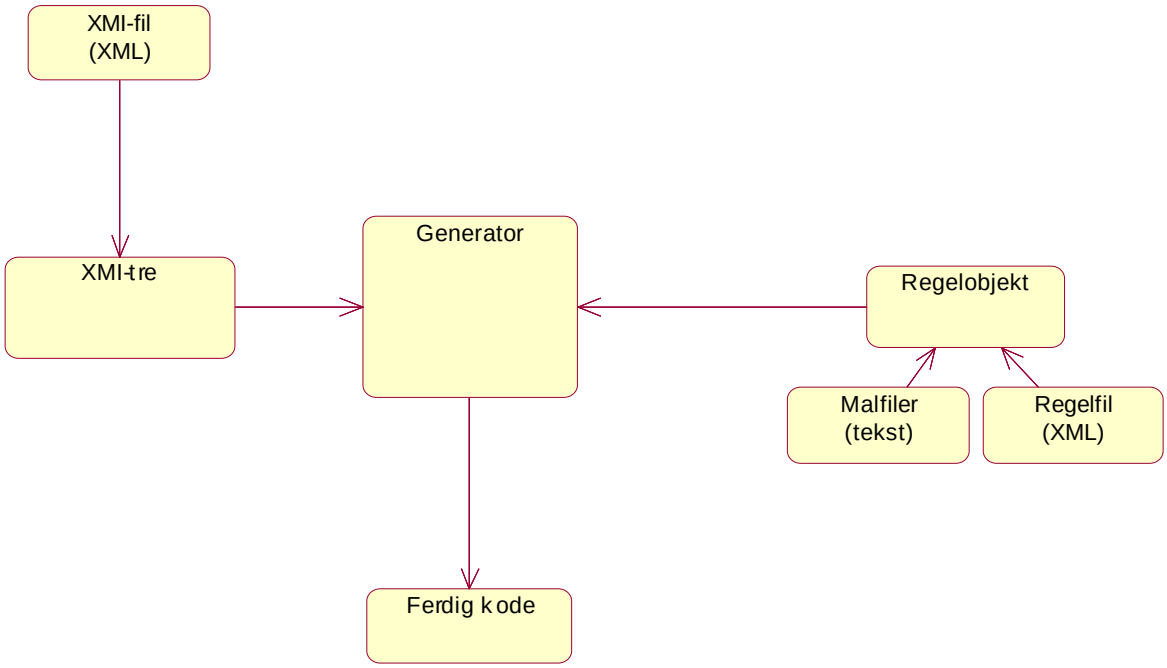
Man tar utgangspunkt i malene. Man vil da for hver entitetsnode i XMI-treet gå gjennom de malene som det er referert til i reglene for å fylle inn riktig informasjon. Ved å gå denne veien er man sikker på at alt i malene vil bli fylt ut, og eventuelt vil det komme en feilmelding hvis noen variabler i malene ikke finnes igjen i treet. Det er mulig at en entitet for eksempel ikke har attributter, og da vil det bli håndtert.

4.1.3 Valg

Vinkling 2 gir den minst komplekse og mest generelle løsningen. Tidsforbruket for kodegenereringen vil være omtrent like stort for begge løsningene. Derfor ble det besluttet å gå videre med vinkling 2, og denne vil bli beskrevet i detalj utover i kapittelet.

4.2 Overordnet virkemåte

Hovedprinsippet med kodegeneratoren som lages er at det skal være generell, det vil si uavhengig av hvilket språk eller arkitektur som benyttes. For å få til dette må alt språk- og arkitekturspesifikt legges i regler og maler. Reglene vil også bli avhengig av hva som finnes i XMIen som gies som input. Dette fordi kodegeneratoren ikke skal hardkodes til å lete seg fram i XMIen, men bare bruke reglene til å finne ut hvor relevant info er plassert i treet.



Figur 4.1 – Hvordan brukes reglene og malene

I figuren ovenfor ser man grovt hvordan flyten av data er i systemet. All data samles i generatoren som bruker reglene til å sette XMIen og malene sammen til ferdig kode. Generatoren tilbyr ferdigdefinerte metoder som det kan refereres til i reglene. Disse metodene er så generelle at man uten problem kan forandre på XMIen, reglene og malene uten å måtte gjøre noe med selve generatoren.

4.3 Beskrivelse av reglene og malene

Her vil det komme en beskrivelse av hvordan reglene og malene skal fungere. En oppskrift på hvordan man lager seg regler og maler vil komme i systembeskrivelsen.

4.3.1 Malfilene

Malene skal være vanlige flate tekstfiler som skal være mulig å søke seg gjennom. Det settes inn variable på formen `%VARIABELNAVN%` der man ønsker at det skal settes inn data fra XMI-treet. Tegnet som markerer variabelnavnet er valgfritt og det velges i regelfilen.

Et lite eksempel på mal:

```
import javax.ejb.*;

public abstract class %CLASSNAME% implements javaxejb.EntityBean
{
      /*
      Attributes declaration
      */
      %ATTRIBUTES%

      ...
}
```

Her ville programmet satt inn navnet på klassen i stedet for `%CLASSNAME%`. Navnet på klassen finnes i XMI-treet. I reglene er det definert nodereferanse slik at det blir mulig å gjøre oppslag etter klassenavnet, basert på variabelen i malen. `%ATTRIBUTES%` vil bli erstattet med info om alle attributtene entiteten skal inneholde. Et fullt eksempel finnes i vedlegg 1.

4.3.2 Regelfilen

Reglene kan være noe komplekse, og det kan være nyttig å ha eksempelet fra vedlegg 2 foran seg når man leser dette kapittelet.

Reglene lagres i en XML-fil som parses opp til et tre. Filen blir da grunnlaget for regelobjektet som kodegeneratoren benytter under kodegenereringen. I reglene blir det definert hvilke maler som skal brukes og hvilke XMI-subtre malene skal brukes på. Det mest sentrale som beskrives i reglene er hva som skal skje når programmet treffer på variabler i malene. Regelfilene inneholder også noen interne maler som blir benyttet i de tilfellene da malene må benyttes mer enn en gang inne i en fil, for eksempel til attributter. Det finnes blant annet et element i regelfilen som angir hvilket tegn som skal benyttes for å skille ut variabler i malene.

Regelfilen består av fem hovedelementer, som her blir forklart i denne rekkefølgen:

1. Hvilket tegn som skal omslutte variabler i malene. (`<VARIABLE_CHARACTER..>`)
2. Navnet på id-attributter i XMI-filene. (`<ID_ATTRIBUTE..>`)
3. Referanser til de lagrede malfilene. (`<TEMPLATES>..`)
4. Interne maler. (`<INTERNAL_TEMPLATES>..`)
   Brukes til repeterte elementer, for eksempel attributter og metoder.
5. Regler.(`<RULES>..`)

Disse fem hovedelementene omsluttes av en rotnode med navn `<RULEFILE>`, og alle de fem delene må være med i filen.

1. Hvilket tegn som skal omslutte variabler i malene.
Slik angir man hvilket tegn som skal omslutte variablene i malene. Linjen plasseres først av de fem hovedelementene.

```
<VARIABLE_CHARACTER char="%" />
```

Dette tegnet bør velges ut fra hvilke tegn som ikke benyttes i den arkitekturen regelfilen er laget for. Dette tegnet brukes så i malene for å omslutte variablene man vil at skal byttes ut med data fra XMI-treet. Det kan også kun være ett tegn, ikke en serie med tegn.

2. Navnet på id-attributter i XMI-filene
Navnet på en id-attributt angis på følgende måte:

```
<ID_ATTRIBUTE name="xmi.id" />
```

I XMI-filer har alle noder en unik id. Attributtet som inneholder denne iden angis regelfilen, slik at programmet kan finne fram i XMIen basert på id.

3. Referanser til lagrede malfiler
Denne delen omsluttes av en `<TEMPLATES>` tag. En referanse til en malfil vil se ut som følger:

```
<T1 trigger="Foundation.Core.Class" source="ejb.qpt"
      target="%CLASSNAME%EJB.java" />
```

Her er 'trigger' navnet på noden i XMI-treet som inneholder det undertreet som skal benyttes for å generere kode. 'source' angir hvilken mal som skal benyttes for genereringen. 'target' er navnet på den på filen hvor den genererte koden skal legges.

Det er denne referansen generatoren starter opp med for å finne riktig plass i XMI-treet for å starte selve kodegenereringen. Det kan være flere referanser til samme 'trigger', og

det vil si at flere maler skal benyttes på det samme undertreet. For eksempel ved generering av EJB vil det være fire referanser til samme noden i XMI-treet fordi det skal genereres fire forskjellige kodefiler ut i fra subtreet til den XMI-noden. I eksempelet over refereres det til XMI-noder av typen ”Foundation.Core.Class”. Det kan være flere noder i XMI-treet av denne typen, og da vil den samme genereringsprosessen kjøres for alle disse nodenes subtrær.

4. Interne maler.
Denne delen omsluttet av `<INTERNAL_TEMPLATES>`. Interne maler ser slik ut:

```
<ATTRIBUTE_TEMPLATE>%ATTRIBUTE_VISIBILTY% %ATTRIBUTE_TYPE%
      %ATTRIBUTE_NAME%;</ATTRIBUTE_TEMPLATE>
```

Interne maler brukes for å angi hvordan repeterte elementer skal formes. De er lagt til regelfilen, og ikke i egne malfiler fordi det kan hende at den samme interne malen skal benyttes i flere malfiler, som for eksempel i EJB der metodene skal genereres både i remote og implementasjonsklassen.

Det vil bli referert til de interne malene fra reglene, og de fungerer må samme måte som de malene som er lagret eksternt i egne filer.

Det tegnet som brukes foran og bak variabelnavnene må også her stemme overens med det som er angitt i punkt 1.

5. Regler
Denne delen omsluttet av `<RULES>`, en typisk regel vi se ut som følger.

```
<CLASSNAME type="replace" link=""
      source="Foundation.Core.ModelElement.name" mandatory=”true” />
```

Elementet, eller regelen, `<CLASSNAME .... />`refererer til triggeren `%CLASSNAME%` i malene. XML-attributtene angir hva som skal skje. En regel kan ha forskjellige typer attributter og forskjellige verdier på attributtene. Det er attributtene som bestemmer hva kodegeneratoren skal gjøre med regelen. Man kan selv bestemme hvilke attributter man vil benytte på de forskjellige malene, men noen attributter må benyttes for at genereringen skal fungere.

Her følger en oversikt over de forskjellige attributtene som kan brukes, og hvilke verdier de kan ha: (alle gyldige verdier er tekststrenger)

Attributt	**Gyldige verdier**	**Beskrivelse**
Type	replace	teksten i noden det refereres til i ”source” skal settes direkte inn i malen
	filereplace	For å støtte dynamiske filnavn på de ferdige kodefilene. Dvs. denne benyttes når en ferdig kode- eller skript- fil skal lagres med et navn som inneholder tekstdata fra XMI-treet.
	multi	Sier at det kan finnes mange elementer av denne typen, f.eks. metoder. Man må da først hente inn en intern mal som skal brukes på alle disse elementene. Denne typen regler vil alltid ha barnenoder av typen ”replace”.
	type1;...;typeN	En regel kan ha flere typer. Ikke alle kombinasjoner er mulige, men en god kombinasjon er ”replace;filereplace”.
source	”Navn på en node i XMI-treet”	Refererer til hvilken node i XMI-treet som inneholder data for å sette inn i malen. Se egen forklaring senere i dette kapittelet.
sourcerestriction	”Begrensning på hvilke noder som har gyldige verdier”	Brukes i forbindelse med type=”multi”. Det skal ikke genereres kode ut i fra noden angitt med source-attributtet, hvis ikke ”sourcerestriction” er tilfredsstilt. Se egen forklaring senere i kapittelet.
link	”Navn på node i XMI-treet som inneholder ønsket informasjon”	Kan benyttes hvis man må hoppe i XMI-treet for å finne riktig informasjon. Dette kan for eksempel være hvis man først må gå til en node der man finner IDen til den noden man virkelig er interessert i. Det er mulig å oppgi navnet til en node, samt navnet til et attributt i noden. (Se eksempel). Se egen forklaring senere i kapittelet.
template	”Navn på en inten mal”	Navnet skal matche navnet til et element under `<INTERNAL_TEMPLATES>`, hvor malen som skal brukes ligger.

Attributt	Gyldige verdier	Beskrivelse
separator	Angir et skilletegn	Separator brukes i forbindelse med type=”multi”. Skilletegnet legges mellom hver gang den interne malen kjøres gjennom. Separator trengs bl.a. når man skal sette inn metoder som kan ha flere inputvariabler. Inputvariablene må da skilles fra hverandre.
mandatory	true	Angir at noden som regelen referer til, må eksistere i XMIen.
	false	Angir at det ikke har noe å si om noden som regelen referer til, ikke eksisterer i XMIen. Hvis man velger å ikke angi ’mandatory’ i en regel blir den satt til ’false’ som standard.
map	”Referanse til en regel for bytting av verdier"	Format for attributtet: xmltag;subXmltag xmltag referer til den tagen som inneholder alle mapinger for den regelen map-attributtet står i. subXmltag er navnet på hver enkel mapping. En slik tag definerer bytting av verdier fra XMIen, før verdier skrives ut til fil. Se eksempel senere i kapittelet

Det henvises forøvrig til vedlegg 2 for eksempler på bruk av attributtene.

Forklaring og format for source-attributtet:
Dette attributtet er skal skrives på følgende format:

```
NodeMedVerdi;AttributtMedVerdi
```

Attributtet refererer til en node i XMIen med navnet `NodeMedVerdi`. Verdien som skal hentes ligger i denne noden. Dersom `AttributtMedVerdi`er gitt, skal verdien hentes ut fra attributtet med det navnet fra den gitte noden:

```
<NodeMedVerdi AttributtMedVerdi="verdi"></NodeMedVerdi>
```

Dersom ikke ligger verdien som ren tekst under noden:

```
<NodeMedVerdi>verdi</NodeMedVerdi>.
```

Navnet på noden må være unikt innenfor det aktuelle subtreet. Hvis en regel referer til en node med navn *ParentNode* skal *NodeMedVerdi* være et unikt navn i ParentNodes subtre.

Forklaring og format for link attributtet:
Dette attributtet er skal skrives på følgende format:

```
NodeMedLink;AttributtMedLink
```

Attributtet fungerer på nøyaktig samme måte som source, men verdien som hentes ut brukes på en annen måte. Verdien er en id-referanse til en node et helt annet sted XMIen.

Den noden som finnes etter at linken er brukt, er den noden som har sourcenoden i sitt subtre, og hvor dermed source attributtet til den opprinnelige noden gir et unikt navn.

Format for sourcerestriction:
Dette attributtet er skal skrives på følgende format:

```
NodeMedAttributt;AttributtMedVerdi;VerdiForAttributt
```

Attributtet fungerer på omtrent samme måte som source, og må sees i sammenheng med den. `NodeMedAttributt` er navnet på en node i subtreet til noden referert til av source-attributtet. Attributtet `AttributtMedVerdi` til denne noden sjekkes. Hvis verdien til attributtet ikke er likt `VerdiForAttributt` forkaster generatoren XMI-noden som er referert til i source.

Eksempel, map-attributtet: Konverteringsregler for datatyper.

Map-attributtet kan brukes på følgende måte:

```
<ATTRIBUTE_TYPE
      type="replace" link="Foundation.Core.Classifier;xmi.idref"
      source="Foundation.Core.ModelElement.name"map="TYPEMAPS;MAP">
</ATTRIBUTE_TYPE>
```

Når regelen `<ATTRIBUTE_TYPE>` brukes i kodegenereringen, skal programmet finne en verdi i XMI-filen, og legger denne verdien på en bestemt plass i malen. På grunn av map-attributtet skal noen av disse verdiene fra XMI-filen byttes ut med andre verdier. I dette eksempelet refereres det til en tag `<TYPEMAPS>`. Bytteregelene ligger som `<MAP> tagger`.

Generatoren sjekker alle verdier den finner i forbindelse med behandlingen av regelen `<ATTRIBUTE_TYPE>`, opp mot tagene under `<TYPEMAPS>`. Hvis verdien ligger ligger i en `old`-attributt i en `map`-tag, byttes verdien ut med verdien i `new`-attributtet.

I dette eksempelet skal attributt-typer byttes fra java-typer til sql-typer:

```
<TYPEMAPS>
      <MAP old="String" new="varchar(20)"/>
      <MAP old="int" new="number(8)"/>
</TYPEMAPS>
```

Det er viktig at hele denne bolken er plassert inne i `<RULES>....</RULES>`.

4.3.3 Database

Generatoren vil også støtte generering av databaseskript, for eksempel SQL, for opprettelse av nødvendig tabeller. Det er derfor nødvendig med egne regler og maler for dette. Virkemåten for disse vil bli akkurat som beskrevet over.

4.4 Detaljert virkemåte

For å beskrive hvordan reglene og malene blir benyttet i genereringsprosessen er det skissert opp pseudokode.

Generatoren starter i XMI-treet for å finne elementer det skal genereres kode ut fra. Det er allerede definert i reglene hvilke noder som er relevante. Her er metoden som kjører på treet:

```
generate(XMItre-rot rot)
     For alle barneNoder bnode til rot
          Hvis bnode er triggernode i regel-fil
             templateGenerate(bnode,templatefil)
          Ellers
             generate(bnode)
```

Når et interessant element blir funnet starter kodegenereringen. En mal blir lest inn, og data fra treet blir plassert i malen:

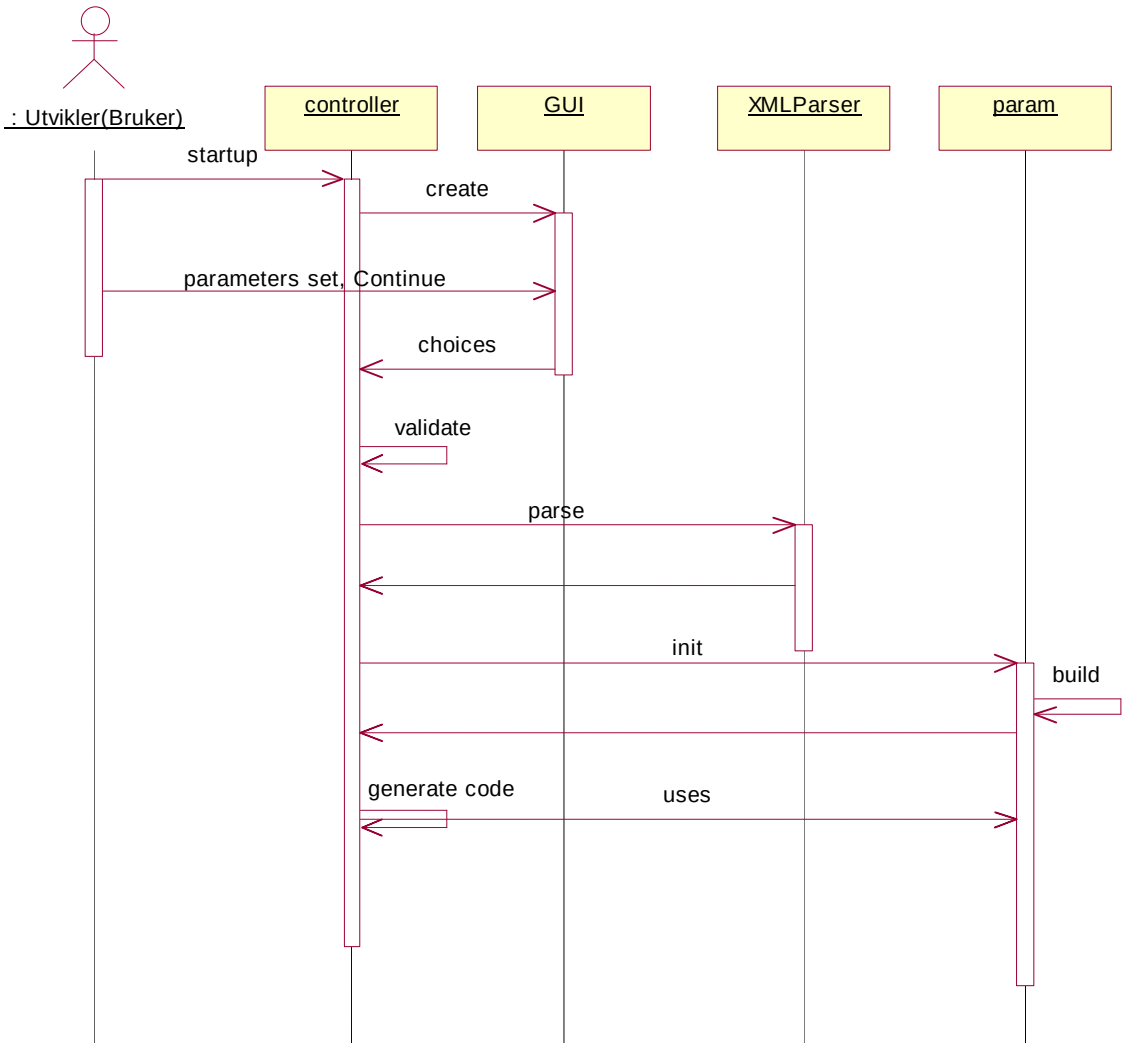
```
templateGenerate(subtre,templatefil)
     For alle Elementer tempEl (%...%) i templatefil
          Finn tilsvarende element e i rules-fil
          Finn node n i subtre hvor n.navn = e.sourceattributt
          Hvis regelen sier bytt
             Legg n.value på plassen til tempEl
          Hvis regelen peker på en mal
             templateGenerate(n,mal)
          Hvis regelen ikke finnes
             gå videre
```

5 Overordnet systembeskrivelse

I dette kapitelet beskrives et system som bruker de reglene og malen som ble spesifisert i forrige kapittel. Systemet vil bli beskrevet på et forholdsvis overordnet nivå og ved utstrakt bruk av diagram. De ulike diagramtypene og symbolene som brukes er forklart i vedlegg 3 – tegnforklaringer, de følger Rational Rose sin nyeste standard.

5.1 Overordnet scenario

Det overordnede scenario for systemet beskriver hovedflyten av hendelser som tar plass når en bruker ønsker å generere kode fra sin XMI modell. Det scenarioet som her skisseres antar at alle inndata brukeren gir er gyldige og at ingen feilsituasjoner oppstår i validere-, parse- eller kodebyggingsprosessen.



Figur 5.1 – Overordnet sekvensdiagram for systemet.

På dette høye abstraksjonsnivået er objektene ’controller’, ’GUI’, ’XMLParser’ og ’param’ tatt med. Hver av disse representerer essensielle deler av systemet, og vil utgjøre viktige moduler i den videre spesifiseringen av systemet.

Det første som skjer er at brukeren starter opp systemet. Dette er illustrert ved flyten ’startup’ fra ’Utvikler(Bruker)’ til ’controller’. Denne modulen oppretter så et brukergrensesnitt i modulen ’GUI’ som vises til brukeren. Dette er symbolisert ved hjelp av flyten ’create’.

Det at brukeren har gitt all nødvendig inndata og gitt signal om at prosessen skal forsette beskrives ved hjelp av flyten ’parameter set, continue’. Når dette skjer videreformidles inndataene til ’controller’ fra ’GUI’ i flyten ’choices’.

Inndataene fra brukeren valideres så i ’controller’ før ’XMLParser’ benyttes til å parse alle nødvendige XML dokumenter. Dette illustreres i flyten ’parse’. Trestrukturen som er resultatet av parsingen returnes så til ’controller’ fra ’XMLparser’.

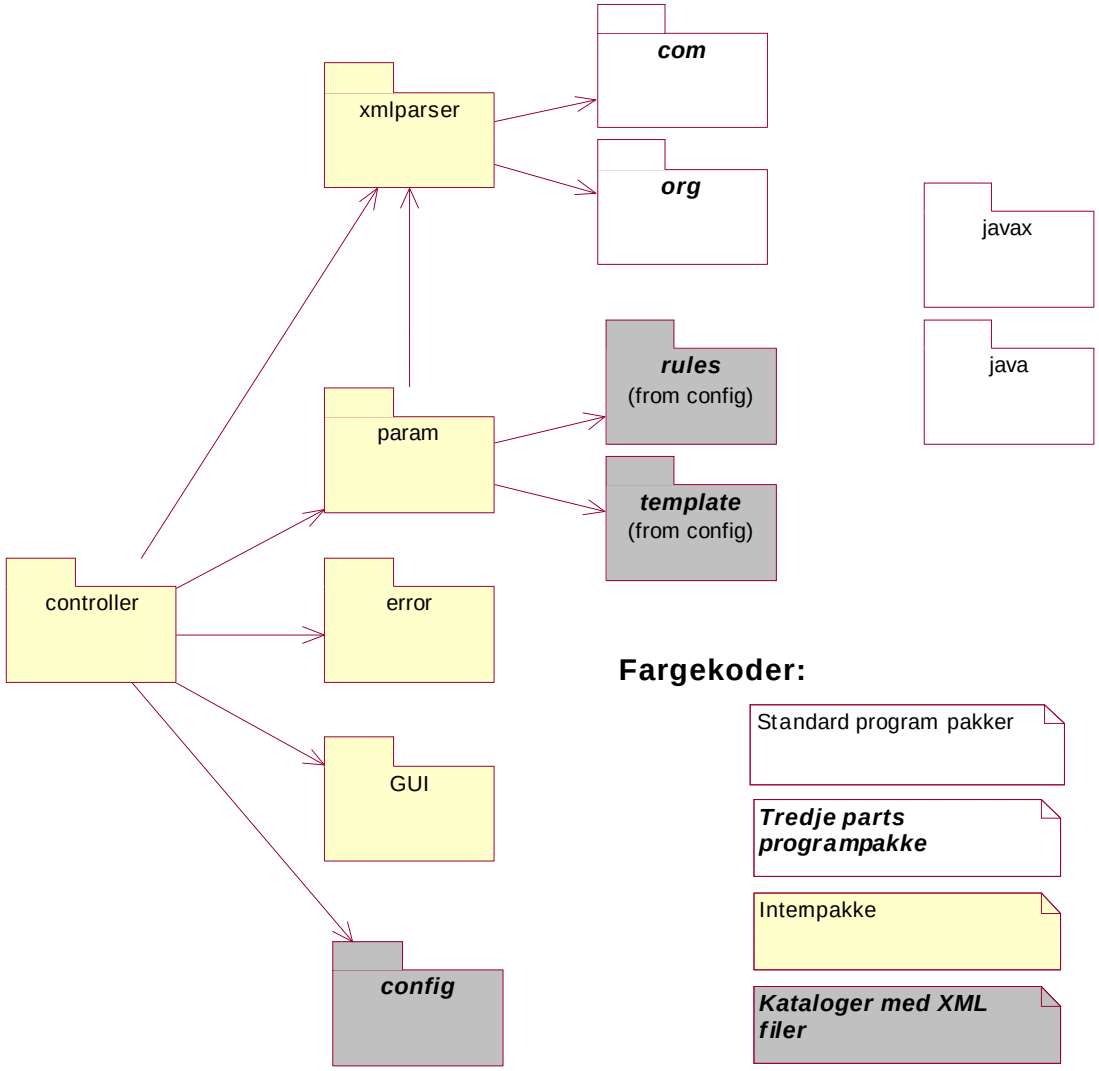
Flyten ’init’ beskriver opprettelsen av den interne representasjonen av de nødvendige dataene i modulen ’param’. Den interne byggingen av nødvendige regel, mal, og inndata objekter beskrivers ved flyten ’build’. Når oppbyggingen er fullført returneres en referanse til de ferdige objektene ’controller’ modulen.

Når så alt er kommet på objektform er det kun selve kodegenereringen som gjenstår. Denne gjøres i ’controller’ modulen ved hjelp av de oppbygde XMImodell-, regel- og mal-objektene i den tidligere konstruerte ’param’ modulen. Flyten ’uses’ illustrerer bruken av de ulike Param objektene i kodegenereringsprosessen. Selve genereringsprosessen foregår etter den pseudokode som er beskrevet i kapittel 4 – Regler & Maler.

5.2 Moduler

Som nevnt i forrige delkapittel er systemet delt opp i flere moduler. Her beskrives sammenhenger mellom de ulike modulene og ansvaret til hver enkelt modul.

I tillegg til de modulene som var aktører i det overordnede sekvensdiagrammet blir både kataloger, standard programpakker, tredjeparts programpakker og interne pakker viktige moduler i det konstruerte systemet.



Figur 5.2 – Klassediagram med systemet i moduler

'java' og 'javax' er standard programpakker som er en del av java-språket. Disse modulene er essensielle komponenter i alle javabaserte applikasjoner og beskrives derfor ikke nærmere i dette dokumentet. Se [JAV1] for mer informasjon om disse modulene.

Det er brukt deler av 2 tredjeparts moduler i systemet. Disse er 'org.w3c' og 'com.sun' som er laget av Sun [SUN1]. Bruken av disse modulene var et av de teknologiske valgene som ble gjort rede for i kapittel 3, og ligger til grunn for resten av konstruksjonen av systemet.

'config', 'rules' og 'templates' er kataloger som inneholder XML-filer og andre interne data som ikke er java-kode. Disse modulene utgjør de statiske og persistente aspektene i systemet.

’config’ inneholder en fil som definerer hvilke arkitekturer systemet kan generere kode for, samt navn på regelfilen for hver arkitektur. ’rules’ og ’templates’ er underkataloger til ’config’. ’rules’ modulen inneholder regelfilene i systemet og ’templates’ modulen inneholder malfilene i systemet.

Det er 5 interne pakker i systemet. En intern pakke kan sees på samling av konstruert kode, og til sammen utgjør disse 5 pakkene all koden som skal implementeres. De fem interne pakkene i systemet er ’controller’, ’xmlparser’, ’param’, ’error’ og ’GUI’.

’controller’ inneholder det meste av kontrollflyt i programmet, og bruker dermed alle de andre interne pakkene. Det er denne modulen som startes opp først og som initialiserer alle de andre på grunnlag av brukerens inndata og de innstillinger som finnes i ’config’ modulen.

’xmlparser’ inneholder det som trengs for å gjennomføre parsingen av XML-dokumenter. Dette gjelder både parsing av XMI som vil være grunnlaget for den videre kodegenereringen, og parsing av andre XML-filer som for eksempel regler. Det er i all hovedsak fra denne pakken de tredjeparts programpakkene blir brukt.

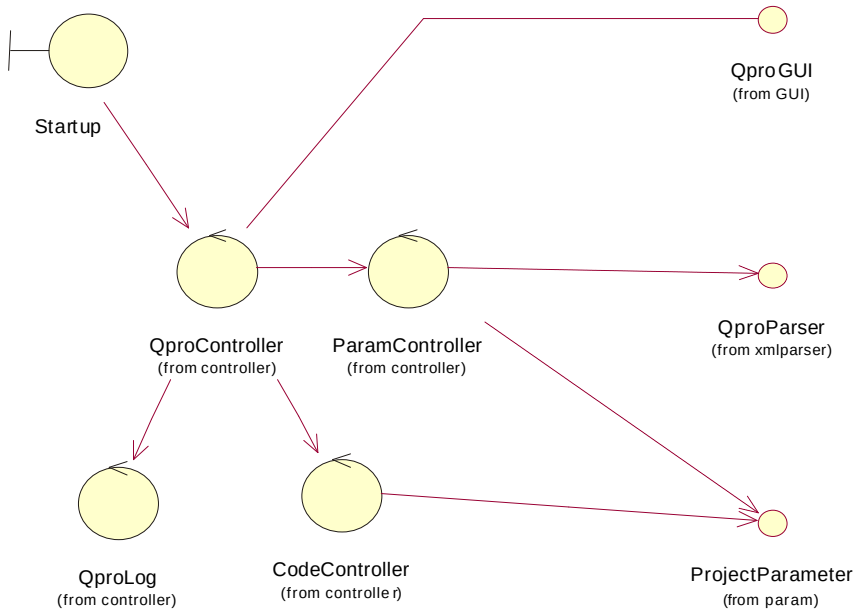
Modulen ’param’ inneholder entitetsklasser for alle parametere som systemet trenger. Dette blir bygget opp på grunnlag av innholdet i katalogene ’rules’ og ’templates’, samt informasjon som brukeren gir inn ved oppstart.

I pakken ’error’ ligger mekanismer for feilmeldinger og feilhåndtering i systemet. Dette blir realisert ved bruk av standard unntakshåndtering.

’GUI’ inneholder det grafiske brukergrensesnittet for systemet. Dette er skilt ut i en egen modul for å enklere kunne utvide det senere, samt å understreke uavhengigheten mellom brukergrensesnittet og det resterende systemet.

5.3 Sentrale objekter og grensesnitt

Som nevnt i forrige kapittel utgjør 5 interne pakker all koden som skal implementeres i systemet. Det er i disse pakkene at all bruk av det spesifiserte regel og mal formatet vil foregå. Her beskrives kort de mest sentrale objektene og grensesnittene i de interne pakkene, og sammenhengene mellom dem.



Figur 5.3 – Klassediagram med sentrale objekter og grensesnitt.

De mest sentrale objektene finnes i ’controller’ pakken da det er denne pakken som står for det meste av kontrollflyten i systemet. Pakkene ’GUI’, ’XMLparser’ og ’param’ har alle et sentralt grensesnitt hver som definer de tjenester som de tilbyr.

Grenseklassen ‘Startup’ er en enkel oppstartsklasse for systemet.

’QproController’ er den sentrale kontrollklassen i systemet som står for den interne kommunikasjonen i systemet.

’QproGUI’ er et grensesnittet som definerer hvilke tjenester det grafiske brukergrensesnittet skal tilby til systemet.

Kontrollklassen ’QproLog’ tar seg av logging mens systemet kjører, og skriver dette til en loggfil.

’ParamController’ er en kontrollklasse som validerer inndata og oppretter den interne representasjonen av disse.

Grensesnittet ’ProjectParameter’ definerer alle tjenestene den interne representasjonen av parametrene i systemet må tilby.

’QproParser’ et grensesnittet som definerer tilbudet som pakken ’XMLParser’ skal tilby.

Kontrollklassen ’CodeController’ er ansvarlig for å bygge kode ut i fra den parsede XMIen og de andre parametrene.

6 Detaljert systembeskrivelse

I dette kapitelet detaljeres beskrivelsen av systemet ytterligere. Kapittelet tar for seg hver av de ulike modulene fra den overordnede systembeskrivelsen og detaljerer konstruksjonen ned til metode og attributt nivå. Til slutt samles alle tråder i en modell som beskriver hele systemet på klassenivå.

6.1 Interne pakker

Her beskrives alle de modulene av typen interne pakker med tilhørende klasser i klassediagrammer som viser sammenhenger mellom klassene, og de ulike klassenes metoder og attributter. Beskrivelsen av hver pakke starter med en standard beskrivelse av pakken etterfølgt av et klassediagram og standard beskrivelse av hver klasse i pakken. Denne oversikten beskriver generelt kun offentlige metoder og attributter, og oppgaven til mange av de metodene som her er beskrevet bør deles mellom flere private metoder internt i klassen. Selve feilhåndteringen er heller ikke tatt med i denne beskrivelsen. Alle relasjoner til eksterne klasser skrives som ’pakkenavn.klassenavn’

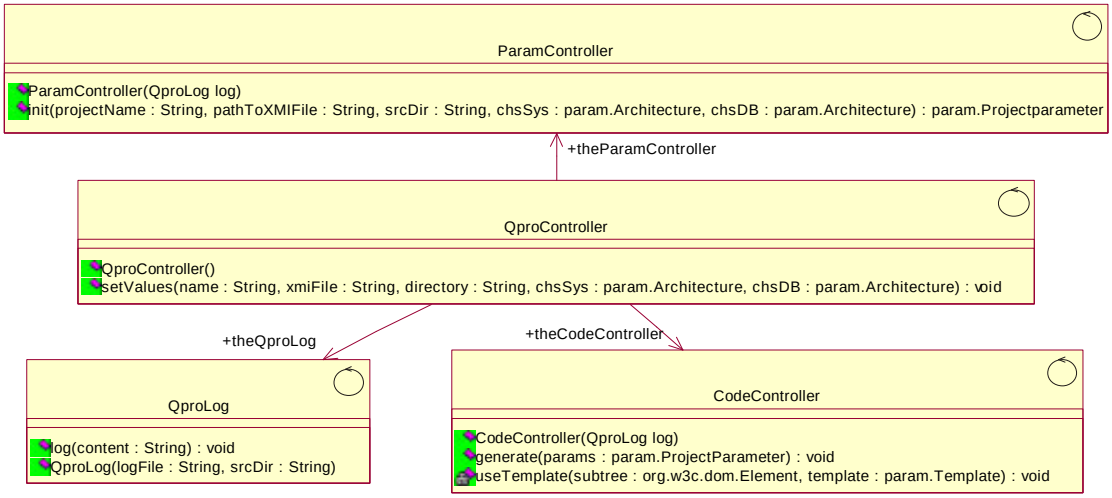
6.1.1 Pakken ’controller’

Beskrivelse: Den sentrale pakken i systemet. Inneholder det meste av kontrollflyten i systemet, herunder hovedlinjene i validering, logging og generering.

Avhengigheter: Bruker alle de andre interne pakkene, samt ’config’ katalogen.

Klasser:

- ?? QproController
- ?? ParamController
- ?? QproLog
- ?? CodeController



Figur 6.1 – Klassediagram for ‘controller’ - pakken

Klassen 'QproController'

Beskrivelse: Ansvarlig for all intern kommunikasjon, og initialiserer brukergrensesnitt og loggingen til systemet. Denne klassen tar seg av den endelig feilhåndteringen i systemet.

Relasjoner: Navigerbar relasjon til 'ParamController', 'QproLog', 'CodeController' og 'GUI.QproGUI'. Bruker også innholdet i modulen 'config' til å initialisere systemet.

Konstruktor: ?? QproController()
Initialiserer systemet og starter opp brukergrensesnittet.

Metoder: ?? setValues(name,directory,chsSys,chsDB)
Denne metoden blir kallet når brukeren har satt alle verdiene i brukergrensesnittet og starter opp 'QproLog', 'ParamController' og til slutt 'CodeBuilder'.

Klassen 'ParamController'

Beskrivelse: Denne klassen ansvarlig for validering og oppretting av den interne parameter

Relasjoner: Bruker 'XmlParser.QproParser' og oppretter 'param.ProjectParameter'. Kan kaste 'error.ParamException'.

Konstruktor: ?? ParamController(log)
Initialiserer klassen og lagrer referansen til logobjektet

Metoder: ?? init(projectname,pathToXMIFile, srcDir, chsSys, chsDB)
Tar inn alle parameterene for validering og opprettelse av de interne objektene. Denne metoden sørger også for å validere XMI modellen som brukeren gav som inndata.

Klassen 'CodeController'

Beskrivelse: Denne klassen har ansvaret for kodegenereingen i systemet. Den skal på grunnlag av det ferdigparsede XMI dokumentet bruke den interne representasjonen av regler og maler for valgte arkitekturer.

Relasjoner: Bruker 'param.ProjectParam' og oppretter kildekode som tekstfiler. Kan kaste 'error.BuildException'

Konstruktor: ?? CodeController(log)
Initialiserer klassen og lagrer referansen til logobjektet

Metoder: ?? generate(params)
Tar inn den interne represenatsjone av alle parameterene og starter kodegenerering på denne.

?? useTemplate(element, template)
Tar i bruk en mal på et del tre av modellen.

Disse to metodene i CodeController utgjør de metodene som er fremstilt i pseudo kode under beskrivelsen av den detaljerte virkemåten i kapitell 4 – Maler og Regler.

Klassen ’QproLog’

Beskrivelse: Ansvarlig for loggingen til fil av parseprosessen.

Relasjoner: Kan kaste ’error.LogException’

Konstruktor: ?? QproLog(logfil,srcDir)
Initialiserer logging og oppretter logfilen.

Metoder: ?? log (content)
Skriver innparameteret til loggfilen.

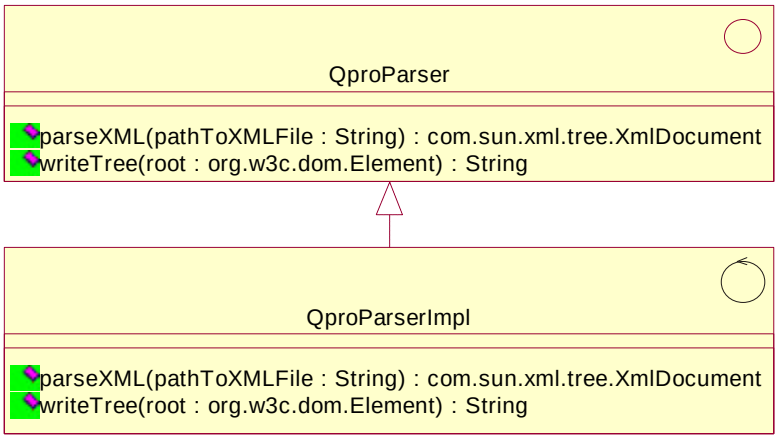
6.1.2 Pakken ’xmlparser’

Beskrivelse: Inneholder all parsing av XML dokumentet.

Avhengigheter: Bruker i stor grad de to tredjeparts pakkene ’org’ og ’com’.

Grensesnitt: ?? QproParser

Klasser: ?? QproParserImpl



Figur 6.2 – Klassediagram for ‘xmlparser’ pakken

Grensesnittet 'QproParser'

Beskrivelse: Definerer alle tjenestene som pakken 'XMLParser' skal tilby.

Relasjoner: Kan kaste 'error.ParseException'

Metoder:
?? parseXML(pathToXMLFile)
Parser det spesifiserte dokumentet og returnerer en objectrepresentasjon av parseresultat
?? writeTree(rotelement)
Returnerer en tekstlig representasjon av et XML-tre.

Klassen 'QproParserImpl'

Beskrivelse: En implementasjon av 'QproParser' grensesnittet og realiserer alle metodene som er spesifisert der ved hjelp av tredjepartsmodulene 'org' og 'com'.

Relasjoner: Implementerer 'QproParser'

Konstruktor:
?? QproParserImpl()
Standard konstruktor.

Metoder:
?? Alle metodene spesifisert i QproParser.

6.1.3 Pakken 'param'

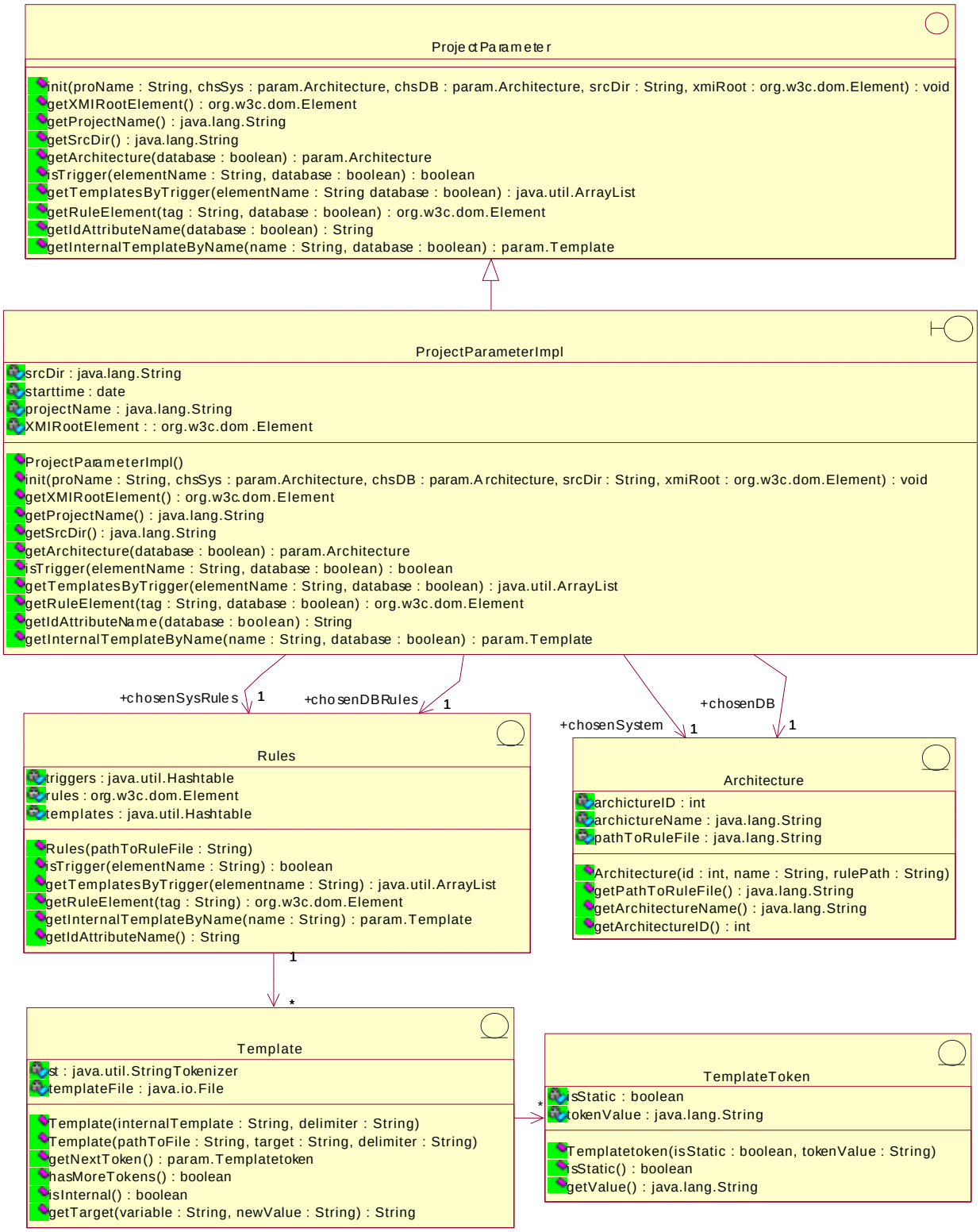
Beskrivelse: Inneholder alle interne representasjoner av parameterne som entitetsklasser. Hele pakken blir med alt dets innhold blir opprettet på grunnlag av brukerens valg.

Avhengigheter: I stor grad avhengig av XML-modulene 'rules' og 'templates'. Bruker i tillegg 'xmlParser.QproParser' i opprettelsen av objektene. Inneholder et regel- og malobjekt for valgt databasestruktur og et regel- og malobjekt for valgt systemarkitektur.

Grensesnitt:
?? ProjectParameter

Klasser:
?? ProjectParameterImpl
?? Architecture
?? Rules
?? Template
?? TemplateToken

NB! I denne pakken er noen relevante private attributter tatt med for å vise hvordan grensesnittet er tenkt oppfylt i de ulike klassene.



Figur 6.3 – Klassediagram for 'param' pakken

Grensesnittet 'ProjectParameter'

Beskrivelse: Definerer de sentrale tjenestene som pakken 'param' skal tilby. For alle metodene som gjelder arkitektur eller regler må det spesifisere om det er database- eller systemarkitekturen som ønskes ved den boolske verdien 'database'.

Relasjoner: -

Metoder:
- ?? init(proName,chsSys,chsDB,srcDir,xmiRoot)
  Definerer hvilke inndata som skal til for å opprette et ProjectParameter, og står for selve oppbyggingen av objektet.
- ?? getXMIRootElement()
  Returnerer rotelementet til XMI tredet.
- ?? getProjectName()
  Returnerer navnet til prosjektet.
- ?? getSrcDir()
  Returnerer den valgte utkatalogen
- ?? getArchitecture(database)
  Returnerer den arkitekturen som ligger til grunn for dette objektet, enten database arkitektur eller systemarkitektur.
- ?? isTrigger(elementName,database)
  forteller om en tag i XMI-modellen hører til en eller flere maler ("templates") som den da utløser (trigger).
- ?? getTemplatesByTrigger(elementName,database)
  Returnerer en liste med alle maler en trigger utløste.
- ?? getRuleElement (tag,database)
  Returnerer rotnoden til regelen som matcher den gitt tagen.
- ?? getInternalTemplateByName(name,database)
  Returner den intern malen som matcher navnet
- ?? getIdAttributeName(database)
  Returnerer navnet på den attributtet der ID som skal brukes ligger

Klassen 'ProjectParameterImpl'

Beskrivelse: En implementasjon av ProjectParser' grensesnittet som realiserer alle metodene som er spesifisert der ved hjelp av hjelpeklassene Rules og Architecture. Har ingen kompliserte metoder i seg selv.

Relasjoner: Implementerer 'ProjectParser', har to navigerbare relasjoner til Rules og Architecture en for database og en for systemarkitektur.

Konstruktor:
- ?? ProjectParameterImpl()
  Standard konstruktor.

Metoder:
- ?? Alle metodene spesifisert i 'ProjectParameter'.

Attributter
- ?? srcDir – inneholder utkatalogen til kildekoden
- ?? projectName – inneholder navnet på prosjektet
- ?? xmiRootElement – inneholder rotelement på XMI modellen

Klassen ’Architecture’

Beskrivelse: En objektrepresentasjonen av en arkitektur med id, navn, og path til tilhørende regelfil

Relasjoner: -

Konstruktor:
?? Architecture(id,name,rulePath)
Standard konstruktor som larger verdiene.

Metoder:
?? getPathToRuleFile()
?? getArchitectureName()
?? getArchitectureID()

Atributter
?? architectureID- identifikatoren til arkitekturen
?? architectureName – navnet på arktitekturen
?? pathToRuleFile – pathen til den relaterte regelfilen

Klassen ’Rules’

Beskrivelse: En objektrepresentasjon av en regel fil på det format som er beskrevet i kapittel 4 – Maler og Regler

Relasjoner: Har en navigerbar relasjon til mange ’Template’, samt bruker ’xmlParser.QproParser’ i sin interne oppbygning. Er fullstendig av hengig av regelfilen i ’rules’ som objektet skal bygges over.

Konstruktor:
?? Rules(pathToRuleFile)
Bygger opp det komplekse ’Rules’ objektet på grunnlag av innholdet i filen.

Metoder:
?? isTrigger(elementName)
forteller om en tag i XMI modellen hører til en eller flere maler som den da utløser trigger.
?? getTemplatesByTrigger(elementName)
Returnerer en liste med alle maler en trigger utløste.
?? getRuleElement (tag)
Returnerer rotnoden til regelen som matcher den gitt tagen.
?? getInternalTemplateByName(name)
Returner den intern malen som matcher navnet
?? getIdAttribute()
Returnerer navnet på det attributtet som eventuelle linker ligger under

Attributter
?? triggers – en oppslagstruktur for triggere koblet til maler
?? rules – regeldelen av regelfilen i en trestruktur
?? templates – en oppslagsstruktur for interne maler

Realiserer metodene ’isTrigger’,’getTemplatesByName’,’getRuleElement’, ’getIdAttribute’ og ’getInternalTemplateByName’ fra det overordnede grensesnittet ’ProjectParameter’ etter at ’ProjectParmeterImpl’ har videreformidlet kallet til rette ’Rules’-objekt.

Klassen 'Template'

Beskrivelse: En objektrepresentasjonen av en malfil i systemet eller av interne maler fra regelfilene

Relasjoner: Oppretter flere 'TemplateToken' instanser. Er fullstendig av hengig av mal filen i 'templates' som objektet skal bygges over.

Konstruktor:
?? Template(internalTemplate, delimitator)
Konstruktor som oppretter en objektrepresentasjon av en intern maler.
?? Template(pathToFile, target, delimitator)
Konstruktor som oppretter en objektreresentasjon av en malfil.

Metoder:
?? getNextToken()
Returnerer den neste biten av malen som en 'TemplateToken'.
?? getTarget(variable, newValue)
Returnerer navnet til målfilen for denne templaten, etter at den variable biten av navnet er satt.
.

Attributter
?? st – Et standardverktøy fra Java som brukes for å parse malene
?? target – målfilen for en template.

Klassen 'TemplateToken'

Beskrivelse: En objektrepresentasjonen av en del av mal

Relasjoner: -

Konstruktor:
?? TemplateToken(isStatic,tokenValue)
Konstruktor som lagrer de gitte verdiene

Metoder:
?? getValue ()
Returnere verdien til 'TemplateToken'
?? isStatic()
Viser om dette er en statisk tekst eller om det er et dynamisk token som det må brukes regler på..

Attributter
?? st – Et standardverktøy fra Java som brukes for å parse malene
?? target – målfilen for en template.

6.1.4 Pakken 'GUI'

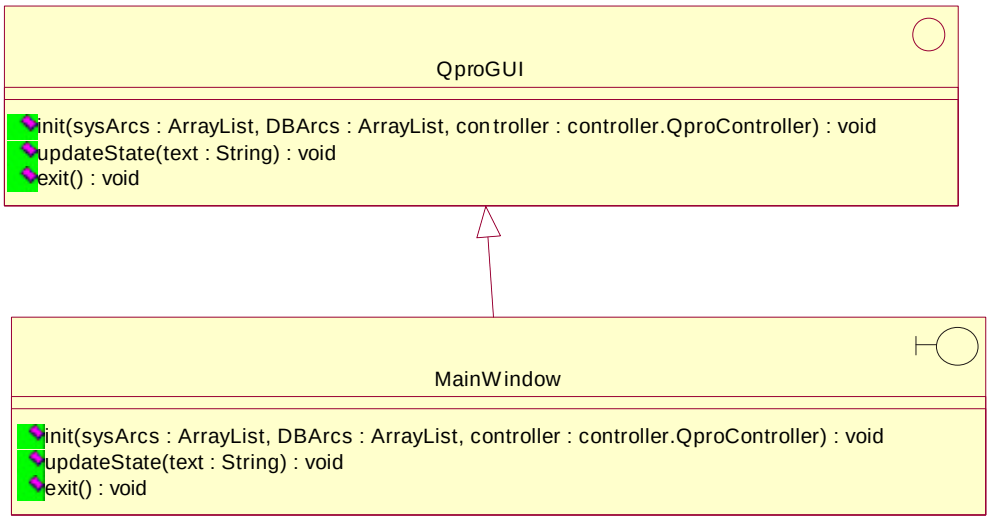
Beskrivelse: Inneholder og innkapsler alt som finnes av brukergrensesnitt i systemet.

Avhengigheter: -

Grensesnitt: ?? QproGUI

Klasser: ?? MainWindow

Denne pakken er kun konstruert ut ifra hvilke metoder systemet behøver et brukergrensesnitt, det er ikke tatt hensyn til implementasjon av selve brukergrensesnittet.



Figur 6.4 – Klassediagram for 'GUI' pakken

Grensesnittet 'QproGUI'

Beskrivelse: Definerer alle tjenestene som pakken 'GUI' skal tilby systemet.

Relasjoner: Har en navigerbar relasjon til 'QproController' for å kunne sende informasjonen fra brukergrensesnittet inn til systemet.

Metoder:
?? init(sysArcs,DBArcs,controller)
Oppretter GUI'en og viser denne til brukeren.
?? updateState(text)
Metode for å vise systemstatus til bruker.
?? exit()
Metode for avslutte og lukke brukergrensesnittet.

Klassen 'MainWindow'

Beskrivelse: Symboliserer en mulig realisering av grensesnittet 'QproGUI'

Relasjoner: -

Konstruktor: ?? MainWindow()
Standar konstruktor

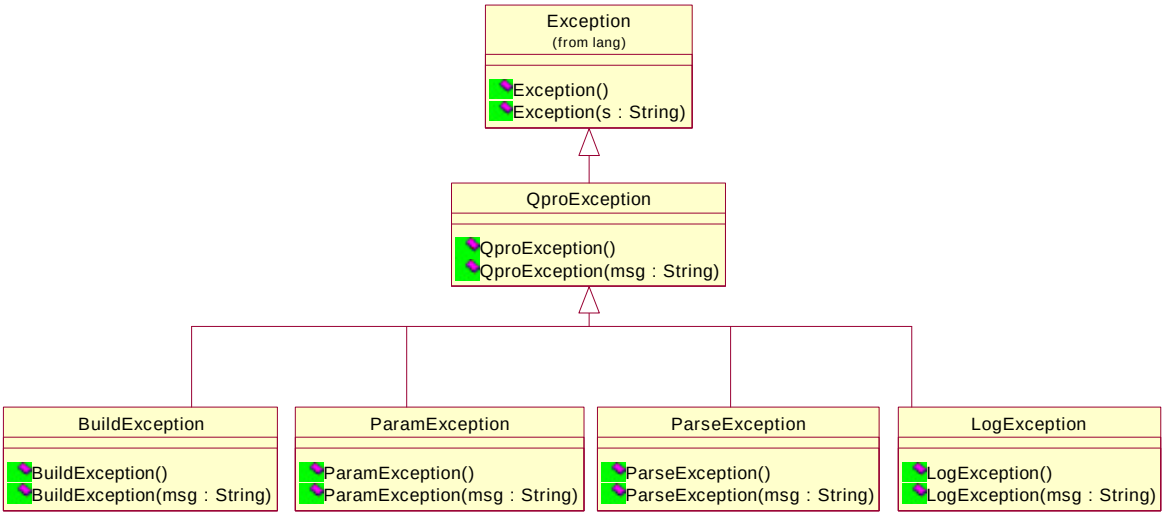
Metoder: ?? Alle metodene spesifisert i 'QproGUI'

6.1.5 Pakken 'error'

Beskrivelse: Inneholder mekanismer for feilmeldinger og feilhåndtering i systemet. Bygger i stor grad på den ferdige feilhåndteringen som allerede eksisterer i Java med 'exceptions'.

Avhengigheter: -

Klasser:
?? QproException
?? BuildException
?? ParamException
?? ParseException
?? LogException



Figur 6.5 – Klassediagram for 'error' pakken

Klassen ’QproException’

Beskrivelse: Er en generell feilhåndteringsklasse for systemet. Denne blir ikke kastet selv, men er superklasse for de unntakene som blir kastet. Dette er organisert slik med tanke på eventuelle utvidelser og generell feilhåndtering.

Relasjoner: Arver direkte fra ’java.lang.Exception’

Konstruktor:
?? QproException()
Bruker kun super – konstruktor
?? QproException(msg)
Bruker kun super – konstruktor

Metoder: ?? Alle metodene arvet fra Exception

Klassen ’BuildException’

Beskrivelse: Representerer feil som oppstår under kodebyggingen.

Relasjoner: Arver direkte fra ’QproExcption’

Konstruktor:
?? BuildException()
Bruker kun super – konstruktor
?? BuildException(msg)
Bruker kun super – konstruktor

Metoder: ?? Alle metodene arvet fra QproException

Klassen ’ParamException’

Beskrivelse: Representerer feil som oppstår under opprettelse av parameterobjektene og under validering av inndata.

Relasjoner: Arver direkte fra ’QproExcption’

Konstruktor:
?? ParamException()
Bruker kun super – konstruktor
?? ParamException(msg)
Bruker kun super – konstruktor

Metoder: ?? Alle metodene arvet fra QproException

Klassen 'ParseException'

Beskrivelse: Representerer feil som oppstår under parsingen av XML-filer.

Relasjoner: Arver direkte fra 'QproExcption'

Konstruktor: ?? ParseException()
Bruker kun super – konstruktor
?? ParseException(msg)
Bruker kun super – konstruktor

Metoder: ?? Alle metodene arvet fra QproException

Klassen 'LogException'

Beskrivelse: Representerer feil under loggingen i systemet.

Relasjoner: Arver direkte fra 'QproExcption'

Konstruktor: ?? LogException()
Bruker kun super – konstruktor
?? LogException(msg)
Bruker kun super – konstruktor

Metoder: ?? Alle metodene arvet fra QproException

6.2 Kataloger med XML filer

I dette kapittelet beskrives de modulene som var kataloger med XML-filer nærmere, og det beskrives hvilke typer XML-filer de ulike katalogene inneholder. Disse katalogene med innhold utgjør de statiske og persistente aspektene i systemet.

6.2.1 Katalogen 'config'

Katalogen 'config' inneholder en fil som definerer hvilke arkitekturer systemet kan generere kode for, samt navn på regelfilen for hver arkitektur. Denne filen er skrevet i XML format og har en egen DTD som bestemmer hvilket format som er lov å bruke i denne filen.

DTD til denne filen er definert som:

```
<!ELEMENT architectures (architecture+)>
<!ELEMENT sysarchitecture (id,name,rulepath)>
<!ELEMENT dbarchitecture (id,name,rulepath)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT rulepath (#PCDATA)>
```

Et eksempel på en slik fil kan være:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE architectures SYSTEM "archs.dtd">

<architectures>
      <sysarchitecture>
            <id>1</id>
            <name>ejb</name>
            <rulepath>config/rules/ejbRules2.xml</rulepath>
      </sysarchitecture>
</architectures>
```

Her vil hver arkitektur som systemet kan generere kode for være representert som en ’architecture’ entitet som forteller om identifikator, navn og banen til regelfilen. I tillegg bestemmes typen til arkitekturen.

Katalogene ’rules’ og ’templates’ er underkataloger til ’config’.

6.2.2 Katalogen ’rules’

Katalogen ’rules’ inneholder regelfilene i systemet. Det finnes en regelfil per systemarkitektur systemet støtter, og en regelfil per databasearkitektur systemet støtter. Regelfilene ligger på det format som er beskrevet i kapittel 4, og et eksempel på en slik regelfil kan sees i vedlegg 2 – Eksempel på regelfil.

6.2.3 Katalogen ’templates’

Katalogen ’templates’ inneholder malfilene i systemet. Det kan finnes flere maler per arkitektur og hver en av disse er lenket til fra regelfilen. Malfilene ligger på det format som er beskrevet i kapittel 4, og et eksempel på en slik fil kan sees i vedlegg 1 - Eksempel på mal.

6.3 Tredjeparts moduler

I dette kapittelet skriver vi litt om tredjeparts pakker som vil bli benyttet i prosjektet. Disse pakkene skal godkjennes av kunden.

6.3.1 Modulen ’com’

Dette prosjektet bruker ikke alle programpakkene som finnes under navnet ’com’ men kun en submengde. Se [SUN1] for mer informasjon om de pakkene som blir brukt. De pakkene fra com som brukes er:

com.sun.xml.parser

Inneholder to raske XML prosessorer som er definert i XML 1.0 spesifikasjonen, disse to er parsere en validerende og en ikke-validerende med noen støtteklasser og grensesnitt.

com.sun.xml.tree

Denne pakken støtter i-minne XML dokumenter i form av et parse tre samsvarende med W3C DOM Level 1 Core Recommendation [XML2], som med utvidelser støtter XML navnerom som definert av den nåværende XML standarden.

com.sun.xml.util

Denne pakken inneholder verktøyklasser som er til hjelp når man bygger XML relaterte programvare rammeverk.

6.3.2 Modulen 'org'

Dette prosjektet bruker ikke alle programpakkene som finnes under navnet 'org' men kun en submengde. Se [SUN1] for mer informasjon om de pakkene som blir brukt.
De pakkene fra org som brukes er:

org.w3c.dom

DOM [DOM1] er et anbefaling fra World Wide Web Consortium (W3C) [WWW1] , som definerer programmeringsgrensesnitt mot XML (og HTML) dokument.

org.xml.sax

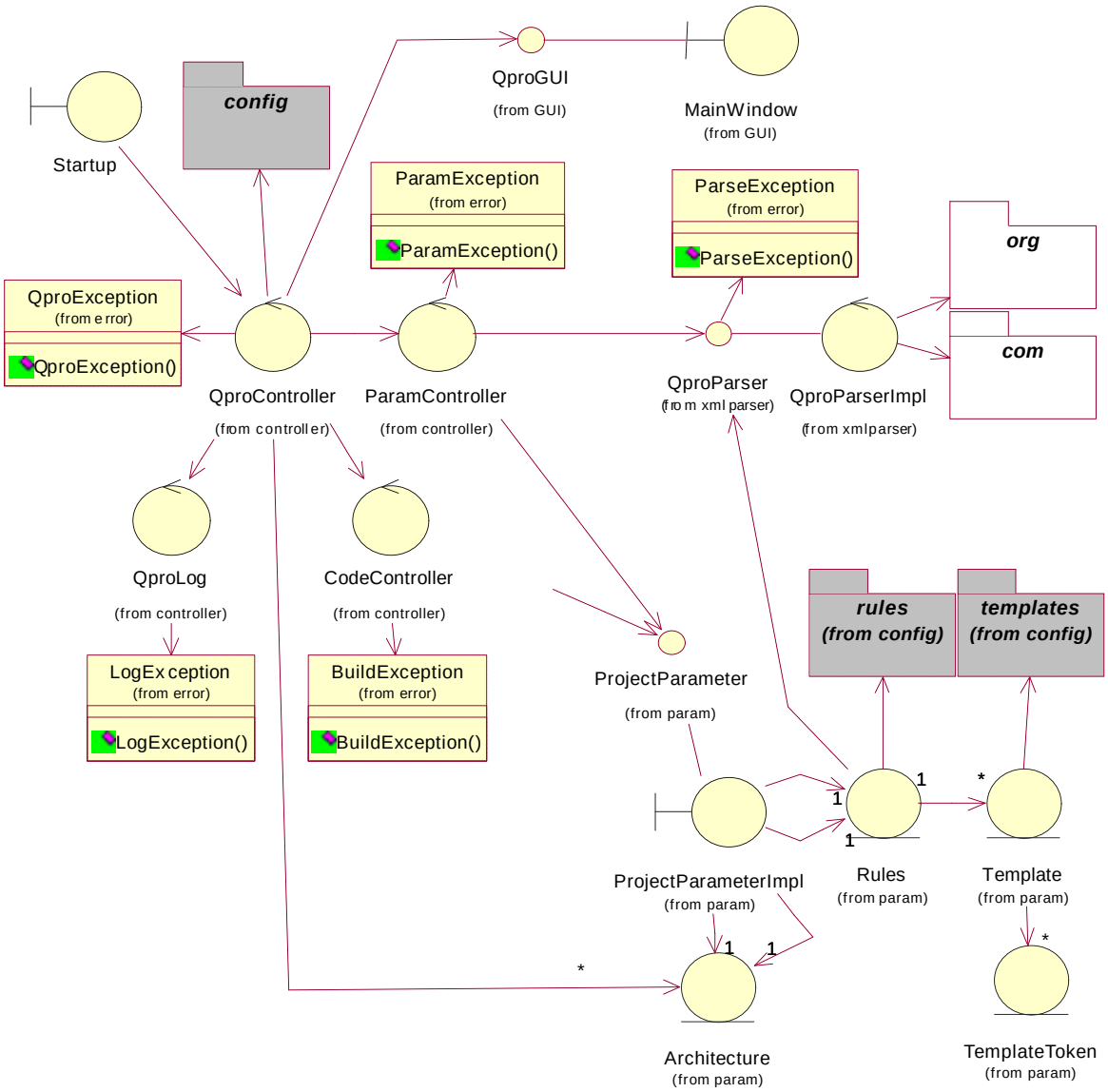
SAX (Simple API to XML) [XML3] er en hendelsesdreven parser API, som støtter mesteparten av de mange tilgjengelige XML parserne.

org.xml.sax.helper

Denne pakken håndtere enkle hjelpe klasser som kan hjelpe programmerer I å komme i gang med å bruke de ulike SAX bibliotekene.

6.4 Hele systemet

Dette kapittelet samler alt som har blitt beskrevet i detalj i en modell på klasse nivå.



Figur 6.6 – Overordnet klassediagram for hele systemet

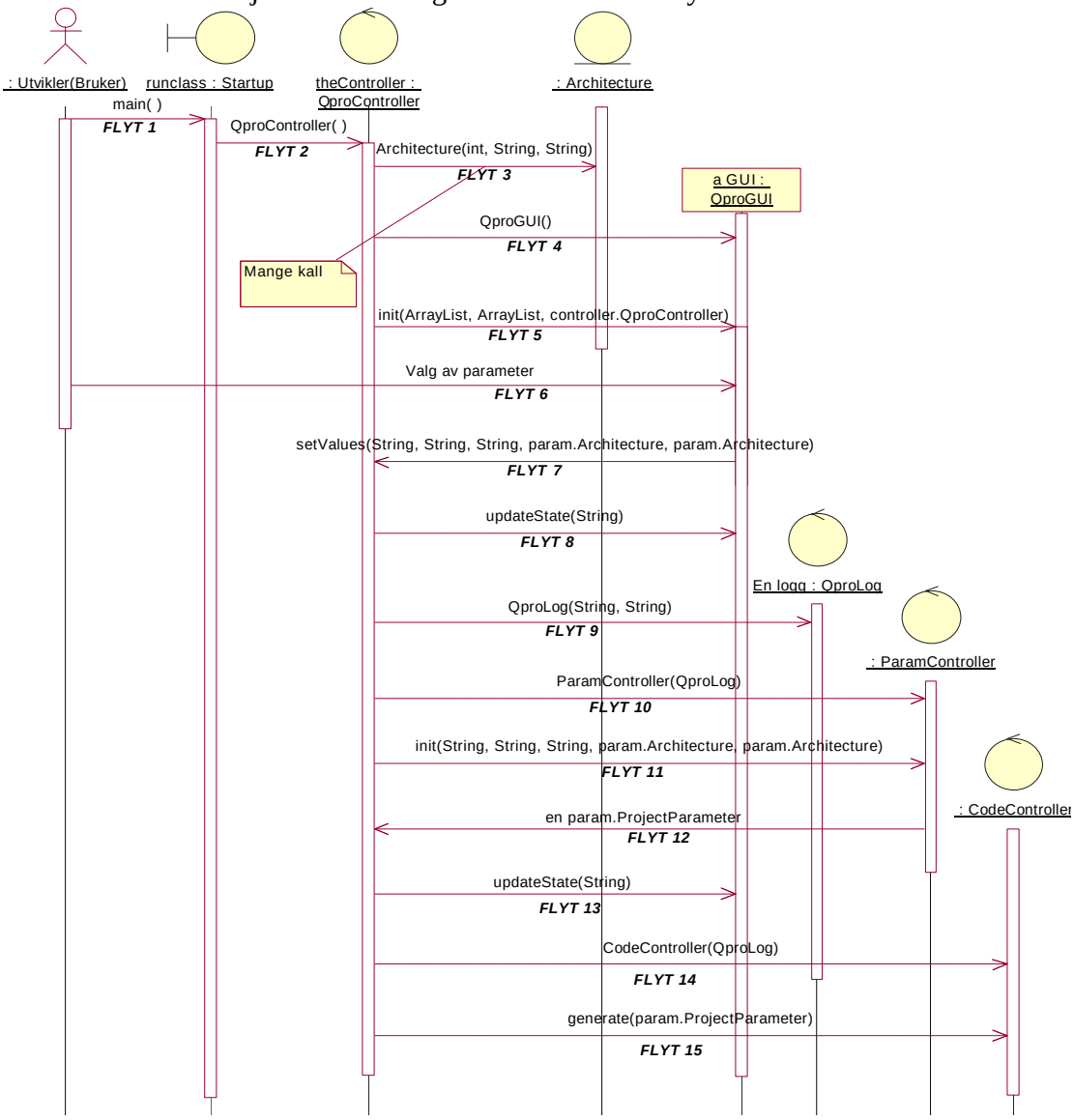
Modellen i figur 6.6 gir ingen ny kunnskap, den bare samler trådene av alt som har blitt beskrevet tidligere. For mer detaljert opplysninger henvises det til de detaljerte beskrivelsene gitt tidligere i dette kapittelet.

7 Intern kommunikasjon

Dette kapittelet belyser kommunikasjonen internt i systemkonstruksjonen ved standard vellykket kjøring. Den interne kommunikasjonene skildres ved hjelp av metodekall som blir gjennomført.

7.1 Overordnet kommunikasjon

Den overordnede kommunikasjonen tar utgangspunkt i 'controller.QproController' klassen og alle kall som går ut av denne. Da denne klassen har hovedansvaret for den interne kommunikasjonen vil dette gi et overblikk over systemet.



Figur 7.1 – Overordnet sekvensdiagram for metodekallene i Systemet

Flytbeskrivelse – flyt for flyt

FLYT 1 **main()**
Brukeren starter opp kjøreklassen til systemet

FLYT 2 **QPROController()**
Oppretter en instans av QproController via et kall til konstruktør.

FLYT 3 **Architecture(int, String, String)**
Et arkitekturobjekt opprettes for alle mulige valg, bade database arkitekturer og system arkitekturer.

FLYT 4 **QPROGUI()**
Et brukergrensesnitt som oppfyller grensesnittet skapes

FLYT 5 **init(ArrayList, ArrayList, controller.QproController)**
Brukergrensesnittet skapes som skal vise alle valgbare systemarkitekturer og databasearkitekturer og får en referanse tilbake til QproController

FLYT 6 **Valg av parameter**
Brukeren velger de arkitekturene som det skal genereres kode for, og setter de andre nødvendige parametrene. Dette må minst inneholde referanse til XMI-fil, navn på prosjekt, og målkatalog.

FLYT 7 **setValues(String, String, String, param.Architecture, param.Architecture)**
Brukergrensesnittet videreformidler de valgte verdiene til systemet.

FLYT 8 **updateState(String)**
Status skrives til brukergrensesnittet.

FLYT 9 **QproLog(String, String)**
En loggskriver opprettes i målkatalogen.

FLYT 10 **ParamController(QproLog)**
En instans av ParamController opprettes, og det sendes med en referanse til loggskriveren.

FLYT 11 **init(String, String, String, param.Architecture, param.Architecture)**
Alle parametere sendes over for validering og oppbygging av intern objektrepresentasjon.

FLYT 12 **en Param.ProjectParmeter**
Den interne representasjonen av parameteren returneres.

FLYT 13 **updateState(String)**
Status skrives til brukergrensesnittet.

FLYT 14 **CodeController(QproLog)**
En instans av CodeController opprettes, og det sendes med en referanse til loggskriveren.

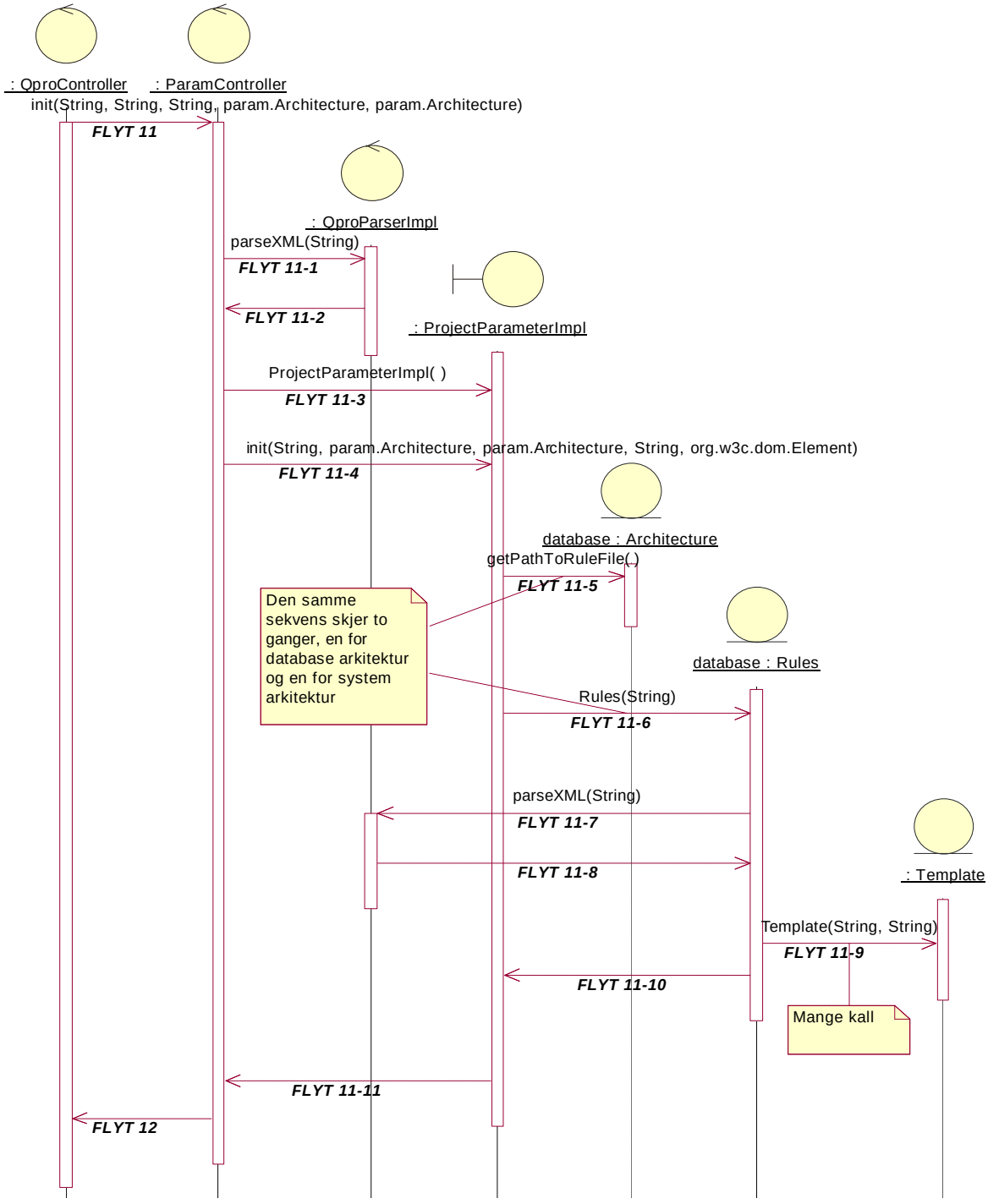
FLYT 15 **generate(param.ProjectParameter)**
Den interne objetktstrukturen sendes over for parsing

Kommentarer

Sett ut fra dette sekvensdiagrammet isolert er QproController en gudeklasse som sitter med det aller meste av ansvaret. Mye av ansvaret er dog fordelt på ParamController og CodeController, som vi skal se på de to neste sekvensdiagrammene. Disse tre Controller-klassene har sammen ansvaret for det meste av flyten, som man vil se her etter hvert.

7.2 Opprettelse av parameterobjekter

Dette kapittelet spesifiserer hva som skjer i 'FLYT 11' ParamController.init(…) fra figur 7.1 og gir et dypere innblikk i hvordan validering og oppbygging av den interne objektrepresentasjonen er tenkt.



Figur 7.2 – Detaljert sekvensdiagram om opprettelse av paramterobjektene

Flytbeskrivelse – flyt for flyt

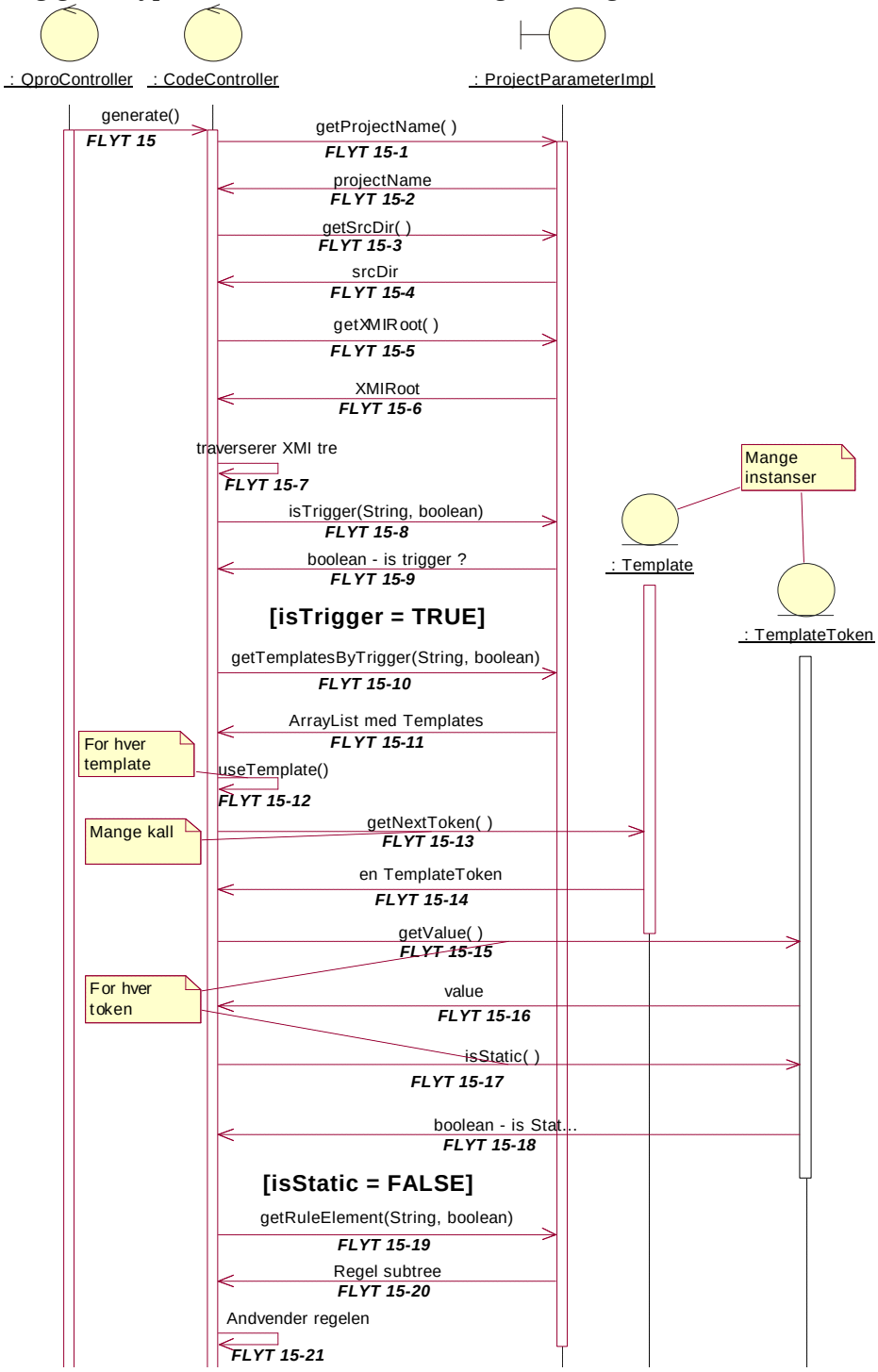
FLYT 11	**init(String, String, String, param.Architecture, param.Architecture)** Overordnet flyt fra kapittel 7.1, Beskrevet som: Alle parametere sendes over for validering og oppbygging av intern objekt representasjon.
FLYT 11–1	**parseXML(String)** XMI dokumentet parses og valideres til en trestruktur
FLYT 11–2	Trestrukturen returneres
FLYT 11–3	**param.ProjectParameterImpl** En instans av ProjectParameterImpl opprettes
FLYT 11–4	**init(String, param.Architecture, param.Architecture, String, Element)** Byggingen av den interne objekt strukturen starter
FLYT 11–5	**getPathToRuleFile()** Pathen til gjeldene arkitektur hentes ut.
FLYT 11–6	**Rules(String)** Oppbyggingen av et regelobjekt startes.
FLYT 11–7	**ParseXML(String)** Gjeldene regelfil parses og valideres til en trestruktur
FLYT 11–8	Trestrukturen returneres
FLYT 11–9	**Template(String, String)** Alle interne og eksterne mal objekter tilhørende regelfilen opprettes.
FLYT 11–10	Et ferdig Regel objekt returneres
FLYT 11–11	Et ferdig ProjectParameter objekt returneres
FLYT 12	**en Param.ProjectParmeter** Overordnet flyt fra kapittel 7.1, Beskrevet som: Den interne representasjonen av parameteren returneres.

Kommentarer
Kompleksiteten her ligger i det som skjer i Rules-objektet, når det blir opprettet. Kompleksiteten er internt i Rules, og medfører ikke mye kommunikasjon med andre objekter i konstruksjonen.

Flyt 11-5 til Flyt 11-10 gjentaes for hvert av de valgte arkitektobjektene.

7.3 Kodegenerering

Dette kapittelet spesifiserer hva som skjer i ’FLYT 15’ CodeController.generate(…) fra figur 7.1 og gir et dypere innblikk i hvordan kodegenereringen er tenkt.



Figur 7.3 – Detaljert sekvensdiagram for kodegenereringen

Flytbeskrivelse – flyt for flyt

FLYT 15 **generate(param.ProjectParameter)**
Overordnet flyt fra kapittel 7.1, beskrevet som:
Den interne objetktstrukturen sendes over for parsing.

FLYT 15–1, 2 **getProjectName()**
Henter prosjektnavnet.

FLYT 15–3, 4 **getSrcDir()**
Henter målkatalogen for koden

FLYT 15–5, 6 **getXMIRoot()**
Henter ut roten til treet som er resultatet av parsingen av XMI modellen.

FLYT 15–7 **Traverserere XMI tre**
XMI modellen traverseres for å kunne ta i bruk reglene og malene slik som beskrevet i kapittel 4.4

FLYT 15–8, 9 **isTrigger(String, boolean)**
Sjekker om den nåværende noden i XMI modellen utløser en mal. Dersom ikke forsetter traverseringen av XMI modellen og neste node sjekkes.

Dersom nåværende node utløste en mal (isTrigger =true)

FLYT 15–10, 11 **getTemplatesByTrigger(String, boolean)**
Forespør hvilke maler som matchet noden. Siden en node kan matche med flere maler returneres en liste over alle malene som skal brukes

FLYT 15–12 **useTemplate()**
Metode som kjøres for hver av malene returnert i flyt 15-11 og bruker malen.

FLYT 15-13, 14 **getNextToken()**
Henter ut neste delen av malen som skal taes i bruk.

FLYT 15-15, 16 **getValue()**
Henter ut verdien til den delen av malen som skal benyttes nå.

FLYT 15-17, 18 **isStatic()**
Finner ut om dette er en statisk verdi som kun skal skrives ut til koden, eller om det er en dynamisk parameter som det må benyttes regler på.

Dersom nåværende token ikke er statisk tekst. (isStatic = false)

FLYT 15-19, 20 **getRuleElement(String, boolean)**
Henter ut gjeldende regel for denne parameteren i malen.

FLYT 21 **Anvender regelen**
Bruker regelen på malen ved hjelp av verdiene i XMI-modellen.

8 Kravoppfyllelse

De funksjonelle kravene fra kravspesifikasjonen [KRA] beskriver funksjonalitet som systemet skal håndtere. Realiseringen av funksjonaliteten er spredt på forskjellige deler av systemet, og derfor knyttes her brukstilfeller opp mot pakkene og kravene opp mot de viktigste klassene og kravene. Det vil også bli nevnt litt om oppfyllelsen av de ikke-funksjonelle kravene.

8.1 Brukstilfeller koblet opp mot pakker

Systemet er delt inn i pakkene GUI, param, XMI-parser, error og config, og disse knyttes her opp mot brukstilfellene til systemet. Totalt kolonnen i tabellen nedenfor gir en oversikt over hvilke brukstilfeller som er dermed er dekket av hele systemet.

Forklaring av tegn
✔ betyr at brukstilfellet blir behandlet i pakken.

	GUI	param	XMI-parser	error	config	Totalt
1 Lage XMI-fil						
2 Generere kode fra XMI			✔	✔		✔
3 Sette prosjektinfo	✔	✔				✔
4 Velge komponentmal	✔	✔			✔	✔
5 Velge komponentregler	✔	✔			✔	✔
6 Velge katalogstruktur	✔	✔			✔	✔
7 Velge XMI-fil	✔	✔			✔	✔
8 Lage komponentregler						
9 Lage komponentmal						
10 Lage katalogstruktur						

Tabell 8.1 – Matrise for kobling av brukstilfeller opp mot pakker.

Kommentarer til matrisen
Matrisen viser at brukerens interaksjon med systemet i stor grad håndteres av pakkene 'GUI', 'Param' og 'Config', og i liten grad av de andre pakkene. Systemets viktigste oppgave, dekket av brukstilfelle 2, generere kode fra XMI, håndteres av XMI-parser- og Error-pakken.

Brukstilfelle 1 er ikke med i konstruksjonen, men tas hånd om ved hjelp av et eksternt modelleringsverktøy som Rational Rose eller Select. Brukstilfellene 2 til 7 involverer store deler av systemet, og kravene knyttet opp mot disse brukstilfellene blir derfor tatt hånd om i detalj. Brukstilfellene 8, 9 og 10 har det ikke blitt konstruert spesiell støtte for. Regel- og malfiler må skrives for hånd, og katalogstrukturen blir angitt i reglene.

8.2 Krav koblet opp mot klasser, pakker og regel/mal-format

De fleste av kravene implementeres i de tre pakkene Error, XMI-parser og Param, samt de tre klassene QproController, CodeController, ParamController og regel/mal-formatet spesifisert ved hjelp av XML. Totalt kolonnen i tabellen nedenfor viser at alle kravene i kravspesifikasjonen vil bli oppfylt med det konstruksjonen som er blitt valgt.

Pakker er skrevet i **fet skrift**.
Klasser er skrevet vanlig skrift.
XML-filer skrevet i *kursiv*.

I X betyr Input krav X.
P X betyr Parse krav X.
U X betyr Utdata krav X.

Forklaring av tegn

✔ betyr at kravet blir behandlet i pakken, klassen eller av regel/mal-formatet.

	error	Qpro Controller	Code Controller	**XMI-parser**	Param Controller	**param**	*Regel/mal-format*	Totalt
I 1				✔				✔
I 2		✔			✔	✔		✔
I 3						✔		✔
I 4	✔		✔	✔	✔			✔
I 5				✔			✔	✔
I 6				✔			✔	✔
I 7							✔	✔
P 1			✔				✔	✔
P 2			✔			✔	✔	✔
P 3			✔				✔	✔
P 4			✔				✔	✔
P 5			✔				✔	✔
P 6		✔	✔			✔		✔
U 1			✔				✔	✔
U 2			✔				✔	✔
U 3			✔				✔	✔
U 4			✔				✔	✔
U 5			✔				✔	✔
U 6			✔				✔	✔

Tabell 8.2 – Matrise for kobling av krav opp mot klasser, pakker og regel/mal-format.

Kommentarer til matrise

For inndata krav 4 gjelder at en eventuell feil i XMI'en kan oppdages på forskjellige tidspunkt. Feil filreferanser håndteres av klassen ParamController, mens feil XMI-struktur kontrolleres av XMI-parseren. Feil i innholdet i XMIen kontrolleres av CodeController. Når feil oppstår kastes det unntak av typen BuildException, ParamException respektive ParseException med en beskrivende feilmelding.

For alle parse- og utdatakrav gjelder at genereringen tas hånd om av CodeController. Tilpasning til forskjellige teknologier tas hånd om av reglene og malene. CodeController er dermed 100% uavhengig av teknologivalget.

Parsekrav 4 handler om at brukeren enkelt skal kunne lage egne regler og maler. Dette avhenger av hvor lett det er å forstå strukturen vi har valgt i regel- og malfilene. Kravet gjør det naturlig å implementere et eget program for oppretting av slike filer, men det finnes ikke i dette konstruksjonen, og må derfor betraktes som en mulig senere utvidelse. Hvis nye regel- og malfiler skal fungere opp mot en ny teknologi, må CodeController være totalt uavhengig av teknologi og av XMI-input til systemet.

8.3 Ansvarsfordeling

Ut fra krav/klasse-matrisen er det tydelig at CodeController og regel/mal-format er involvert i det store flertallet av kravene. Dette indikerer den svært sentrale posisjonen til CodeController og regel/malformatet har i det konstruerte systemet.

De fleste parsekravene handler om konfigurering av kodegeneratoren i forhold til forskjellige teknologier, for eksempel SQL, VB og EJB. De fleste utdatakravene handler om hvilke klasser, skript og kode som skal genereres. Alle variasjoner på disse områdene dekkes av muligheten til å lage forskjellige regelfiler for hver komponentteknologi, og egne malfiler for alle de forskjellige klassene som skal genereres innen en teknologi. Krav knyttet til VB støttes av egne mal- og regelfiler, mens krav knyttet til EJB støttes av andre filer. Dette er også en form for modularisering, og ansvaret for forskjellige teknologier spres på det vi kan kalle mal- og regelmoduler.

CodeController er involvert i veldig mange krav, siden det er den klassen som har ansvaret for at det blir produsert noe. Oppgaven til CodeController er så generell, og så avhengig av hva som ligger i regel- og malfilene, at ansvaret til CodeController er mye mindre enn krav/klasse-matrisen kan gi inntrykk av.

8.4 Oppfyllelse av ikke-funksjonelle krav

Når det gjelder kravet til maskin- og programvare virker det som om vi har klart å hindre maskinvareavhengighet og støtter flere operativsystemer enn MS Windows. Programkoden som vi har konstruert et skjellett for ved hjelp av Rational Rose skulle heller ikke by på de helt store kravene til prosesseringskraft eller minneforbruk. En PC med 350 MHz prosessor og 128 MB RAM bør altså være mer enn tilstrekkelig. Vi mener at en slik PC ikke vil ha problemer med å møte de romslige kravene til tidsforbruk ved generering av kode eller subsett av kode.

Programkoden blir i sin helhet utviklet i Java SDK 1.3.1 og hver programmerer har ansvaret for å lage og oppdatere Javadoc samtidig med utviklingen av koden. Når det gjelder tredjeparts programvare har vi valgt å benytte oss av en ekstern Java XML-parser fra SUN. Denne pakken er blitt godkjent av kunden i henhold til miljø krav 2 som sier at alle eksterne pakker skal godkjennes av kunde. Konstruksjonen og prosjektoppfølging tilsier at de ikke-funksjonelle kravene blir oppfylt i henhold til beskrivelsen i kravspesifikasjonen.

9 Bevis av konsept

Som et risikominkende tiltak ble det implementert et bevis av konsept (”proof of concept”) som skulle vise at konseptet i konstruksjonen var praktisk mulig å gjennomføre samt å prøve ut de valgene som ble gjort. Dette kapittelet forklarer hva som ble gjort, hvordan det ble gjort og hva som var resultatet.

9.1 Hva

Bevis av konsept skulle implementeres som et Java basert program som tar inn en beskrivelse av en modell i XML-format og brukte et regelsett lagret som XML filer til å generer kodefiler og en databasefil. Programmet skulle videre bruke tredjepartsverktøyene i pakkene org og com til å parse de ulike XML-filene.

Inndataene ble fastlagt til å være på følgende enkle format:

```
<ENTITIES>
      <ENTITY>
            <NAME>..</NAME>
            <ATTRIBUTE>..
                  <NAME>..</NAME>
                  <TYPE>..</TYPE>
            </ATTRIBUTE>
            <RELATION>
                  <FIELD>..</FIELD>
                  <ENTITY>..</ENTITY>
                  <CARDINALITY1>..</CARDINALITY1>
                  <CARDINALITY2>..</CARDINALITY2>
            </RELATION>
      </ENTITY>
</ENTITIES>
```

Reglene ble fastlagt til å være på følgende enkle format:

```
<rules>
      <control><code>..<code><sql>..</sql></control>
<entity>
   <start><code>..</code><sq>..</sql></start>
   <name><code>..</code><sql>..</sql></name>
   <attribute><code>..</code><sql>..</sql></attribute>
   <method><code>..</code><sql>..</sql></method>
   <relation><code>..</code><sql>..</sql></relation>
   <end><code>..</code><sql>..</sql></end>
</entity>
</rules>
```

Der ’..’ var malen som skulle settes inn ved bruk av denne regelen. En slik mal kunne da inneholde flere %parameter% som måtte da byttes ut med korrekt verdi. Her ser man tydelig matchingen mellom taggen benyttet i inndataene og tilhørende regel i regelfilen. Se for øvrig vedlegg 4 – ’Bevis av konsept’ for eksempel på inndata og regler

9.2 Hvordan

Beviset ble realisert som en svært forenklet versjon av det fullstendige konstruksjonen. Totalt 4 klasser ble implementert: 'POCStart', 'POCParser', 'POCRules', og 'POCCodeBuilder'.

'POCStart' er en forenklet versjon av funksjonalitet konstruert i klassene 'Startup' og 'controller.QproController'. Denne klassen tar dermed for seg den interne kommunikasjonen og oppstarten.

'POCParser' var en klasse som kun parset XML-dokumenter ved hjelp av org og com pakkene.

'POCRules' er en forenklet versjon av konstruert funksjonalitet i og 'param' pakken 'controller.ParamController' klassen. Denne klassen tar for seg parsingen av innparameterne og all bruk av de enkle reglene, samt skrivingen til fil.

'POCCodeBuilder' er en forenklet versjon av 'controller.CodeController'.

Se vedlegg 4 –'Bevis av konsept' for kildekoden.

9.3 Resultat

Resultatet av beviset var en kodegenerator som kunne generer veldig enkel kode på grunnlag av det minimalistiske inndataformatet. Det at denne kodegeneratoren fungerte i praksis viser at konseptet som ligger til grunne for konstruksjonen fungerer i praksis.

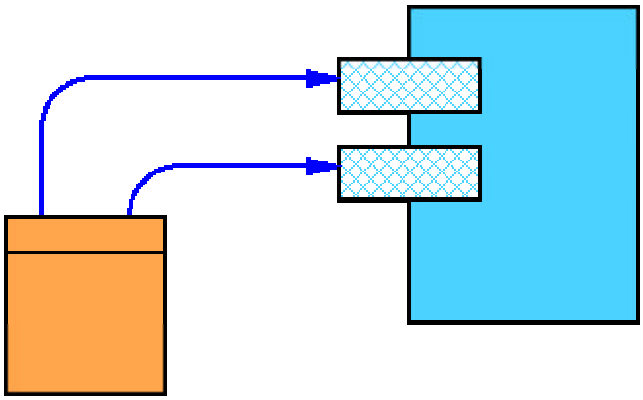
Tredje parts pakkene org og com ble forsket på og etter hvert mye brukt, slik at tiden det vil ta å implementere de delene av det faktiske systemet som har med dette å gjøre vil bli betraktelig lavere.

Se for øvrig vedlegg 4 – 'Bevis av konsept' for kjøreeksempel og resulterende kode.

10 Testprosedyrer

Først har vi modultest som handler om test av de ulike modulene i systemet. Deretter kommer integrasjonstest der vi setter sammen moduler og komponenter til et større system. Til slutt har vi regresjonstest som omhandler korrigering av feil.

10.1 Modultest



Figur 10.1 – Modultest.

Den blå boksen forestiller en modul og den oransje forestiller testen av denne modulen.

Ved modulær programmering i de fleste av dagens programmeringsspråk tenker vi på enkelte programsnutter som i innhold og logikk naturlig danner en enhet eller modul. Ved å benytte oss av spesialdefinerte grensesnitt mellom modulene sikrer vi et klart skille mellom de ulike modulene og gjør det samtidig enkelt å teste de ulike modulene uavhengig og hver for seg. Man lager altså små programsnutter som tester ulike scenarioer ved å kalle metoder i grensesnittet til en modul og undersøke at det som returneres fra hver av disse er fornuftig. Det er viktig at man på forhånd har tenkt igjennom hva man vil få ut av et bestemt testscenario slik at man ikke godtar returverdier som er gale. Modultest gjennomføres så snart den som programmerer føler seg ferdig med modulen og gjennomføres av den som har programmert modulen slik at han kan rette feilene han finner før han leverer modulen fra seg.

10.1.1 Praktisk gjennomføring

Vårt system består av følgende moduler/pakker: Controller-pakke, XML-parser-pakke, Param-pakke, GUI-pakke, Error-pakke. Det er blitt bestemt at de ansvarlige for programmeringen av en modul også er ansvarlige for å lage en testmodul for å teste sin modul. I de påfølgende tabellene er det forsøkt å sette opp endel testmomenter som

programmerer må tenke på når programmerer utarbeider disse testklassene. Testpunktene er prioritert og koblet opp mot kravene fra kravspesifikasjonen. Alle testresultater fra modultesten skal fremlegges kvalitetsansvarlig Atle Nes, som vil fremlegge dette på neste prosjektmøte. Testresultatene skal føres inn en felles testlogg.

Eksempel på testlogg:

TestID	Testdato	Testtidspunkt	Testet av	Resultat
Parser1	05.11.2001	08.57	Martin Sleire Vatne	FEIL: <det som ble avdekket>
Parser2	05.11.2001	09.05	Odd Christian Landmark	OK
Param1	06.11.2001	10.15	Atle Nes	OK

Dersom feil rettes skal disse føres i en egen endringslogg, og testen gjennomføres på ny.

Eksempel på endringslogg:

TestID	Endringsdato	Endrings-tidspunkt	Endret av	Endring
Parser1	05.11.2001	08.57	Martin Sleire Vatne	La inn glemt parameter
Parser2	05.11.2001	09.05	Odd Christian Landmark	Fikset opp korrupt metode
Param1	06.11.2001	10.15	Atle Nes	Endret på innparametere

10.1.2 Modultest av 'XMLparser'-pakken

ID	Beskrivelse	Krav	Prioritet
Modultest Parser 1	Testmoment godkjennes om det kan det leses inn fra fil en XMI-representasjon eksportert fra Rational Rose.	Inndata Krav 1,5	Kritisk
Modultest Parser 2	Testmoment godkjennes om det kan leses inn fra fil en XMI-representasjon eksportert fra Select.	Inndata Krav 6	Middels
Modultest Parser 3	Testmoment godkjennes dersom det ikke stilles så store og komplekse krav til XMI-inndata formatet at brukeren selv ikke kan lage gyldige dokumenter.	Inndata Krav 7	Middels
Modultest Parser 4	Testmoment godkjennes dersom det blir generert en tekstlig beskrivelse av treet til loggen.		Lav
Modultest Parser 5	Testmoment godkjennes dersom det blir generert et parset tre av XML dokumentet.		Kritisk

Modultest Parser 6	Testmoment godkjennes dersom pakken kaster ParseException hvis XMI-filen som skal benyttes er korrupt. Det skal komme en beskrivende feilmelding.	Inndata Krav 4	Middels

10.1.3 Modultest av 'param'-pakken

ID	Beskrivelse	Krav	Prioritet
Modultest Param 1	Testmoment godkjennes om pakken håndterer parametere som omhandler hvilken type database som blir benyttet og har evnen til å delegere denne informasjonen videre til andre pakker.		Kritisk
Modultest Param 2	Testmoment godkjennes om pakken håndterer parametere som omhandler hvilken arkitektur som er blitt valgt og har evnen til å delegere denne informasjonen videre til andre pakker.		Kritisk
Modultest Param 3	Testmoment godkjennes dersom pakken kan ta fortelle om en tag i XMI modellen hører til en eller flere maler og kan returnere en liste av disse.		Middels
Modultest Param 4	Testmoment godkjennes dersom pakken kan returnere rotnoden til regelen som matcher en gitt tag.		Middels

10.1.4 Modultest av 'GUI'-pakken

ID	Beskrivelse	Krav	Prioritet
Modultest GUI 1	Testmoment godkjennes dersom det i brukergrensesnittet er mulig å velge databasearkitektur, med eventuelt standard SQL som et standardvalg.	Parse Krav 5	Lav
Modultest GUI 2	Testmoment godkjennes dersom det i brukergrensesnittet er mulig å velge ulike arkitekturer som det skal genereres kode for, med eventuelt EJB som standardvalg.	Inndata Krav 2	Høy
Modultest GUI 3	Testmoment godkjennes dersom statusen til systemet blir presentert for brukeren slik at det går an å se hvor langt systemet er kommet. Dette kan for eksempel være en enkel tekststreng.		Middels

Modultest GUI 4	Testmoment godkjennes dersom det finnes en exit knapp slik at man kan avslutte programmet på en ryddig måte.		Middels
Modultest GUI 5	Testmoment godkjennes om man i brukergrensesnittet kan velge hvor kildekoden skal legges når den er ferdig generert.	Inndata Krav 2	Høy
Modultest GUI 6	Testmoment godkjennes dersom man i brukergrensesnittet har muligheter for å definere prosjektnavn.	Inndata Krav 3	Middels

10.1.5 Modultest av 'controller'-pakken

ID	Beskrivelse	Krav	Prioritet
Controller Modultest 1	Testmoment godkjennes om pakken sørger for logging av systemhendelser til fil.		Lav
Controller Modultest 2	Testmoment godkjennes om det er det lagt opp til initiering av et brukergrensesnitt.		Middels
Controller Modultest 3	Testmoment godkjennes om det er tatt høyde for håndtering av feil både i egen pakke og tilstøtende pakker.		Middels

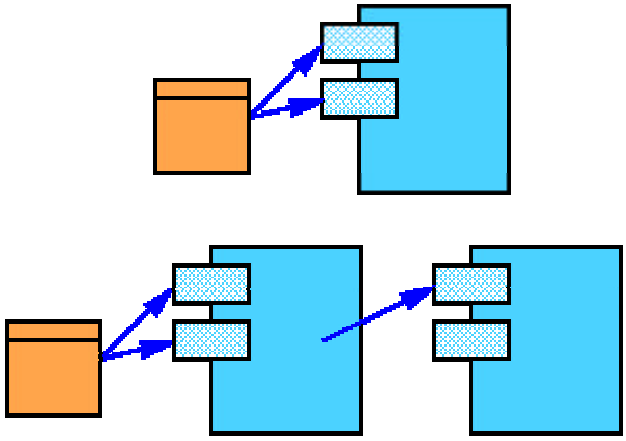
10.1.6 Modultest av 'error'-pakken

ID	Beskrivelse	Krav	Prioritet
Modultest Error 1	Testmoment godkjennes dersom pakken kaster de korrekte unntakene avhengig av hvilken del av systemet som det oppstår en feilsituasjon i. Unntakene her er QproException, BuildException, ParamException, ParseException og LogException.		Middels

10.2 Integrasjonstest

Ved integrasjonstest blir de enkelte ferdig testede komponenter og moduler satt sammen til et stort system. Man antar altså at de enkelte delene fungerer godt alene og tester altså kun samspillet mellom komponenter og moduler. Dette handler mye om at kommunikasjon og parameteroverføringer fungerer. Ved større systemer er det vanlig å

utføre integrasjonstest etter hvert som man legger til flere moduler og dermed forenkle feilsøkingen i systemet. Dette blir ikke aktuelt i vårt tilfelle ettersom vi har et system med relativt få moduler. Når man har fått vekk de feil som hindrer samspillet mellom modulene, bør man begynne å teste systemet i praktisk bruk og gjerne på en representativ brukergruppe. Alfatest er tester som gjøres av utviklerne når systemet er ferdig kalles alfatest i motsetning til betatest som gjøres av representative brukere.



Figur 10.2 – Integrasjonstest med gjenbruk fra modultest.

Bottom-up-testing tar utgangspunkt i de minste modulene og tester disse. For hver modul utvikles en testmodul som tester denne modulen. Så snart en modul er testet blir modulene som benytter seg av denne inkludert en etter en. Testen av disse modulene vil nødvendigvis benytte lavnivå-modulene og dermed gjentatt teste de mest brukte delene av programkoden.

Top-down-testing går motsatt vei. Man starter med den mest sentrale delen av programmet og arbeider seg utover. Det blir altså gradvis lagt til mer funksjonalitet underveis. Modulene blir altså simulert av små kodebiter.

10.2.1 Praktisk gjennomføring

Det virker som en hybrid løsning passer best i prosjektet vårt. Dette innebærer at vi kombinerer top-down og bottom-up-teknikker. Det er mest naturlige er at vi starter med XML-parser-pakken, deretter integrerer vi Param pakken og Controllen pakken. Begge disse pakkene benytter seg av XML-parseren og det vil derfor være ønskelig å få testet ut funksjonaliteten opp imot denne. Til slutt legges vi til GUI-pakken og Error-pakken.

Integrasjonstesten utføres ved å bruke de allerede ferdige modultestklassene, samt eventuelle nye momenter som man ønsker å få testet. Dette handler i første omgang om test av grensesnitt og kommunikasjon mellom de ulike modulene for å se at det ikke modulene er inkompatible med hverandre. Gruppa tar sikte på å gjennomføre en integrasjonstest så snart de ulike modulene er blitt ferdig implementert og testet av

programmerer. Selve integrasjonstesten utføres som et samarbeid mellom de ulike programmererne. Derfor vil det være naturlig at de fleste er med på selve integrasjonen i mer eller mindre grad. Hovedansvaret for integrasjonstesten ligger likevel hos kvalitetsansvarlig Atle Nes, som vil rapportere status og resultater på prosjektmøtet. Som en utvidelse av integrasjonstesten eller i sammenheng med integrasjonstesten vil det bli utført en intern akseptansetest.

Kronologisk gjennomgang av forberedelser til akseptansetest:

?? Lag en UML modell ved hjelp av et egnet modelleringsverktøy (fortrinnsvis Rational Rose, eventuelt Select). Denne modellen bør inneholde nok objekter til at man kan få et inntrykk av tidsforbruket ved kjøring.
?? Eksporter UML modellen til en XMI representasjon.
?? Bruk Qpro-Generator til å få generert SQL og JavaBean kode fra XMI'en.
?? Bruk SQL'en til å opprette databasen i MS SQL Server 2000.
?? Deploy JavaBean kode på WebLogic 6.1 server med eller uten ANT hjelpeverktøy.

For å få gjennomført dette har vi satt opp MS SQL Server 2000 og WebLogic 6.1 Server på testmaskinen. Testmaskinen er en Pentium III 1000 MHz med 256 MB RAM. Godkjenning av Qpro-Generatoren skjer ved å la en testklient gjøre endringer i databasen ved å kommunisere med JavaBean kode som vi har lagt ut på WebLogic serveren. Ettersom denne koden er Container Managed vil oppdateringer mot databasen skje automatisk, vi slipper altså å skrive SQL kode inn i JavaBean koden. Om oppdateringene definert i klienten gjenspeiles korrekt i databasen under kjøring er generatoren altså godkjent.

10.3 Regresjonstest

En regresjonstest finner sted når en feil blir korrigert. Da må testscenarioene for den modulen som er blitt korrigert gjennomføres på ny, for å sikre at:

?? feilen virkelig er blitt borte.
?? det ikke er dukket opp noen nye feil som tidligere var skjult av feilen.
?? det ikke er blitt bygget inn nye feil ved korrigeringen.

10.4 Inspeksjon av kode

Når den eller de som programmerer føler seg ferdig med sin kode skal denne koden inspiseres av personer som ikke har vært med på å programmere denne kodebiten. Inspeksjon av kode er en god metode for å luke ut ikke bare slurvefeil, men spesielt "ad hoc" metoder som bør omskrives. Inspektøren får i oppgave å levere godt kommentert programkode tilbake til programmerer som tar stilling til synspunktene som er blitt fremlagt av inspektør og retter opp det som bør rettes opp. Koden skal, som nevnt i kravspesifikasjonen, følge SUN Java 2 SE v1.3.1 sine kodestandarder. Inspektør skal forsikre seg om at det ikke blir brukt andre pakker enn org.w3c.dom og com.sun.xml. Vedlegg 5 inneholder en sjekkliste for javakode hvor inspektør kan hake ut sjekkpunkter etter hvert som de er kontrollert.

11 Mulige utvidelser

I dette kapittelet vil noen mulige utvidelser av kodegeneratoren bli beskrevet

11.1 Konverteringsverktøy for XMI

Slik systemet blir laget i dag må man i reglene ta hensyn til hvilken versjon av XMI som bli valgt som input. Dette fører til at man må lage flere regelsett (for samme arkitektur) hvis man vil støtte flere versjoner av XMI-standarden. Det som er en mulig utvidelse av generatoren er at man lager et konverteringsverktøy som kan ta inn flere versjoner av XMI, og generere en standard versjon som brukes videre. Dette vil føre til at man ikke trenger å tenkte på XMIen når man setter opp reglene. Man kan da også hardkode mye mer i selve generatoren fordi man vil være sikker på at XMIen vil inneholde kjente nødvendige attributter. Dette konverteringsverktøyet vil eventuelt bli et lag mellom XMI-parseren og kodegenereatoren.

11.2 Verktøy for oppretting/editering av regelfiler

Regelfilene til kodegeneratoren vil være noe kompleks. Dette er fordi mye informasjon må inn i reglene for at generatoren skal bli helt generell. Reglene vil også stort sett ha samme oppbygning, og det kan derfor tenkes at et verktøy for å lage og editere slike regler ikke vil være så vanskelig å utvikle, samtidig som det vil gjøre jobben med å opprette og endre nye regelfiler mye enklere.

11.3 Støtte for flere enn to arkitekturer samtidig

Det kan tenkes at det er ønskelig å kunne generere kode for flere arkitekturer. For eksempel vil man kanskje lage databaseskript for til database sammen med kodegenereringen. Kanskje vil man også lage kode for COM og EJB samtidig. Denne utvidelsen medfører at generatoren må endres noe, men jobben er ikke alt for omfattende.

12 Indekser

12.1 Figurliste

12.2 Tabelliste

13 Litteraturliste

[DOM1] Document Object Model
http://www.w3.org/DOM/

[JAV1] The Java™ 2 Platform, Standard Edition v1.3.1_01
http://java.sun.com/j2se/1.3/
http://java.sun.com/j2se/1.3/docs/api/index.html

[SUN1] Java API for XML Parsing
http://granitepeaks.com:8011/webdocs/jaxp1_0-ea1/docs/api/

[XMI1] XML Metadata Interchange
http://www-4.ibm.com/software/ad/library/standards/xmi.html

[XML1] Extensible Markup Language (XML)
http://www.w3.org/XML/

[XML2] W3C DOM Level 1 Core Recommendation
http://www.w3.org/TR/REC-DOM-Level-1/

[XML3] Simple API for XML
http://sax.sourceforge.net/

14 Vedlegg

14.1 Vedlegg 1 – Eksempel på mal

```
public abstract class %CLASSNAME%EJB implements javax.ejb.EntityBean{

       private javax.ejb.EntityContext ctx;

       public %CLASSNAME%EJB(){

       }

       public void ejbActivate(){

       }

       public void ejbPassivate(){

       }

       public void ejbLoad(){

       }

       public void ejbStore(){

       }

       public void ejbRemove() throws javax.ejb.RemoveException{

       }

       public void setEntityContext(javax.ejb.EntityContext ctx){
              this.ctx=ctx;
       }


       public void unsetEntityContext(){
              this.ctx=null;
       }

       public void ejbCreate(%INPUT_ATTRIBUTES%) throws
javax.ejb.CreateException{

    %EJB_CREATE%

       }

       public void ejbPostCreate() throws javax.ejb.CreateException{

       }

       /** CONTAINER MANAGED FIELDS */
    %CONTAINER_MANAGED_FIELDS%



    %OPERATION%
}
```

14.2 Vedlegg 2 – Eksempel på regelfil

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE RULES SYSTEM "rules.dtd">
<RULEFILE>
   <VARIABLE_CHARACTER char="%" />
   <ID_ATTRIBUTE name="xmi.id" />
   <TEMPLATES>
     <T1 trigger="Foundation.Core.Class" source="ejb2_ejb.qpt"
        target="%CLASSNAME%EJB.java" />
     <T2 trigger="Foundation.Core.Class" source="ejb2_pk.qpt"
        target="%CLASSNAME%PK.java" />
     <T3 trigger="Foundation.Core.Class" source="ejb2_home.qpt"
        target="%CLASSNAME%Home.java" />
     <T4 trigger="Foundation.Core.Class"
        source="ejb2_remote.qpt" target="%CLASSNAME%.java"
        />
   </TEMPLATES>
   <INTERNAL_TEMPLATES>
     <ATTRIBUTE_TEMPLATE>%ATTRIBUTE_VISIBILITY%
        %ATTRIBUTE_TYPE%
        %ATTRIBUTE_NAME%;</ATTRIBUTE_TEMPLATE>
     <INPUT_ATTRIBUTE_TEMPLATE>%ATTRIBUTE_TYPE%
        %ATTRIBUTE_NAME%</INPUT_ATTRIBUTE_TEMPLATE>
     <STANDARD_OPERATION>%ATTRIBUTE_VISIBILITY%
        %ATTRIBUTE_TYPE%
        %ATTRIBUTE_NAME%</STANDARD_OPERATION>

        <OPERATION_TEMPLATE>/**************************
        ************************************ * @Name
        %OPERATION_NAME% * * @Input
        %OPERATION_INPUT% * @Return %OPERATION_TYPE%
        ********************************************
        ****************/ %OPERATION_VISIBILITY%
        %OPERATION_TYPE%
        %OPERATION_NAME%(%OPERATION_INPUT%){ return
        %RETURN_DEFAULT_VALUE%; }</OPERATION_TEMPLATE>

        <INTERFACE_OPERATION_TEMPLATE>/****************
        ********************************************
        * * @Name %OPERATION_NAME% * * @Input
        %OPERATION_INPUT% * @Return %OPERATION_TYPE%
        ********************************************
        ****************/ %OPERATION_VISIBILITY%
        %OPERATION_TYPE%
        %OPERATION_NAME%(%OPERATION_INPUT%);</INTER
        FACE_OPERATION_TEMPLATE>

        <EJB_CREATE>set%ATTRIBUTE_NAME%(%ATTRIBUTE_N
        AME%);</EJB_CREATE>
```

```
    <CONTAINER_MANAGED_FIELDS_TEMPLATE>public abstract
      %ATTRIBUTE_TYPE% get%ATTRIBUTE_NAME%(); public
      abstract void
      set%ATTRIBUTE_NAME%(%ATTRIBUTE_TYPE%
      val);</CONTAINER_MANAGED_FIELDS_TEMPLATE>
    <OPERATION_INPUT>%OPERATION_INPUT_TYPE%
      %OPERATION_INPUT_NAME%</OPERATION_INPUT>
    <OPERATION_TYPE>%RETURN_TYPE%</OPERATION_TYPE>

      <OPERATION_RETURN_TYPE>%RETURN_DEFAULT_TYPE%<
      /OPERATION_RETURN_TYPE>
  </INTERNAL_TEMPLATES>
  <RULES>
    <IMPORT />
    <CLASSNAME type="replace;filereplace" link=""
      source="Foundation.Core.ModelElement.name"
      target="%CLASSNAME%" mandatory="true" />
    <CLASS_VISIBILITY type="replace" link=""
      source="Foundation.Core.ModelElement.visibility;xmi.valu
      e" target="%CLASS_VISIBILITY%" />
    <CLASS_TYPE />
    <CLASS_EXTENDS />
    <CLASS_INHERITS />
    <CLASS_IMPLEMENTS />
    <ATTRIBUTE type="multi" source="Foundation.Core.Attribute"
      template="ATTRIBUTE_TEMPLATE" mandatory="false"
      separator="">
      <ATTRIBUTE_VISIBILITY type="replace" link=""
        source="Foundation.Core.ModelElement.visibility;xmi.
        value" target="%ATTRIBUTE_VISIBILTY%" />
      <ATTRIBUTE_TYPE type="replace"
        link="Foundation.Core.Classifier;xmi.idref"
        source="Foundation.Core.ModelElement.name"
        target="%ATTRIBUTE_TYPE%" />
      <ATTRIBUTE_NAME type="replace" link=""
        source="Foundation.Core.ModelElement.name"
        target="%ATTRIBUTE_NAME%" />
    </ATTRIBUTE>
    <INPUT_ATTRIBUTES type="multi"
      source="Foundation.Core.Attribute"
      template="INPUT_ATTRIBUTE_TEMPLATE"
      mandatory="false" separator=",">
      <ATTRIBUTE_VISIBILITY type="replace" link=""
        source="Foundation.Core.ModelElement.visibility;xmi.
        value" target="%ATTRIBUTE_VISIBILTY%" />
      <ATTRIBUTE_TYPE type="replace"
        link="Foundation.Core.Classifier;xmi.idref"
        source="Foundation.Core.ModelElement.name"
        target="%ATTRIBUTE_TYPE%" />
      <ATTRIBUTE_NAME type="replace" link=""
        source="Foundation.Core.ModelElement.name"
        target="%ATTRIBUTE_NAME%" />
```

```
</INPUT_ATTRIBUTES>
 <STANDARD_OPERATION type="multi"
    source="Foundation.Core.Attribute"
    template="STANDARD_OPERATION" mandatory="false"
    separator="">
    <ATTRIBUTE_VISIBILITY type="replace" link=""
       source="Foundation.Core.ModelElement.visibility;xmi.
       value" target="%ATTRIBUTE_VISIBILTY%" />
    <ATTRIBUTE_TYPE type="replace"
       link="Foundation.Core.Classifier;xmi.idref"
       source="Foundation.Core.ModelElement.name"
       target="%ATTRIBUTE_TYPE%" />
    <ATTRIBUTE_NAME type="replace" link=""
       source="Foundation.Core.ModelElement.name"
       target="%ATTRIBUTE_NAME%" />
</STANDARD_OPERATION>
 <OPERATION type="multi"
    source="Foundation.Core.Operation"
    template="OPERATION_TEMPLATE" mandatory="false">
    <OPERATION_VISIBILITY type="replace" link=""
       source="Foundation.Core.ModelElement.visibility;xmi.
       value" target="%OPERATION_VISIBILTY%" />
     <OPERATION_TYPE type="multi"
       source="Foundation.Core.Parameter"
       template="OPERATION_TYPE"
       sourcerestriction="Foundation.Core.Parameter.kind;xmi
       .value;return">
       <RETURN_TYPE type="replace"
          link="Foundation.Core.Classifier;xmi.idref"
          source="Foundation.Core.ModelElement.name" />
    </OPERATION_TYPE>
     <RETURN_DEFAULT_VALUE type="multi"
       source="Foundation.Core.Parameter"
       template="OPERATION_RETURN_TYPE"
       sourcerestriction="Foundation.Core.Parameter.kind;xmi
       .value;return">
       <RETURN_DEFAULT_TYPE type="replace"
          link="Foundation.Core.Classifier;xmi.idref"
          source="Foundation.Core.ModelElement.name"
          map="TYPEMAPS;MAP" />
    </RETURN_DEFAULT_VALUE>
    <OPERATION_NAME type="replace" link=""
       source="Foundation.Core.ModelElement.name"
       target="%OPERATION_NAME%" />
     <OPERATION_INPUT type="multi"
       source="Foundation.Core.Parameter"
       template="OPERATION_INPUT" separator=","
       sourcerestriction="Foundation.Core.Parameter.kind;xmi
       .value;inout">
       <OPERATION_INPUT_TYPE type="replace"
          link="Foundation.Core.Classifier;xmi.idref"
```

```
            source="Foundation.Core.ModelElement.name"
            target="%OPERATION_INPUT_TYPE%" />
          <OPERATION_INPUT_NAME type="replace" link=""
            source="Foundation.Core.ModelElement.name"
            target="%OPERATION_INPUT_NAME%" />
        </OPERATION_INPUT>
      </OPERATION>
      <INTERFACE_OPERATION type="multi"
        source="Foundation.Core.Operation"
        template="INTERFACE_OPERATION_TEMPLATE"
        mandatory="false">
        <OPERATION_VISIBILITY type="replace" link=""
          source="Foundation.Core.ModelElement.visibility;xmi.
          value" target="%OPERATION_VISIBILTY%" />
        <OPERATION_TYPE type="multi"
          source="Foundation.Core.Parameter"
          template="OPERATION_TYPE"
          sourcerestriction="Foundation.Core.Parameter.kind;xmi
          .value;return">
          <RETURN_TYPE type="replace"
            link="Foundation.Core.Classifier;xmi.idref"
            source="Foundation.Core.ModelElement.name"
            map="TYPEMAPS;MAP" />
        </OPERATION_TYPE>
        <OPERATION_NAME type="replace" link=""
          source="Foundation.Core.ModelElement.name"
          target="%OPERATION_NAME%" />
        <OPERATION_INPUT type="multi"
          source="Foundation.Core.Parameter"
          template="OPERATION_INPUT" separator=","
          sourcerestriction="Foundation.Core.Parameter.kind;xmi
          .value;inout">
          <OPERATION_INPUT_TYPE type="replace"
            link="Foundation.Core.Classifier;xmi.idref"
            source="Foundation.Core.ModelElement.name"
            target="%OPERATION_INPUT_TYPE%" />
          <OPERATION_INPUT_NAME type="replace" link=""
            source="Foundation.Core.ModelElement.name"
            target="%OPERATION_INPUT_NAME%" />
        </OPERATION_INPUT>
      </INTERFACE_OPERATION>
      <CONTAINER_MANAGED_FIELDS type="multi"
        source="Foundation.Core.Attribute"
        template="CONTAINER_MANAGED_FIELDS_TEMPLATE"
        mandatory="false">
        <ATTRIBUTE_VISIBILITY type="replace" link=""
          source="Foundation.Core.ModelElement.visibility;xmi.
          value" />
        <ATTRIBUTE_TYPE type="replace"
          link="Foundation.Core.Classifier;xmi.idref"
          source="Foundation.Core.ModelElement.name" />
```

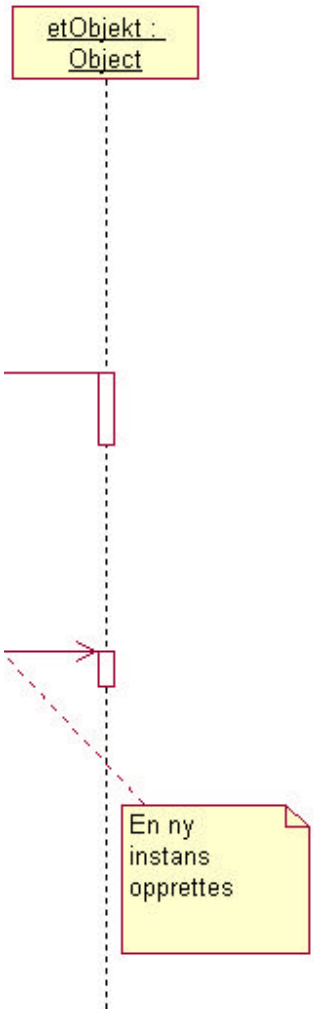
```
        <ATTRIBUTE_NAME type="replace" link=""
          source="Foundation.Core.ModelElement.name" />
      </CONTAINER_MANAGED_FIELDS>
      <EJB_CREATE type="multi"
        source="Foundation.Core.Attribute"
        template="EJB_CREATE" mandatory="false">
        <ATTRIBUTE_VISIBILITY type="replace" link=""
          source="Foundation.Core.ModelElement.visibility;xmi.
          value" />
        <ATTRIBUTE_TYPE type="replace"
          link="Foundation.Core.Classifier;xmi.idref"
          source="Foundation.Core.ModelElement.name" />
        <ATTRIBUTE_NAME type="replace" link=""
          source="Foundation.Core.ModelElement.name" />
      </EJB_CREATE>
      <TYPEMAPS>
        <MAP old="boolean" new="true" />
        <MAP old="int" new="0" />
      </TYPEMAPS>
    </RULES>
</RULEFILE>
```

14.3 Vedlegg 3 – Tegnforklaringer til systembeskrivelser

Sekvensdiagram

Et sekvensdiagram viser tidsrekkefølgen for meldinger mellom objekter, og dermed hvordan de samarbeider. Et sekvensdiagram beskriver dermed flyt i systemet for et gitt scenario.

Objekter
Objektene i et sekvensdiagram står som regel øverst i diagrammet. Et objekt kan være en uspesifisert aktør, en modul , en klasse eller noe annet, så lenge den spiller en er rolle i sekvensen som skal beskrives. Symbolet for et objekt kan derfor variere litt etter hvilke type objekt det er snakk om, men generelt er symbolet som vist her. Se derfor symbolforklaringen for ulike klasser under klassediagram for beskrivelser av andre symboler som måtte dukke opp. Legg merke til navngivingen av et objekt som *navn:implementasjon*



Tidslinje
Tiden i et sekvensdiagram går nedover, og dette illustrer med den stiplede linjen som går under ned fra vær instans av et objekt.

Instans
Levetiden til en instans av objektet er markert som en boks på tidslinjen. Dette gjør sekvensdiagrammet i stand til å beskriver bruken av mange ulike instanser av samme objekt spredt i tid.

Melding (a.k.a. kall)
En melding eller kall som det også kalles, går mellom to instanser av objekt og er symbolisert som en pil fra den som utfører kallet til den som mottar kallet. En slik melding kan opprette en ny instans av et objekt eller den gå til en allerede eksisterende instans.

Returverdi
Returverdier er ikke alltid tatt med i sekvensdiagrammer da de regnes som en naturlig del av kallet, men når de taes med beskrives de som en stiplet pil tilbake fra den kallet instansen til den kallende instansen med en beskrivelse av hva som returneres.

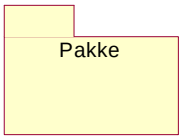
Kommentarer
Kommentarer markers som en 'innsatt side' i diagrammet, og er som regel knyttet til et kall ved hjelp av en stiplet linje. Kommentarer prøver som regel å forklare noe som ikke er mulig representere i et sekvens diagram, som for eksempel repeterende kall osv.

Klassediagram

Et klassediagram viser pakkene, klassene og grensesnittene et system består av, samt relasjonene og samspillet mellom dem. Et klassediagram beskriver dermed egenskapene til og sammenhengene mellom de ulike delene av systemet.

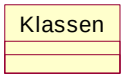
Pakke (a.k.a. modul)

En pakke eller modul modelleres i et klassediagram med et symbol som ser ut som en av mappene i Windows. En pakke symboliserer er større del av et system og kan inneholde flere klasser og liknende.



Klasse

En klasse modelleres med mange ulike symbol avhengig av rollen til klassen og granulariteten til klassen. Generelt er symbolet til en klasse en firkant inndelt i tre seksjoner. Øverst kommer navnet til klassen og eventuell annen overordnet informasjon, så følger attributtene, og nederst eventuelle metoder klassen har.



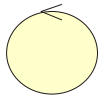
Det finnes mange ulike spesialtyper av klasser, og hver enkelt av disse har egne symboler. De mest vanlige typene er:

Grensesnitt

Et grensesnitt er en definisjon på hva en klasse må inneholde. Slike grensesnitt brukes ofte for å definere de tjenester en modul eller pakke skal tilby og gjøre den indre strukturen i pakken eller modulen transparent for brukeren av grensesnittet. Symbolet for et grensesnitt er en liten sirkel som illustrer at dette kun er definisjonen på en klasse, ikke selve implementasjonen.

Kontrollklasse

En kontrollklasse er en klasse som inneholder kontrollflyt i programmet. Her finner vi som regel alle typer logikk klasser, for eksempel foretningslogikk, og eventuelle 'gudeklasser'. Symbolet for en kontroll klasse er en sirkel med en pil som illustrerer flyten av kontroll.

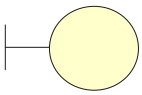


Entitetsklasse

En entitetsklasse er en klasse som representerer et dataobjekt i systemet. Dette er klasser som kun inneholder data, og modifiserer sitt dataobjekt på en eller flere måter. Symbolet for en entitetsklasse er et 'omega' tegn som symboliserer at dette objektet representer det laveste nivået i systemet

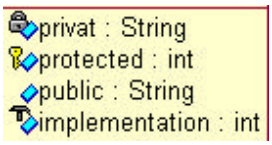
Grenseklasse

En grenseklasse symboliserer at den er en grense i systemet. En slik grense kan være for eksempel ut mot bruker, mot andre systemer eller mot andre pakker/moduler. Symbolet for en grenseklasse er en sirkel med en strek ut til en tegnet grenselinje.



Attributter

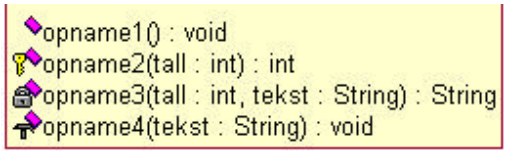
En attributt hører alltid til en klasse eller et grensesnitt, og kan ha forskjellige egenskaper.



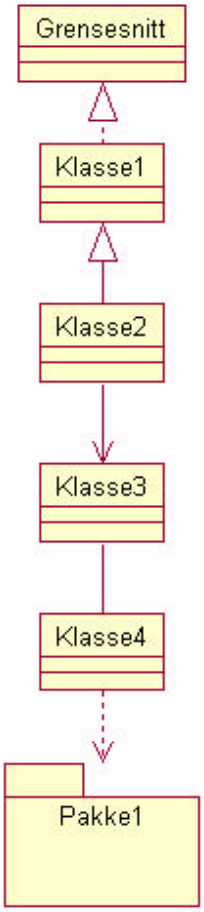
Den kan være privat, protected, public eller implementation i Java, og tilsvarende i de fleste andre språk. Alle attributter har et lys blått rektangel som grunnsymbol, i tillegg illustreres de ulike typen med ulike tillegg til ikonet , en hengelås for private, en nøkkel for protected, og en hammer for implementation. Legg for øvrig merke til skrive måten for en attributt som er *navn : type*

Metoder

En metode hører alltid til en klasse eller grensesnitt, og har mange ulike egenskaper. En metode har samme synlighetstyper som en attributt, men har i tillegg parameter med typer og en retur type. Alle metoder har et rosa rektangel som grunnsymbol, og har de samme tilleggsikonene for å illustrere synlighet som attributtene. Legg også merke til skrivemåten for en metode som er *metodenavn(parameternavn : parameterType, …) : returtype*



Relasjon



En relasjon beskriver en sammenheng mellom to objekter i et klasse diagram. Alle relasjoner kan ha spesifiserte kardinaliteter, men dette er ikke alltid tilfelle. Det finnes mange ulike typer relasjoner, men de vanligste er:

Implementering av et grensesnitt

Implementering symboliseres ved en hel pilspiss på en stiplet strek.

Arv

Arv symboliserer med en hel pilspiss på en helltrukket strek

En veis navigerbar relasjon

En slik relasjon illustrerer det faktum at det er kun mulig å kommunisere fra det ene objektet til det andre og ikke motsatt. Symbolet for dette er en relasjonsstrek med en pil i ene enden som viser hvilke veg denne relasjonen er navigerbar.

To veis navigerbar relasjon

Begge objektene har en referanse til hverandre slik at kommunikasjonen kan gå begge veger. Dette illustreres ved at relasjonen ikke har noen pil

Instansiering eller avhengighet

En slik relasjon symboliserer at det ikke finnes noe fast relasjon, men det kan forekomme bruk, initialisering eller avhengighet mellom objektene. Dette symboliseres med en stiplet linje med en pil som viser retningen fra den som er aktiv til den som blir brukt/opprettet.

14.4 Vedlegg 4 – Bevis av konsept

INNDATA

Prosjektnavn: test

Datamodell:

```
<ENTITIES>
 <ENTITY>
       <NAME>Person</NAME>
       <ATTRIBUTE>
             <NAME>Personnummer</NAME>
             <TYPE>int</TYPE>
       </ATTRIBUTE>
        <ATTRIBUTE>
             <NAME>Navn</NAME>
             <TYPE>varchar</TYPE>
       </ATTRIBUTE>
       <ATTRIBUTE>
             <NAME>Telefon</NAME>
             <TYPE>number</TYPE>
       </ATTRIBUTE>
        <METHOD>
            <NAME>getPerson</NAME>
            <RETURNVALUE>Person</RETURNVALUE>
            <PARAM>Personnummer</PARAM>
        </METHOD>

 </ENTITY>
 <ENTITY>
       <NAME>BIL</NAME>
       <ATTRIBUTE>
             <NAME>Merke</NAME>
             <TYPE>varchar</TYPE>
       </ATTRIBUTE>
       <ATTRIBUTE>
             <NAME>Modell</NAME>
             <TYPE>number</TYPE>
       </ATTRIBUTE>
       <ATTRIBUTE>
             <NAME>Personnr</NAME>
             <TYPE>number</TYPE>
       </ATTRIBUTE>
        <RELATION>
             <FIELD>Personnr</FIELD>
            <ENTITY>Person</ENTITY>
            <CARDINALITY1>1</CARDINALITY1>
             <CARDINALITY2>2</CARDINALITY2>
       </RELATION>
 </ENTITY>
</ENTITIES>
```

REGELFIL

```
<rules>EJB Architecture
     <control><code>%name%.java</code><sql>%projectName%.sql</sql>
     </control>
    <entity>
        <start><code> **Generated as a part of the proof of concept made
             by QPROSYS*/</code></start>
        <name><code>public class %name% {</code><sql>create table
            %name%(</sql></name>
        <attribute><code>%type% %name%;</code><sql>%name%
            %type%,</sql></attribute>
        <method><code>public %returnvalue%
            %name%(%param%){}</code></method>
        <relation><code>public %entity% f_key;</code><sql>constraint
            f_key %field% references %entity%</sql></relation>
        <end><code>}</code><sql>);</sql></end>
    </entity>
</rules>
```

UTDATA

BIL.java

```
**Generated as a part of the proof of concept made by QPROSYS*/
public class BIL {
varchar Merke;
number Modell;
number Personnr;
public Person f_key;
}
```

Person.java

```
**Generated as a part of the proof of concept made by QPROSYS*/
public class Person {
int Personnummer;
varchar Navn;
number Telefon;
public Person getPerson(Personnummer){}
}
```

test.sql

```
create table Person(
Personnummer int,
Navn varchar,
Telefon number,
);
create table BIL(
Merke varchar,
Modell number,
Personnr number,
```

```
constraint f_key Personnr references Person
);
```

test.log (automatisk generert logfil)

```
***
CodeBuilder startet with following XMI tree:
***
ENTITIES-NODE
       #text-NODE with value: []
       ENTITY-NODE  has 0 atributes
              #text-NODE with value: []
              NAME-NODE  has 0 atributes
                     #text-NODE with value: [Person]
              #text-NODE with value: []
              ATTRIBUTE-NODE  has 0 atributes
                     #text-NODE with value: []
                     NAME-NODE  has 0 atributes
                            #text-NODE with value: [Personnummer]
                     #text-NODE with value: []
                     TYPE-NODE  has 0 atributes
                            #text-NODE with value: [int]
                     #text-NODE with value: []
              #text-NODE with value: []
              ATTRIBUTE-NODE  has 0 atributes
                     #text-NODE with value: []
                     NAME-NODE  has 0 atributes
                            #text-NODE with value: [Navn]
                     #text-NODE with value: []
                     TYPE-NODE  has 0 atributes
                            #text-NODE with value: [varchar]
                     #text-NODE with value: []
              #text-NODE with value: []
              ATTRIBUTE-NODE  has 0 atributes
                     #text-NODE with value: []
                     NAME-NODE  has 0 atributes
                            #text-NODE with value: [Telefon]
                     #text-NODE with value: []
                     TYPE-NODE  has 0 atributes
                            #text-NODE with value: [number]
                     #text-NODE with value: []
              #text-NODE with value: []
              METHOD-NODE  has 0 atributes
                     #text-NODE with value: []
                     NAME-NODE  has 0 atributes
                            #text-NODE with value: [getPerson]
                     #text-NODE with value: []
                     RETURNVALUE-NODE  has 0 atributes
                            #text-NODE with value: [Person]
                     #text-NODE with value: []
                     PARAM-NODE  has 0 atributes
                            #text-NODE with value: [Personnummer]
                     #text-NODE with value: []
              #text-NODE with value: []
       #text-NODE with value: []
       ENTITY-NODE  has 0 atributes
              #text-NODE with value: []
              NAME-NODE  has 0 atributes
                     #text-NODE with value: [BIL]
              #text-NODE with value: []
              ATTRIBUTE-NODE  has 0 atributes
                     #text-NODE with value: []
                     NAME-NODE  has 0 atributes
                            #text-NODE with value: [Merke]
                     #text-NODE with value: []
```

```
                    TYPE-NODE  has 0 atributes
                          #text-NODE with value: [varchar]
                    #text-NODE with value: []
             #text-NODE with value: []
             ATTRIBUTE-NODE  has 0 atributes
                    #text-NODE with value: []
                    NAME-NODE  has 0 atributes
                          #text-NODE with value: [Modell]
                    #text-NODE with value: []
                    TYPE-NODE  has 0 atributes
                          #text-NODE with value: [number]
                    #text-NODE with value: []
             #text-NODE with value: []
             ATTRIBUTE-NODE  has 0 atributes
                    #text-NODE with value: []
                    NAME-NODE  has 0 atributes
                          #text-NODE with value: [Personnr]
                    #text-NODE with value: []
                    TYPE-NODE  has 0 atributes
                          #text-NODE with value: [number]
                    #text-NODE with value: []
             #text-NODE with value: []
             RELATION-NODE  has 0 atributes
                    #text-NODE with value: []
                    FIELD-NODE  has 0 atributes
                          #text-NODE with value: [Personnr]
                    #text-NODE with value: []
                    ENTITY-NODE  has 0 atributes
                          #text-NODE with value: [Person]
                    #text-NODE with value: []
                    CARDINALITY1-NODE  has 0 atributes
                          #text-NODE with value: [1]
                    #text-NODE with value: []
                    CARDINALITY2-NODE  has 0 atributes
                          #text-NODE with value: [2]
                    #text-NODE with value: []
             #text-NODE with value: []
      #text-NODE with value: []
***
CodeBuilder startet with following Rules tree:
***
rules-NODE
      #text-NODE with value: [EJB Architecture]
      control-NODE  has 0 atributes
             code-NODE  has 0 atributes
                    #text-NODE with value: [%name%.java]
             sql-NODE  has 0 atributes
                    #text-NODE with value: [%projectName%.sql]
      #text-NODE with value: []
      entity-NODE  has 0 atributes
             #text-NODE with value: []
             start-NODE  has 0 atributes
                    code-NODE  has 0 atributes
                          #text-NODE with value: [**Generated as a part of the
proof of concept made by QPROSYS*/]
             #text-NODE with value: []
             name-NODE  has 0 atributes
                    code-NODE  has 0 atributes
                          #text-NODE with value: [public class %name% {]
                    sql-NODE  has 0 atributes
                          #text-NODE with value: [create table %name%(]
             #text-NODE with value: []
             attribute-NODE  has 0 atributes
```

```
                  code-NODE  has 0 atributes
                        #text-NODE with value: [%type% %name%;]
                  sql-NODE  has 0 atributes
                        #text-NODE with value: [%name% %type%,]
            #text-NODE with value: []
            method-NODE  has 0 atributes
                  code-NODE  has 0 atributes
                        #text-NODE with value: [public %returnvalue%
%name%(%param%){}]
            #text-NODE with value: []
            relation-NODE  has 0 atributes
                  code-NODE  has 0 atributes
                        #text-NODE with value: [public %entity% f_key;]
                  sql-NODE  has 0 atributes
                        #text-NODE with value: [constraint f_key %field%
references %entity%]
            #text-NODE with value: []
            end-NODE  has 0 atributes
                  code-NODE  has 0 atributes
                        #text-NODE with value: [}]
                  sql-NODE  has 0 atributes
                        #text-NODE with value: [);]
            #text-NODE with value: []
      #text-NODE with value: []

****Made CODE in Person.java
 **Generated as a part of the proof of concept made by QPROSYS*/
public class Person {
int Personnummer;
varchar Navn;
number Telefon;
public Person getPerson(Personnummer){}
}

****Made SQL in:test.sql
create table Person(
Personnummer int,
Navn varchar,
Telefon number,
);

****Made CODE in BIL.java
 **Generated as a part of the proof of concept made by QPROSYS*/
public class BIL {
varchar Merke;
number Modell;
number Personnr;
public Person f_key;
}

****Made SQL in:test.sql
create table BIL(
Merke varchar,
Modell number,
Personnr number,
constraint f_key Personnr references Person
);
```

Kildekode

POCStart.java

```
package POC;

class POCStart {

    public static void main(String[] args) {
        String xmifile,rulefile,src,projectname;
        System.out.println("***** Starting Proof of Concept in the
Qpro2001 project *****");
        if (args.length < 4) {
            System.out.println("Wrong usage of program!! Correct
usage:\tjava POCStart [xmi-file] [rule-file] [src-dir] [projectname]");
            System.exit(1);
        }
        xmifile = args[0];
        rulefile = args[1];
        src = args[2];
        projectname= args[3];

        System.out.println(">See [src-dir]/[project-name].log for
further details\n>Validating rules ...");
        POCRules theRules= new POCRules(rulefile,projectname,src);
        System.out.println(">Rules validated\n>Initiating
Codebuilding");
        POCCodeBuilder theBuilder = new POCCodeBuilder(projectname,src);
        System.out.println(">Codebuilding ready\n>Starting
Codebuilding");
        theBuilder.generate(POCParser.parse(xmifile),theRules);
        System.out.println(">Codebuilding complete");
    }
}
```

POCParser.java

```
package POC;

import java.io.*;

import com.sun.xml.tree.*;
import com.sun.xml.parser.*;
import org.w3c.dom.*;
import org.xml.sax.*;

/**
A proof of Concept of the Parser class...
 @author Martin Sleire Vatne
 */

public class POCParser {

    public static XmlDocument parse(String pathToXMLFile){
        File ruleFile = new File(pathToXMLFile);
        try {
            InputSource src = Resolver.createInputSource(ruleFile);
```

```
            return XmlDocument.createXmlDocument(src, !true);
            //System.out.println(writeTree(xml.getDocumentElement(),
0).toString());
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return null;
        }
    }
}
```

POCRules.java

```
package POC;

import com.sun.xml.tree.XmlDocument;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.util.StringTokenizer;

/**
A proof of Concept on the Rules file...
This class represent the object impelementation of the corresponding
rules file written i XML.
Generates code and an sqlscript from an XMI model acording to the rules
found in the given rules file.
 @author Martin Sleire Vatne
 */

public class POCRules {

    private XmlDocument xml = null;
    private Element xmlRoot = null;
    private StringBuffer tree = null;
    private String projectName = null;
    private String srcPath = null;

    /** Creates the rulesobject related to the rules file*/
    public POCRules(String pathToRuleFile, String projectName, String
srcPath) {
        this.projectName = projectName;
        this.srcPath = srcPath;
        xml = POCParser.parse(pathToRuleFile);
        xmlRoot = xml.getDocumentElement();
    }

    /**Returns an StringBuffer representation of the parsed Rules tree*/
    public StringBuffer getRulesTree() {
        writeTree(xmlRoot, 0);
        return tree;
    }
```

```
    /**Writes code based on the rules acording to an Entity in the XMI
model
     * @param e the "entity" element with its corresponding subtree of
the XMI modell
     * @return String containing information regarding the codecreation
process.*/
    public String makeEntityCode(Element e) {
        //defining
        String current_tag,codeFile,sqlFile;
        StringBuffer code, sql; //using buffers to contain the code!
        Element current_element;

        //Initiating
        code = new StringBuffer();
        sql = new StringBuffer();
        current_element = null;

        code.append(applyRule("start", true, e));
        sql.append(applyRule("start", false, e));

        NodeList nl = e.getChildNodes(); //getting all the children

        for (int i = 0; i < nl.getLength(); i++) {
            if (nl.item(i).hasChildNodes()) {
                current_tag = nl.item(i).getNodeName(); //gets the tag
                current_element = (Element) nl.item(i); //gets element

                code.append(applyRule(current_tag, true,
                current_element));
                sql.append(applyRule(current_tag, false,
                current_element));
            }
        }

        code.append(applyRule("end", true, e));//applying  end rules
        sql.append(applyRule("end", false, e));

        codeFile = applyRule("control", true, e).toString(); //control
        sqlFile = projectName + ".sql"; //Cheatin abit..

        writeFile(code, true, codeFile); //writes the codefile
        writeFile(sql, false, sqlFile); //appends the sqlfile

        return "\n****Made CODE in " + codeFile + "\n" + code +
               "\n****Made SQL in:" + sqlFile + "\n" + sql;
    }

    /**Applies a rule from the rulefile corresponding to the tag on the
element in the XMI model
     * @param tag containing the tag of element e
     * @param code a boolean true = this is code generation, false =
this is sql generation
     * @param e the element in the XMI model which the rule shall be
applied on
     * @return StringBuffer containing the resulting code after the rule
has been applied**/
```

```
    private StringBuffer applyRule(String tag, boolean code, Element e)
{
        //defining
        StringBuffer sb;
        String temp,mal;
        StringTokenizer st;
        boolean intoken;

        //initiating
        intoken = false; //starts outside an token.
        sb = new StringBuffer();

        mal = getRule(tag, code); //gets the rule that matches this tag.
        if (mal == null)
            return sb;   //no rule, nothing to do...

        st = new StringTokenizer(mal, "%", true); //seperates tokens

        while (st.hasMoreTokens()) {
            //runs through all the tokens.
            temp = st.nextToken();
            if (temp.equals("%")) {
                //got a token, change state.
                intoken = !intoken;
            } else if (intoken) {
                //is inside a token, must replace it.
                sb.append(replaceToken(temp, e));
            } else {
                //outside a token --> print away.
                sb.append(temp);
            }
        }
        if (!(tag.equalsIgnoreCase("control")))  //no linebreaks
            sb.append("\n");
        return sb;
    }

    /**Gets the rule corresponding to a given tag and the given
targetcode
     * @param tag containing the tag to be matched
     * @param code boolean, true=code, false=sql
     * @return String containing the rule found, no rule found-->
returning null**/
    private String getRule(String tag, boolean code) {
        NodeList nl = xmlRoot.getElementsByTagName(tag.toLowerCase());

        if (nl.getLength() == 0) return null;   //no match

        Element e = (Element) nl.item(0); //the rules element
        if (code) {
            //only intrested in code-related rules.
            nl = e.getElementsByTagName("code"); //looking for <code>
            if (nl.getLength() != 0)
                return ((Element)
nl.item(0)).getFirstChild().getNodeValue();
            else
                return null; //no <code> related rules in this tag
```

```
        } else {
            //only intressed in sql-related rules.
            nl = e.getElementsByTagName("sql");
            if (nl.getLength() != 0)
                return ((Element)
nl.item(0)).getFirstChild().getNodeValue();
            else
                return null; //no <code> related rules in this tag
        }
    }

    /**Replaces the token with the wanted value from the XMI modell.
     * @param s containing the token to be replaced
     * @param e containing the element from the XMI that is to be
matched
     * @return String containing the value to beused instead of the
token */
    private String replaceToken(String s, Element e) {
        if (e.getNodeName().equalsIgnoreCase(s)) {
            //is the root element the matching node itself ?
            return e.getFirstChild().getNodeValue();
        } else {
            //search the tree of children
            NodeList nl = e.getElementsByTagName(s.toUpperCase());
            if (nl.getLength() == 0) {
                //no match found, the tag is missing in the XMI model
                return "#NOT_FOUND_ON_IN_MODEL#";
            } else {
                //get the value that shall replace the token
                return ((Element)
nl.item(0)).getFirstChild().getNodeValue();
            }
        }
    }

    /**Writes the code generated to a new file, or appends it on to an
existing
     * @param s containing the content of the file
     * @param toNewFile boolean true= this is a new File (and shall
overwrite any previous files with idetical name), false = append the
content if the file already exists.
     * @param fileName a String containing the wanted name of the file*/
    private void writeFile(StringBuffer s, boolean toNewFile, String
fileName) {
        try {
            File aFile = new File(srcPath, fileName);
            if (toNewFile) {
                //go ahead overwrite any existing ones.
                aFile.createNewFile();
                FileWriter fw = new FileWriter(aFile);
                fw.write(s.toString());
                fw.flush();
                fw.close();
            } else {
                //append this content to any already existing file
                if (!aFile.exists())
                    aFile.createNewFile();
```

```
                FileOutputStream fos = new
FileOutputStream(aFile.getAbsolutePath(), true);
                fos.write(s.toString().getBytes());
                fos.close();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private boolean writeTree(Element e, int level) {
        if (e == null) return true;
        //Element n;
        NamedNodeMap nnm;
        NodeList nl, nlT;
        Element eT, eN;
        nl = e.getChildNodes();
        if (level == 0) {
            tree = new StringBuffer();
            tree.append(e.getNodeName() + "-NODE\n");
        }
        for (int i = 0; i < nl.getLength(); i++) {
            for (int j = 0; j <= level; j++) {
                tree.append("\t");
            }
            if (nl.item(i).hasChildNodes()) {
                //har barn
                eT = (Element) nl.item(i);
                tree.append(nl.item(i).getNodeName() + "-NODE ");
                nnm = nl.item(i).getAttributes();
                if (nnm != null) tree.append(" has " + nnm.getLength() +
" atributes");
                tree.append("\n");
                writeTree(eT, level + 1); //setter nåverende som rot
(DFS)

            } else if (nl.item(i).getNodeType() == Node.TEXT_NODE) {
                //Har ikke barn
                if (nl.item(i).getNodeValue().trim() !="\n") {
                    tree.append(nl.item(i).getNodeName() + "-NODE with
value: [" + nl.item(i).getNodeValue().trim() + "]\n");
                } else
                    tree.append(nl.item(i).getNodeName() + "-NODE \n");
            }
        }
        return true;
    }
}
```

POCRules.java

```
package POC;

import com.sun.xml.tree.XmlDocument;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
```

```
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import java.io.File;
import java.io.FileWriter;

/**
A proof of Concept on the CodeBuilder
 @author Martin Sleire Vatne
 */

public class POCCodeBuilder {
    private StringBuffer tree = null;
    private String projectName = null;
    private String srcPath = null;
    private File logfile = null;
    private FileWriter fw = null;

    public POCCodeBuilder(String projectName, String srcPath) {
        this.projectName = projectName;
        this.srcPath = srcPath;
        init();
    }

    public void generate(XmlDocument xmi, POCRules theRules) {
        Element xmiroot,entity;
        xmiroot = xmi.getDocumentElement();
        log("***\nCodeBuilder startet with following XMI tree\n***\n" +
writeTree(xmiroot, 0).toString());
        log("***\nCodeBuilder startet with following Rules tree\n***\n"
+ theRules.getRulesTree().toString());
        NodeList nl = xmiroot.getElementsByTagName("ENTITY");
        for (int i = 0; i < nl.getLength()-1; i++) {
            entity = (Element) nl.item(i);
            log(theRules.makeEntityCode(entity));

        }
    }

    private void init() {
        try {
            File src = new File(srcPath);
            src.mkdir();
            logfile = new File(src, projectName + ".log");
            logfile.createNewFile();
            fw = new FileWriter(logfile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void log(String s) {
        try {
            fw.write(s);
            fw.flush();
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }

    private StringBuffer writeTree(Element e, int level) {
        if (e == null) return null;

        //Element n;
        NamedNodeMap nnm;
        NodeList nl, nlT;
        Element eT, eN;
        e.normalize();
        nl = e.getChildNodes();

        if (level == 0) {
            tree = new StringBuffer();
            tree.append(e.getNodeName() + "NODE\n");
        }

        for (int i = 0; i < nl.getLength(); i++) {
            for (int j = 0; j <= level; j++) {
                tree.append("\t");
            }
            if (nl.item(i).hasChildNodes()) {
                //har barn
                eT = (Element) nl.item(i);
                tree.append(nl.item(i).getNodeName() + "NODE ");
                nnm = nl.item(i).getAttributes();
                if (nnm != null) tree.append(" has " + nnm.getLength() +
" atributes");
                tree.append("\n");
                writeTree(eT, level + 1); //setter nåverende som rot og
parser ned i treet (DFS)

            } else if (nl.item(i).getNodeType() == Node.TEXT_NODE) {
                //Har ikke barn
                if (nl.item(i).getNodeValue().trim() !="\n") {
                    tree.append(nl.item(i).getNodeName() + "NODE with
value: [" + nl.item(i).getNodeValue().trim() + "]\n");
                } else
                    tree.append(nl.item(i).getNodeName() + "NODE \n");
            }
        }

        return tree;
    }
}
```

14.5 Vedlegg 5 – Sjekkpunkter for kodeinspeksjon

Dette vedlegget inneholder en liste med sjekkpunkter som inspektør kan benytte seg av ved kodeinspeksjon. Sjekkpunkter hakes ut etter hvert som de er kontrollert. Sjekklista er hentet fra http://www.cs.jmu.edu/common/coursedocs/cs345/CS345%20Fall%202000/JavaChk.html.

Java Inspection Checklist

1. Variable, Atrtibute, and Constant Declaration Defects (VC)
✍✍Are descriptive variable and constant names used in accord with naming conventions?
✍✍Are there variables or atributes with confusingly similar names?
✍✍Is every variable and attribute correctly typed?
✍✍Is every variable and attribute properly initialized?
✍✍Could any non-local variables be made local?
✍✍Are all for-loop control variables declared in the loop header?
✍✍Are there literal constants that should be named constants?
✍✍Are there variables or attributes that should be constants?
✍✍Are there attributes that should be local variables?
✍✍Do all attributes have appropriate access modifiers (private, protected, public)?
✍✍Are there static attributes that should be non-static or vice-versa?

2. Method Definition Defects (FD)
✍✍Are descriptive method names used in accord with naming conventions?
✍✍Is every method parameter value checked before being used?
✍✍For every method: Does it return the correct value at every method return point?
✍✍Do all methods have appropriate access modifiers (private, protected, public)?
✍✍Are there static methods that should be non-static or vice-versa?

3. Class Definition Defects (CD)
✍✍Does each class have appropriate constructors and destructors?
✍✍Do any subclasses have common members that should be in the superclass?
✍✍Can the class inheritance hierarchy be simplified?

4. Data Reference Defects (DR)
✍✍For every array reference: Is each subscript value within the defined bounds?
✍✍For every object or array reference: Is the value certain to be non-null?

5. Computation/Numeric Defects (CN)
✍✍Are there any computations with mixed data types?
✍✍Is overflow or underflow possible during a computation?
✍✍For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
✍✍Are parentheses used to avoid ambiguity?

6. Comparison/Relational Defects (CR)
✍✍For every boolean test: Is the correct condition checked?
✍✍Are the comparison operators correct?
✍✍Has each boolean expression been simplified by driving negations inward?
✍✍Is each boolean expression correct?
✍✍Are there improper and unnoticed side-effects of a comparison?
✍✍Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?

7. Control Flow Defects (CF)
✍✍For each loop: Is the best choice of looping constructs used?
✍✍Will all loops terminate?
✍✍When there are multiple exits from a loop, is each exit necessary and handled properly?
✍✍Does each switch statement have a default case?
✍✍Are missing switch case break statements correct and marked with a comment?
✍✍Do named break statements send control to the right place?
✍✍Is the nesting of loops and branches too deep, and is it correct?
✍✍Can any nested if statements be converted into a switch statement?
✍✍Are null bodied control structures correct and marked with braces or comments?
✍✍Are all exceptions handled appropriately?
✍✍Does every method terminate?

8. Input-Output Defects (IO)
✍✍Have all files been opened before use?
✍✍Are the attributes of the input object consistent with the use of the file?
✍✍Have all files been closed after use?
✍✍Are there spelling or grammatical errors in any text printed or displayed?
✍✍Are all I/O exceptions handled in a reasonable way?

9. Module Interface Defects (MI)
✍✍Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?
✍✍Do the values in units agree (e.g., inches versus yards)?
✍✍If an object or array is passed, does it get changed, and changed correctly by the called method?

10. Comment Defects (CM)
✍✍Does every method, class, and file have an appropriate header comment?
✍✍Does every atribute, variable, and constant declaration have a comment?
✍✍Is the underlying behavior of each method and class expressed in plain language?
✍✍Is the header comment for each method and class consistent with the behavior of the method or class?
✍✍Do the comments and code agree?
✍✍Do the comments help in understanding the code?
✍✍Are there enough comments in the code?
✍✍Are there too many comments in the code?

11. Layout and Packaging Defects (LP)
✍✍Is a standard indentation and layout format used consistently?
✍✍For each method: Is it no more than about 60 lines long?
✍✍For each compile module: Is no more than about 600 lines long?

12. Modularity Defects (MO)
✍✍Is there a low level of coupling between modules (methods and classes)?
✍✍Is there a high level of cohesion within each module (methods or class)?
✍✍Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?
✍✍Are the Java class libraries used where and when appropriate?

13. Storage Usage Defects (SU)
✍✍Are arrays large enough?
✍✍Are object and array references set to null once the object or array is no longer needed?

14. Performance Defects (PE) [Optional]
✍✍Can better data structures or more efficient algorithms be used?
✍✍Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
✍✍Can the cost of recomputing a value be reduced by computing it once and storing the results?
✍✍Is every result that is computed and stored actually used?
✍✍Can a computation be moved outside a loop?
✍✍Are there tests within a loop that do not need to be done?
✍✍Can a short loop be unrolled?
✍✍Are there two loops operating on the same data that can be combined into one?
✍✍Are frequently used variables declared register?
✍✍Are short and commonly called methods declared inline?

Kundestyrt prosjekt

Høst 2001

Implementasjon

Versjon 1.02

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
25.10.2001	Dokumentet opprettet	0.01	Magnus
26.10.2001	Førsteutkast til innhold	0.02	Martin
04.11.2001	Versjonskontroll	0.10	Atle
06.11.2001	La til kapittel om utvidelser	0.12	Magnus
07.11.2001	Modultest	0.20	Atle
07.11.2001	Stikkord kap 3 og 4 + layout	0.21	Ole Kristian
09.11.2001	Feil og mangler i generatoren	0.22	Ole Kristian
10.11.2001	Kodeinspeksjon og integrasjonstesting	0.30	Atle
11.11.2001	Kapittel 1	0.36	Martin
11.11.2001	Kapittel 4 fullført	0.40	Ole Kristian
12.11.2001	Dokument ferdig	1.00	Magnus
13.11.2001	Fikset layout	1.01	Ole Kristian
13.11.2001	Korrekturlesing	1.02	Atle

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dette dokumentet er et av resultatene fra implementasjonsfasen, og målet til dette dokumentet er å vise og forklare de valg og endringer som ble gjort i implementasjonsfasen.

1.2 Avgrensning

Dette dokumentet omhandler kun selve implementasjonsprosessen av det systemet som ble designet i Konstruksjonsdokumentet [KON].

1.3 Spesielle definisjoner

Se glossar [GLO]

1.4 Dokumentreferanser

[KON] Konstruksjonsdokumentet, Qpro16 2001
[GLO] Glossar, Qpro16 2001

Alle eksterne referanser er oppgitt i kapittel 9, Litteraturliste

1.5 Dokumentoversikt

Dette dokumentet starter med å forklare de valg som ble gjort i implementasjonsfasen i kapittel 2. Selve arbeidsprosssesen i fasen beskrives i kapittel 3. Deretter forklares alle endringer som måtte gjøres fra den opprinnelige konstruksjonen og den implementerte konstruksjonen beskrives i kapittel 4.

Kapittel 5 tar for seg de kjente feil og manglene i resultatet av implementasjonen og kapittel 6 forklarer hvordan det skal arbeides videre med den implementerte koden. Testingen gjennomført i implementasjonsfasen er dokumentert i kapittel 7 og 8 med modul- og integrasjontester med resultat.

2 Viktige valg

2.1 Kodestandard

Det ble besluttet at Sun sin konvensjon for javakode skal benyttes i forbindelse med implementasjonen. Dette gjelder både selve koden, kommenteringen og dokumentasjonen(javadoc). Det er et faktum at rundt 80% av total livskostnad for programvare går til vedlikehold, samt at det sjelden er de samme som utvikler og vedlikeholder. Det er derfor viktig å bruke en kodekonvensjon som gjør at det blir enkelt å lese og sette seg inn i koden i ettertid. Konvensjonen finnes i sin helhet i vedlegg 1.

2.2 Utviklingsverktøy

Som utviklingsverktøy for kode ble IDEA 2.0 fra IntelliJ benyttet. Alle prosjektmedlemmene stod fritt til å bruke den editoren de ønsket, men den nevnte ble valgt av alle. Dette er et lite brukt, men svært god verktøy for javakoding. Det støtter både koding, kompilering, kjøring og feilfinning. Programmet støtter også editering av XML filer, som vil bli brukt en del i forbindelse med utarbeiding av regelfiler. Modellen for systemet ble laget i Rose, og brukt for generering av et kodeskjelett for systemet. Dette skjelettet ble importert til IDEA og implementert ferdig der. Informasjon om programmet kan finnes på [IJ1].

2.3 Versjonskontroll

Det er blitt bestemt å bruke CVS versjonskontroll i implementasjonsfasen. CVS ble valgt fordi det er et mye brukt versjonskontrollsystem. CVS kan kjøre skript når filer sjekkes inn og ut, for eksempel for å sende mail til alle i prosjektgruppa om endringer. CVS tillater utviklere å editere samme fil samtidig.

Alle medlemmene i gruppa har fått sin egen brukerkonto på prosjektmaskinen, og blitt tilordnet en brukergruppe som er kalt kpro. Dette gjør at alle medlemmene i gruppa kan lese og skrive til filer i kpro16 gruppekatalogen. Det ble opprettet et såkalt ”CVS repository” på prosjektkatalogen til kpro16 og kodeskjelettet som var laget med Rational Rose, samt regler og maler, ble lagt inn som en modul ”source”. For nærmere informasjon om de vanligste CVS kommandoer henvises det til vedlegg 2.

3 Arbeidet i implementasjonsfasen

Alle de interne modulene som var konstruert ble implementert fullt ut. Arbeidet i fasen var særdeles effektivt, og fikk en ”flying start” da skjelettkoden for alle klassen ble generert fra konstruksjonen av Rational Rose. Ansvaret for de uilke modulene ble fordelt utover gruppen, og selve kodingen foregikk i stor grad parallelt.

Totalt ble det produsert i underkant av 3000 kodelinjer i denne fasen, og mye tid har i tillegg gått med til testing og integrering av koden.

En kort beskrivelse av de ulike modulene og arbeidsmengden i hver enkelt modul er gitt under. For mer detaljert beskrivelse henvises det til konstruksjonsdokumentet [KON].

Modulnavn:	**controller**
Antall klasser:	Fem
Beskrivelse	Den sentrale pakken i systemet, inneholder det meste av kontrollflyten i systemet.
Arbeidsmengde:	Stor, og det vanskeligste i denne pakken er den forholdsvis kompliserte kodegenereringen.

Modulnavn:	**xmlparser**
Antall klasser:	To (herav 1 grensesnitt)
Beskrivelse	Inneholder all parsing av XML dokumenter i systemet
Arbeidsmengde:	Liten, tar i stor grad i bruk ferdige metoder i tredjepartsmodulene fra org og com.

Modulnavn:	**param**
Antall klasser:	Seks (herav 1 grensesnitt)
Beskrivelse	Inneholder alle interne objektrepresentasjoner av parametrene til systemet. Tilbyr et knippe tjenester til kodegenereringen definert i grensesnittet.
Arbeidsmengde:	Stor, vanskelig å få til den interne objektrepresentasjonen til å kunne tilby alle de tjenestene som opprinnelig var konstruert.

Modulnavn:	**GUI**
Antall klasser:	Tre (herav 1 grensesnitt)
Beskrivelse	En pakke som inneholder alt brukergrensesnittet til systemet, var ikke konstruert i detalj, og er derfor konstruert i denne fasen.
Arbeidsmengde:	Liten, og forholdsvis nedprioritert da det ikke eksisterer krav mot brukergrensesnittet.

Modulnavn:	**error**
Antall klasser:	Fem
Beskrivelse	Systemets egne feilklasser.
Arbeidsmengde:	Særdeles liten, bruker kun ferdige metoder i java.lang.Exception

En stor og tidkrevende del av implementasjonen var å integrere de ulike delene av systemet. Selv om konstruksjonen var fulgt, så gikk ikke dette helt smertefritt. Det var også under dette arbeidet at mange svakhetene ved konstruksjonen ble oppdaget. En del av disse førte til at konstruksjonen ble endret (se kapittel 4, Endringer fra konstruksjonen), mens andre ble valgt å stå uløste (se kapittel 5, Kjente feil og mangler).

I tillegg til selve implementasjonen av systemet gikk en god del av arbeidet i fasen med til å få regelfilen og malfilene for EJB til å generere gyldig kode og databaseskript.

4 Endringer fra konstruksjonen

I konstruksjonsfasen ble det lagt ned mye tid i å få laget en så grundig konstruksjon som mulig. Hele gruppen jobbet samlet i en strukturert prosess for å forsøke å få belyst flest mulig aspekter ved systemet. I tillegg ble det under konstruksjonsfasen laget et bevis av konsept (se [KON] kapittel 9, Bevis av konsept) der mange av de potensielle problemene ble avdekket.

Til tross for at konstruksjonen ble så grundig gjennomført oppstod det et par små problemer under implementasjonen. Dette førte til at den opprinnelige konstruksjonen måtte utvides noe. De endringene som er gjort er i pakken 'param'.

Lagt til metoden reset() i klassen 'Template'

Hva Lagt til metoden reset() i 'Template'-klassen for å nullstille en mal slik at den kan brukes flere ganger.

Hvorfor Når malen løpes gjennom blir det husket hvor langt den er kommet. Slik det opprinnelig ble konstruert ville det se ut som gjennomløpingen var på slutten av malen når det ble startet på en ny hvis malen hadde blitt gått gjennom tidligere. Derfor ble det nødvendig med en metode for å nullstille malen slik at alt ble satt tilbake til de opprinnelige verdiene og malen kunne benyttes på nytt.

Byttet ut metoden getXMIRootElement()

Hva Metoden getXMIRootElement() i 'Rules'-klassen ble byttet ut med metoden getXMIDocument() og returnerer følgelig en com.sun.xml.tree.XmlDocument i stedet for en org.w3c.dom.Element.

Hvorfor XmlDocument-objektet har bedre muligheter for å søke etter noder i det parsede XMI-treet enn det Element-objektet har. Det ble derfor vurdert som hensiktsmessig å benytte seg av dette i stedet.

Disse endringene er så lite omfattende at det blir vurdert som lite nyttig å legge ved oppdatert klassediagram for den aktuelle pakken.

5 Kjente feil og mangler

Dette kapittelet omhandler kjente feil og mangler ved det implementerte systemet. Det som er implementert er først og fremst ment som en demoversjon for å vise at det som er konstruert fungerer i praksis. Likevel er det en del feil og mangler som det er verdt å påpeke.

5.1 Feil og mangler i generatoren

Metoden getTarget

Beskrivelse Metoden 'getTarget' i klassen 'Template' som skal returnere navnet på filen som skal opprettes har sine begrensninger. Metoden tar som innparametere variabelnavnet og verdien som variabelen skal byttes ut med. Slik denne er implementert nå kan filnavnet kun inneholde en variabel. Hvis det er flere variabler, og det blir forsøkt å bytte dem ut en etter en, så vil kun den siste variabelen være byttet ut i det filnavnet som blir stående til slutt

Grunn 'Template'-objektet inneholder verdien som er definert i target-attributtet for denne malen i reglene. Verdien som blir returnert av 'getTarget' blir ikke tatt vare på, og når metoden så kalles neste gang taes det utgangspunkt i den samme verdien som det ble tatt utgangspunkt i første gangen.

Forbedring En mulig måte å løse dette på vil være å ta vare på verdien til filnavnet etter hvert kall av 'getTarget' i 'Template'-objektet, og dermed ha en oppdatert verdi for hvert kall av metoden med de ulike parametrene.

Filnavn på genererte filer

Beskrivelse Det er ikke mulig å bruke filnavn på genererte filer der det ikke skal byttes ut noen variabel.

Grunn Metoden 'getTarget' blir bare kjørt dersom det finnes en regel som har type 'filereplace'.

Forbedring Det kan tenkes at dette kan løses ved å skrive om koden slik at filnavnet blir hentet ut uavhengig av bruken av reglene ellers.

Filnavn på genererte filer

Beskrivelse Variabelen som skal skiftes ut i filnavnet for genererte filer må også forekomme inne i den tilhørende malen, ellers kan ikke filnavnet hentes ut.

Grunn En regel blir ikke brukt med mindre det blir truffet på en tilhørende variabel under gjennomgangen av malen, og da vil det heller ikke oppdages at denne regelen skulle være brukt for å få ut filnavnet.

Forbedring Skrive om kodegenereringen slik at denne avhengigheten unngåes.

Sourcerestriction-attributtet

Beskrivelse Verdien som brukes i sourcerestriction-attributtet i regelfilene må finnes igjen som en verdi til et attributt i inndataene, og kan ikke finnes som en verdi i en tekstnode.

Grunn CodeController sjekker ikke om angitt inndatanode har tekstverdi i en tekstnode, hvis attributt ikke er angitt.

Forbedring Skrive om koden i CodeController som behandler sourcerestriction-attributtet slik at dette behandler verdier i tekstnoder på samme måten som det gjøres for source-attributtet.

Det grafiske brukergrensesnittet

Beskrivelse Det grafiske brukergrensesnittet er ikke fullt så bra som det som ville være ønskelig. For eksempel blir kun siste delen av feilmeldinger liggende i skjermbildet, og gamle meldinger blir fort overskrevet.

Grunn Grafisk brukergrensesnitt var ikke et krav til systemet, og koding av dette har derfor ikke vært noen prioritert oppgave.

Forbedring GUIen må oppdateres slik at den inneholder alle ønskelige elementer.

5.2 Feil og mangler i regler og maler

Dette avsnittet handler om momenter som omhandler feil og mangler i regel- og malformatet.

5.2.1 Mangler i format

Ikke støtte for standard utbytting

Beskrivelse I regler som har en 'map'-attributt (se [KON] kapittel 4.3.1, Regelfilen) må den tilhørende regelen for bytting av innholdet inneholde eksakt den gjeldende verdien i 'old'-attributtet.

Grunn Det er ikke mulig å definere noen standard utbytting som skal skje hvis alle reglene slår feil.

Forbedring Legge til mulighet for at 'map'-attributtet kan angi standard utbyttingsverdi.

Ikke dobbel linking i reglene

Beskrivelse I en del tilfeller vil det være nødvendig å hente verdier fra flere steder i inndataene for å bestemme hva som skal komme ut. Dette er løst ved 'link'-attributtet (se [KON] kapittel 4.3.1, Regelfilen) i reglene, men i de tilfeller der det må linkes i mer enn ett nivå så er ikke dette mulig.

Grunn Støtte for dette er ikke lagt til. Det er foreløpig valgt å kun linke i ett nivå da det er mulig å løse linking i flere nivåer ved hjelp av interne maler.

Forbedring Legge til mulighet for at 'link'-atttributten kan inneholde flere nivåer med linker for eksempel ved å innføre et skilletegn for å skille nivåer.

I kapittel 6, Videre utvidelser, er det angitt en del mulige forbedringer av systemet. Utover det, og de mangler som er nevnt her er det ikke avdekket noen feil i regel- og mal-formatet.

5.2.2 Mangler i eksempler

Mangler støtte for relasjoner mellom entiteter

Beskrivelse Forhold mellom entiteter i modellene blir ikke representert i den genererte koden.

Grunn Det er ikke tatt hensyn til forhold mellom entiteter i de eksempelmalene og de tilhørende reglene som er laget.

Forbedring Legge til referanseattributt som egen variabel i malene, og lage korresponderende regler i regelfilene. Regelen for assosiasjonsrelasjon må være av typen 'multi' (se [KON], kapittel 4.3.1, Regelfilen) og XMI-nodene som er aktuelle har navnet 'Foundation.Core.Association'. Samme framgangsmåte for arverelasjoner osv.

Genererer ikke deplymentdescriptorer

Beskrivelse Deploymentdescriptor blir ikke generert for EJB-arkitekturene.

Grunn Det er ikke laget mal for "deployment descriptor", og de to regelfilene for de to EJB-arkitekturene har ikke regler som kan fylle ut en slik mal med data.

Forbedring Lage nødvendig mal og tilhørende regler.

6 Videre utvidelser

Her listes det opp en del uløste problemer i implementasjonen, forslag til løsning av problemene og hvordan løsningene kan implementeres.

6.1 Feilhåndtering i behandling av mal

Problem	Feil i generering av kode ut fra en mal gjør at generatoren stopper
Fremtidig løsning	Generatoren går videre i treet når det oppstår feil i behandlingen av en mal i tilknytning til en triggernode.
Implementasjon	*I CodeController-klassen* Catche BuildExceptions ved kall på 'useTemplate'-metoden, og logge advarsler istedenfor å stoppe prosessen.

6.2 Av og til ineffektiv tretraversering

Problem	Hvis et subtre brukes i flere maler i en traversering av et XMI-tre, vil nodene i det treet besøkes flere ganger.
Fremtidig løsning	Behandle maler parallellt ved besøk av XMI-noder.
Implementasjon	*I CodeController-klassen* useTemplate-metoden kan ta inn flere templates, men fremdeles bare ett subtre. I while-løkken hvor det gåes gjennom tokens, kan det hentes ut tokens for hver mal, og alle malene behandles samtidig. Metoden params.isTrigger(...) må brukes på alle nodene før det begynnes med malene når det gjøres på denne måten. Det må bygges opp en passende datastruktur for dette formålet. Denne metoden fører til en annen struktur på generateByElement-metoden.

6.3 Innstillinger av config-pakken

Problem	Tungvindt å linke innstillinger i arkitekturfiler til kataloger, regelfiler og malfiler.
Fremtidig løsning	Legge mest mulig valg av innstillinger i GUI.
Implementasjon	*I config-pakken* Lage en vindu i GUI, som kan åpnes fra MainWindow-klassen, hvor du bl.a. kan angi hvor arkitekturfilen ligger.

6.4 Validering av XML i parseren

Problem XML blir ikke validert før parsing.

Fremtidig løsning Det er mulig å sette på validering av XMLen som parses. Funksjonaliteten er egentlig ferdig implementert.

Implementasjon Det som må gjøres for at det skal virke er bare å "sette på" funksjonaliteten. Dette må gjøres i klassen xmlparser.QproParserImpl, metode parseXML(..). Der er det en linje som per i dag ser ut som følger:

```
return XmlDocument.createXmlDocument(src, !true);
```

For å sette på valideringen må '!' foran 'true' fjernes. Det er i tillegg nødvendig å utarbeide en gyldig DTD for XML filen.

6.5 Forbedret GUI

Problem GUIen er veldig enkel og gir lite hjelp

Fremtidig løsning Det vil være mulig å legge til elementer som for eksempel mulighet for valg av flere arkitekturer, hjelp, bedre visualisering av de forskjellige valgene og bedre informasjon mens genereringen foregår.

Implementasjon Alle slike endringer gjøres i hovedsak i pakken 'GUI'. Ønskede elementer og metoder må legges til. Det kan eventuelt bli nødvendig med støttemetoder i 'controller.QproController' for å gi informasjon til/fra GUIen.

6.6 XMI-konverterer

Problem Per i dag parser systemet en XMI fil rått og leverer et tre til kodegeneratoren. Generatoren må så jobbe mot dette treet. Reglene må nå også kobles opp mot XMI en for å vite hvilken info som skal til hvilken plass i malene.

Fremtidig løsning Hvis det legges inn et konverteringsverktøy i parserne kan det tenkes at et sett med objekter leveres til generatoren i stedet, for eksempel attributter, metoder osv. Det blir da opp til denne konvertereren å bestemme hva som er relevant info i XMIen, og denne infoen trengs ikke i reglene. Dette vil kunne gjøre generatoren raskere, og opprettingen av regler blir enklere.

Implementasjon Denne endringen vil måtte implementeres i parseren til systemet, i pakken 'XMLparser'

7 Modultesting

Dette kapittelet inneholder informasjon om testprosess, kjøreeksempel fra modultestene, samt testlogg og endringslogg.

7.1 Testprossess

Underveis og spesielt mot slutten av implementasjonsfasen ble det startet med testing av koden. Til å begynne med besto testingen av uformell testing av utvikleren som drev med koden. Dette var som oftest gjennomlesing og kompilering, samt enkelte små dummies eller testrigger for å se at kodebiter oppførte seg rett. Etter hvert ble modultester utført. Dette ble utført som beskrevet i konstruksjonsdokumentet med testmoduler som testet om modulen hadde rett oppførsel utad. I noen testtilfeller ble testmodulen benyttet, mens i andre tilfeller holdt det med enkelte interne testmetoder. Det som spesielt ble testet var at parsemodulen kunne ta inn korrekt XMI og levere en korrekt representasjon av denne i form av en trestruktur, samt at controllerpakken i samarbeid med parampakken får generert korrekt kode ut fra de spesifiserte malene og reglene. Det var forventet å lage flere testrigger enn det som ble gjort. Men da de fleste modulene ble ferdig samtidig har også de blitt brukt for å teste hverandre.

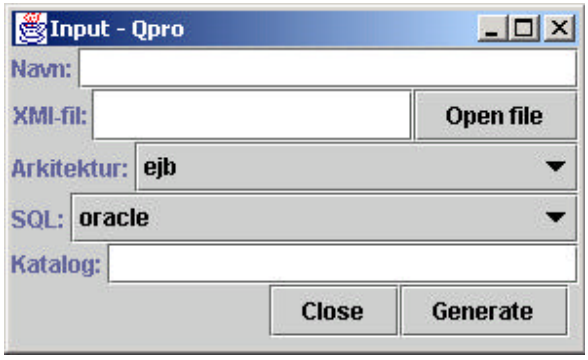
Mot slutten av modultesten ble koden i modulene inspisert av en annen utvikler. Det ble forsøkt å få den som hadde hatt minst med koden i modulen å gjøre til å gjennomføre gjennomlesningen. Men da gruppen er så liten vet alle en god del om hva som er i de ulike modulene. Det førte til at inspeksjonen kunne ha blitt bedre med en helt uavhengig gjennomleser.

7.2 Testresulater

Her vil det bli presentert endel av de resultatene som har fremkommet under testingen av de ulike modulene. Kildekode for alle testmodulene finnes i systemdokumentasjonen sammen med koden for resten av programmet.

7.2.1 Test av 'GUI'-pakken

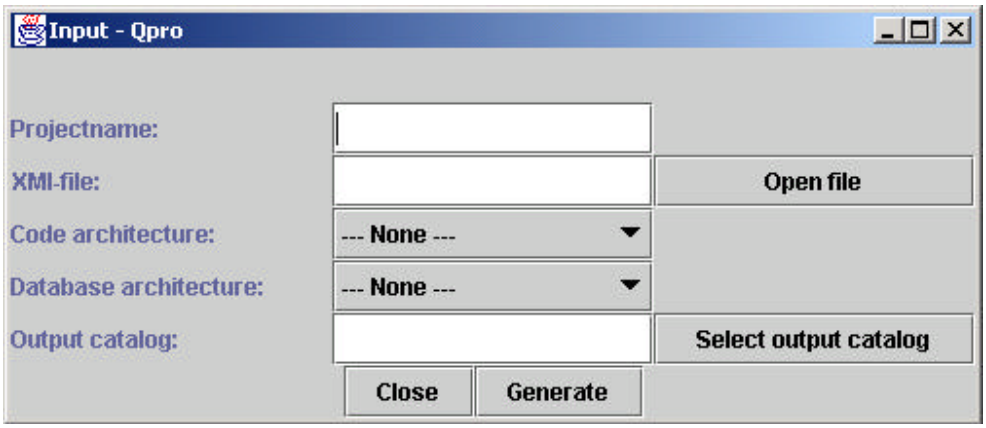
GUI så i første omgang ut som på figuren øverst på neste side.



Figur 7.1. Førsteutkast til GUI

Modultest av GUI har fokusert fokusert på utseende og brukervennlighet, men ettersom det ikke er noe krav til brukergrensesnitt i prosjektet vårt er det lagt mindre vekt på modultesten av denne enn de andre modulene.

Det som var savnet i førsteutkastet til GUI var forklarende feilmeldinger og muligheten for å velge ingen arkitektur. Dessuten var navnene på GUI for lite forklarende, og ikke på engelsk som tidligere bestemt internt i gruppa. Det ble foreslått å endre navn til Project Name, arkitektur til Code Architecture, SQL til Database Architecture og katalog til Output Catalog. Endelig GUI ble seende ut som nedenfor. Et ganske simpelt brukergrensesnitt, men med all nødvendig funksjonalitet.



Figur 7.2. Endelig GUI

7.2.2 Test av 'XMLparser'-pakken

Nedenfor følger et forkortet kjøreeksempel fra parseren. Det ble på forhånd generert en XML-fil med navn 'testXmi.xml', og parseren ble testet på denne. Av kjøreeksempelet ble det bygget opp en trestruktur som forventet. Denne ble verifisert ved å studere inndata og resultat manuelt.

Kjøreeksempel for testParser for godkjent XMI fil:

```
F:\source\class>java test.TestParser testXmi.xml
***** Starting the test of the xmlParser.QproParser *****
All went well, this was the result:
ELEMENT-NODE with name [XMI] and [1] attributes
      TEXT-NODE with value: []
      ELEMENT-NODE with name [XMI.header] and [0] attributes
            TEXT-NODE with value: []
            ELEMENT-NODE with name [XMI.documentation] and [0]
attributes
                  TEXT-NODE with value: []
                  ELEMENT-NODE with name [XMI.exporter] and [0]
attributes
                        TEXT-NODE with value: [Together]
                  TEXT-NODE with value: []

...

                  TEXT-NODE with value: []
            TEXT-NODE with value: []
      TEXT-NODE with value: []
```

Det ble også testet om det var mulig å angi et fil som ikke har gyldig XMI syntaks. Nedenfor kan det sees at dette ga parsefeil.

Kjøreeksempel for testParser for en ikke godkjent XMI fil:

```
F:\source\class>java test.TestParser parse.txt
***** Starting the test of the xmlParser.QproParser *****
Something went wrong:
Parsererror!
 Invalid XML input:
Document root element is missing.
```

Konklusjonen blir at denne modulen fungerer veldig bra og ikke trenger ytterligere modultest. De er antatt at de største problemene ligger i ’param’-pakken og ’controller’-pakken.

7.2.3 Test av ‘param’-pakken

For test av ‘param’-pakken var det laget to testmoduler. En for ParamController klassen og en for hele ’param’-pakken. Testen for hele pakken viser 9 punkter som tester ulike aspekter i ’param’-pakken. Programmerer hadde gjort en god jobb med testklassen slik at testingen gikk veldig greit. Ettersom testParam også tester ParamController vises kun et kjøreeksempel for testParam.

Kjøreeksempel for testParam:

```
F:\source\class>java test.TestParam
Starting test of package Param
...
```

```
1. XMI-root-element correctly returned.
2. Projectname correctly returned.
3. Source-directory correctly returned.
4. System architecture correctly returned.
5. Database architecture correctly returned.
6. Correctly identified trigger in sys-rules.
7. Correctly ignored wrong trigger in sys-rules.
8. Correctly identified trigger in db-rules.
9. Correctly ignored wrong trigger in db-rules.
Hopefully all 9 tests have been completed successfully!

Starting to list templates
Template with target PersonEJB.java , internal: false
Template with target PersonPK.java , internal: false
Template with target PersonHome.java , internal: false
Template with target PersonRemote.java , internal: false

Starting to list templates
Template with target Person.sql , internal: false

The rule CLASSNAME is of the type replace;filereplace
ELEMENT-NODE with name [CLASSNAME] and [5] attributes

The rule ATTRIBUTE is of the type multi
ELEMENT-NODE with name [ATTRIBUTE] and [3] attributes
        TEXT-NODE with value: []
        ELEMENT-NODE with name [ATTRIBUTE_VISIBILTY] and [4] attributes
        TEXT-NODE with value: []
        ELEMENT-NODE with name [ATTRIBUTE_TYPE] and [4] attributes
        TEXT-NODE with value: []
        ELEMENT-NODE with name [ATTRIBUTE_NAME] and [4] attributes
        TEXT-NODE with value: []

Starting to print internal template CMP-FIELDS_TEMPLATE:

<cmp-field>
         <field-name><<<<FIELD>>>></field-name>
</cmp-field>

Starting to print internal template OPERATION_TEMPLATE:

/*************************************************************
 * @Name   <<<<OPERATION_NAME>>>>
 *
 * @Input  <<<<OPERATION_INPUT>>>>
 * @Return <<<<OPERATION_TYPE>>>>
 *************************************************************/
<<<<OPERATION_VISIBILITY>>>> <<<<OPERATION_TYPE>>>>
<<<<OPERATION_NAME>>>>(<<<<OPERATION_INPUT>>>>){

}

Starting to print internal template SQL_ATTRIBUTE:
<<<<ATTRIBUTE_NAME>>>> <<<<ATTRIBUTE_VISIBILITY>>>>,
```

Modultesten av ‘param’-pakken fant lite feil. Det er likevel mulig at det vil dukke opp feil senere i integrasjonstesten.

7.2.4 Test av ‘controller’-pakken

Logging av data ble testet separat med klassen TestLog. Endel testparametere måtte angis. Dette var navn på loggfil, katalog som det skulle logges til, samt en tekststreng med den teksten som skulle logges til fil. Etter kjøring av kommandoen nedenfor var det opprettet en testlogg.txt fil i ønsket katalog med ønsket innhold, nemlig den angitte tekststrengen.

Kjøreeksempel for testLog:

```
F:\source\class>java test.TestLog testlog.txt f\source\class "Tester
loggfunksjonen ...."
***** Starting the test of the controller.TestLog *****
THE TEST SUCCEEDED SEE f:\source\class/testlog.txt to sæ the logg
generated
```

Det var ellers vanskelig å lage noen egen modultest for hele ‘controller’-pakken ettersom den benytter seg flittig av spesielt ’param’-pakken, men også alle de andre. ’controller’-pakken ble testet ved å forsøke å starte hele systemet.

7.2.5 Test av ’error’-pakken

Denne pakken er såpass enkel at dette skal være unødvendig å lage noen testmodul. Det som ble sjekket var at arvingen var korrekt. QproException arver fra java.lang.Exception og fungerer som en superklasse for alle feilmeldingene som kan oppstå. De unntakene som blir kastet er enten BuildException, ParamException, ParseException eller LogException. Det ble sjekket i koden at de nødvendige metoder var med i disse og alle disse klassene arvet QproException.

7.3 Testlogger

Dette avsnittet inneholder testloggen som er en tabell der det er ført inn alle modultestene som er blitt utført, samt en endringslogg som viser hvilke feil fra modultesten som ble rettet og av hvem.

7.3.1 Testlogg

TestID	Testdato	Testet av	Resultat
GUI1	05.11.2001	Atle	Lite forståelige navn i GUI
GUI2	05.11.2001	Atle	Må ha mulighet for å velge ingen arkitektur
GUI3	05.11.2001	Atle	Mangel på feilmeldinger
Parser1	05.11.2001	Atle	OK. 'XMLparser'-pakken godkjent.
Error1	06.11.2001	Atle	OK. 'error'-pakken godkjent.
GUI4	06.11.2001	Atle	Dumt at det må trykkes to ganger på 'Open' når det skal velges output-katalog.
Param1	06.11.2001	Atle	Mangler regler for å generere PrimaryKey klasse i EJB arkitekturen.
Param2	07.11.2001	Atle	OK. 'param'-pakken godkjent.
Controller1	07.11.2001	Atle	OK. Loggfunksjonen godkjent.
Controller2	07.11.2001	Atle	Template blir ikke nullstilt før ny gjennomkjøring.
Controller3	08.11.2001	Martin	Koden som blir generert er EJB 1.0 ikke 2.0.
Controller4	08.11.2001	Atle	OK. 'controller'-pakken godkjent.
GUI5	09.11.2001	Atle	OK. 'GUI'-pakken godkjent.

7.3.2 Endringslogg

TestID	Endringsdato	Endret av	Endring
GUI1	05.11.2001	Magnus	Mer strukturert GUI med forståelige navn.
GUI2	05.11.2001	Magnus	Mulighet for å velge ingen arkitektur.
Param1	06.11.2001	Ole Kristian	Regler for generering av PrimaryKey klasse for EJB arkitekturen opprettet.
Controller2	07.11.2001	Odd Christian	Fikset reset av template før denne kjøres gjennom på ny.
Controller3	08.11.2001	Odd Christian	Endret malen for EJB til å stemme helt overens med EJB 2.0 standarden.

7.4 Kodeinspeksjon

På grunn av tidspress ble det valgt å kun inspisere koden hos de tre viktigste klassene i systemet, nemlig CodeController.java, ParamController.java og Rules.java. Sjekklisten fra konstruksjonsdokumentet ble brukt til hjelp i kodeinspeksjonen. Selve kodeinspeksjonen ble utført av de to på gruppa som hadde programmert minst på systemet.

Følgende momenter ble avdekket i kodeinspeksjonen:

- ?? Referansen til kildefil i toppen av hver fil må bort.
- ?? Variabler som ikke ble brukt og som kan fjernes.
- ?? Kodebiter som ikke ble brukt og kan fjernes.
- ?? Fravær av javadoc på enkelte interne metoder.
- ?? Manglende javadoc, spesielt @throws parameteren.
- ?? Hardkoding av data om XMI-input.
- ?? Usømmelige metodenavn.
- ?? Usømmelig dokumentasjon.
- ?? Unntak som bare blir fanget opp uten at unntaksmeldingen blir sendt videre.
- ?? Variabler som får tildelt samme verdi to ganger.

De enkelte programmererne fikk tilbakemelding og beskjed om å fikse sin kode.

8 Integrasjonstesting

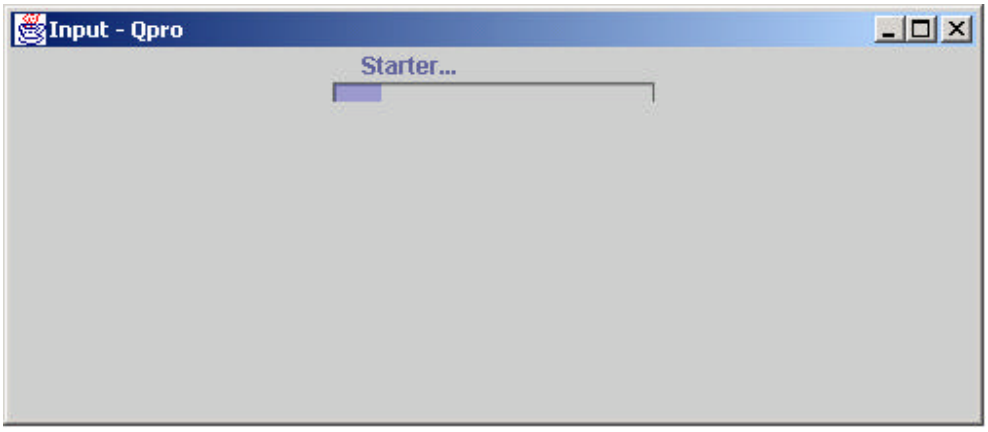
Dette kapittelet beskriver hvordan integrasjonen av de ulike modulene ble utført.

8.1 Testprosess

Noe sammensying ble utført allerede ved modultest av 'controller'-pakken, fordi denne modulen var såpass avhengig av spesielt 'param'-pakken. Etter hvert som sammensyingen ble gjennomført, ble det foretatt en lang rekke uformelle kjøretester som avdekket til dels mange småfeil, og enkelte større feil. Disse ble rettet, og det ble til slutt foretatt en integrasjonstest. Denne bestod av gjennomkjøringer av systemet med fokus på jakt etter elementer som ikke fungerte eller ikke var tilfredsstillende.

8.2 Testresultater

Det har vært en stor fordel at all kommunikasjonen mellom de ulike pakkene stort sett har ligget uforandret fra konstruksjonsfasen. Dette medførte at det ikke var de helt store problemene ved integrasjonen av de ulike modulene. En av de største problemene har vært at GUI låser seg under kodegenereringsprosessen. Det viste seg at koden ble generert uavhengig av det som ble vist, og at dette kun var et problem i GUI og ikke i det underliggende systemet som fryktet.



Figur 8. 1. GUI låser seg ved generering av kode.

Det ble forsøkt å generere kode for ulike arkitekturer. Integrasjonstesten påviste endel feil og mangler i reglene for ulike kode- og databasearkitekturer. Det viste seg at dette i hovedsak skyldes utdaterte regelfiler på grunn av enkelte nye tager som har blitt innført. Gruppa har bestemt seg for i første omgang å konsentrere seg om å få en fungerende EJB 2.0 regelfil med tilhørende maler.

Konklusjon av integrasjonstesten er at forbedringer må gjøres på GUI og regel/mal-filer, men at integrasjonen stort sett har vært problemfri. De ansvarlige gruppemedlemmer har fått beskjed om at endringer må utføres.

9 Litteraturliste

[IJ1] IntelliJ IDEA
http://www.intellij.com/

10 Vedlegg

10.1 Vedlegg 1 – Kodekonvensjon

Dette dokumentet beskriver kodekonvensjonen som er benyttet i forbindelse med implementeringen. Dette er en konvertert utgave av PDF-originalen som kan finnes på: ftp://ftp.javasoft.com/docs/codeconv/CodeConventions.pdf

Java Code Conventions

1 - Introduction

1.1 Why Have Code Conventions

Code conventions are important to programmers for a number of reasons:
• 80% of the lifetime cost of a piece of software goes to maintenance.
• Hardly any software is maintained for its whole life by the original author.
• Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
• If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.
For the conventions to work, every person writing software must conform to the code conventions. Everyone.

1.2 Acknowledgments

This document reflects the Java language coding standards presented in the *Java Language Specification,* from Sun Microsystems, Inc. Major contributions are from Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath, and Scott Hommel.
This document is maintained by Scott Hommel. Comments should be sent to shommel@eng.sun.com

2 - File Names

This section lists commonly used file suffixes and names.

2.1 File Suffixes

Java Software uses the following file suffixes:

File Type Suffix

Java source `.java`
Java bytecode `.class`

2.2 Common File Names

Frequently used file names include:

3 - File Organization

A file consists of sections that should be separated by blank lines and an optional comment identifying each section.
Files longer than 2000 lines are cumbersome and should be avoided.
For an example of a Java program properly formatted, see “Java Source File Example” on page 18.

3.1 Java Source Files

Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.
Java source files have the following ordering:
• Beginning comments (see “Beginning Comments” on page 2)
• Package and Import statements
• Class and interface declarations (see “Class and Interface Declarations” on page 3)

3.1.1 Beginning Comments

All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice:

```
/*
* Classname
*
* Version information
*
* Date
*
* Copyright notice
*/
```

File Name Use

`GNUmakefile` The preferred name for makefiles.
We use `gnumake` to build our software.
`README` The preferred name for the file that summarizes the contents of a particular directory.

3.1.2 Package and Import Statements

The first non-comment line of most Java source files is a `package` statement. After that, `import` statements can follow. For example:

```
package java.awt;
import java.awt.peer.CanvasPeer;
```

3.1.3 Class and Interface Declarations

The following table describes the parts of a class or interface declaration, in the order that they should appear. See “ Java Source File Example” on page 18 for an example that includes comments.

Part of Class/Interface Declaration	Notes
1 Class/interface documentation comment (`/**...*/`)	See “Documentation Comments” on page 8 for information on what should be in this comment.
2 `class` or `interface` statement	
3 Class/interface implementation comment (`/*...*/`), if necessary	

This comment should contain any class-wide or
interface-wide information that wasn't appropriate
for the class/interface documentation comment.
4 Class (`static`) variables First the `public` class variables, then the `protected`,
then package level (no access modifier),
and then the `private`.
5 Instance variables First `public`, then `protected`, then package
level (no access modifier), and then `private`.
6 Constructors
7 Methods These methods should be grouped by functionality
rather than by scope or accessibility. For
example, a private class method can be in
between two public instance methods. The goal is
to make reading and understanding the code easier.

4 - Indentation

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces (not 4).

4.1 Line Length

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

Note: Examples for use in documentation should have a shorter line length—generally no more than 70 characters.

4.2 Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:
• Break after a comma.
• Break before an operator.
• Prefer higher-level breaks to lower-level breaks.
• Align the new line with the beginning of the expression at the same level on the previous line.
• If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.
Here are some examples of breaking method calls:

```
someMethod(longExpression1, longExpression2, longExpression3,
longExpression4, longExpression5);
var = someMethod1(longExpression1,
someMethod2(longExpression2,
longExpression3));
```

Following are two examples of breaking an arithmetic expression. The first is preferred, since the break occurs outside the parenthesized expression, which is at a higher level.

```
longName1 = longName2 * (longName3 + longName4 - longName5)
+ 4 * longname6; // PREFER
longName1 = longName2 * (longName3 + longName4
- longName5) + 4 * longname6; // AVOID
```

Following are two examples of indenting method declarations. The first is the conventional case. The second would shift the second and third lines to the far right if it used conventional indentation, so instead it indents only 8 spaces.

```
//CONVENTIONAL INDENTATION
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
Object andStillAnother) {
...
}
```

```
//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS
private static synchronized horkingLongMethodName(int anArg,
Object anotherArg, String yetAnotherArg,
Object andStillAnother) {
...
}
```

Line wrapping for `if` statements should generally use the 8-space rule, since conventional (4 space) indentation makes seeing the body difficult. For example:

```
//DON'T USE THIS INDENTATION
if ((condition1 && condition2)
|| (condition3 && condition4)
||!(condition5 && condition6)) { //BAD WRAPS
doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS
}
//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
|| (condition3 && condition4)
||!(condition5 && condition6)) {
doSomethingAboutIt();
}
//OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
||!(condition5 && condition6)) {
doSomethingAboutIt();
}
```

Here are three acceptable ways to format ternary expressions:

```
alpha = (aLongBooleanExpression) ? beta : gamma;
alpha = (aLongBooleanExpression) ? beta
: gamma;
alpha = (aLongBooleanExpression)
? beta
: gamma;
```

5 - Comments

Java programs can have two kinds of comments: implementation comments and documentation comments. Implementation comments are those found in C++, which are delimited by `/*...*/`, and `//`. Documentation comments (known as "doc comments") are Java-only, and are delimited by `/**...*/`. Doc comments can be extracted to HTML files using the javadoc tool.

Implementation comments are means for commenting out code or for comments about the particular implementation. Doc comments are meant to describe the specification of the code, from an implementation-free perspective to be read by developers who might not necessarily have the source code at hand.
Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program. For example, information about how the corresponding package is built or in what directory it resides should not be included as a comment.
Discussion of nontrivial or nonobvious design decisions is appropriate, but avoid duplicating information that is present in (and clear from) the code. It is too easy for redundant comments to get out of date. In general, avoid any comments that are likely to get out of date as the code evolves.

Note: The frequency of comments sometimes reflects poor quality of code. When you feel compelled to add a comment, consider rewriting the code to make it clearer.
Comments should not be enclosed in large boxes drawn with asterisks or other characters.

Comments should never include special characters such as form-feed and backspace.

5.1 Implementation Comment Formats

Programs can have four styles of implementation comments: block, single-line, trailing and end-of-line.

5.1.1 Block Comments

Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments may be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe.
A block comment should be preceded by a blank line to set it apart from the rest of the code.

```
/*
* Here is a block comment.
*/
```

Block comments can start with `/*-`, which is recognized by **indent**(1) as the beginning of a block comment that should not be reformatted. Example:

```
/*-
* Here is a block comment with some very special
* formatting that I want indent(1) to ignore.
*
* one
* two
* three
*/
```

Note: If you don't use **indent**(1), you don't have to use `/*-` in your code or make any other concessions to the possibility that someone else might run **indent**(1) on your code.
See also "Documentation Comments" on page 8.

5.1.2 Single-Line Comments

Short comments can appear on a single line indented to the level of the code that follows. If a comment can't be written in a single line, it should follow the block comment format (see section 5.1.1). A single-line comment should be preceded by a blank line. Here's an example of a single-line comment in Java code:

```
if (condition) {
/* Handle the condition. */
...
}
```

5.1.3 Trailing Comments

Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting.
Here's an example of a trailing comment in Java code:

```
if (a == 2) {
return TRUE; /* special case */
} else {
return isPrime(a); /* works only for odd a */
}
```

5.1.4 End-Of-Line Comments

The `//` comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code. Examples of all three styles follow:

```
if (foo > 1) {
// Do a double-flip.
```

```
...
}
else{
return false; // Explain why here.
}
//if (bar > 1) {
//
// // Do a triple-flip.
// ...
//}
//else{
// return false;
//}
```

5.2 Documentation Comments

Note: See "Java Source File Example" on page 18 for examples of the comment formats described here.

For further details, see "How to Write Doc Comments for Javadoc" which includes information on the doc comment tags (`@return`, `@param`, `@see`):
`http://java.sun.com/products/jdk/javadoc/writingdoccomments.html`

For further details about doc comments and javadoc, see the javadoc home page at:
`http://java.sun.com/products/jdk/javadoc/`

Doc comments describe Java classes, interfaces, constructors, methods, and fields. Each doc comment is set inside the comment delimiters `/**...*/`, with one comment per class, interface, or member. This comment should appear just before the declaration:

```
/**
* The Example class provides ...
*/
public class Example { ...
```

Notice that top-level classes and interfaces are not indented, while their members are. The first line of doc comment (`/**`) for classes and interfaces is not indented; subsequent doc comment lines each have 1 space of indentation (to vertically align the asterisks). Members, including constructors, have 4 spaces for the first doc comment line and 5 spaces thereafter.

If you need to give information about a class, interface, variable, or method that isn't appropriate for documentation, use an implementation block comment (see section 5.1.1) or single-line (see section 5.1.2) comment immediately *after* the declaration. For example, details about the implementation of a class should go in in such an implementation block comment *following* the class statement, not in the class doc comment.

Doc comments should not be positioned inside a method or constructor definition block, because Java associates documentation comments with the first declaration *after* the comment.

6 - Declarations

6.1 Number Per Line

One declaration per line is recommended since it encourages commenting. In other words,

```
int level; // indentation level
int size; // size of table
```

is preferred over

```
int level, size;
```

Do not put different types on the same line. Example:

```
int foo, fooarray[]; //WRONG!
```

Note: The examples above use one space between the type and the identifier. Another acceptable alternative is to use tabs, e.g.:

```
int level; // indentation level
int size; // size of table
Object currentEntry; // currently selected table entry
```

6.2 Initialization

Try to initialize local variables where they're declared. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.

6.3 Placement

Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}".) Don't wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.

```
void myMethod() {
int int1 = 0; // beginning of method block
if (condition) {
int int2 = 0; // beginning of "if" block
...
}
}
```

The one exception to the rule is indexes of `for` loops, which in Java can be declared in the `for` statement:

```
for (int i = 0; i < maxLoops; i++) { ... }
```

Avoid local declarations that hide declarations at higher levels. For example, do not declare the same variable name in an inner block:

```
int count;
...
myMethod() {
if (condition) {
int count; // AVOID!
...
}
...
}
```

6.4 Class and Interface Declarations

When coding Java classes and interfaces, the following formatting rules should be followed:

• No space between a method name and the parenthesis "(" starting its parameter list
• Open brace "{" appears at the end of the same line as the declaration statement
• Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"

```
class Sample extends Object {
int ivar1;
int ivar2;
Sample(int i, int j) {
ivar1 = i;
ivar2 = j;
}
int emptyMethod() {}
...
}
```

• Methods are separated by a blank line

7 - Statements

7.1 Simple Statements

Each line should contain at most one statement. Example:

```
argv++; // Correct
argc++; // Correct
argv++; argc--; // AVOID!
```

7.2 Compound Statements

Compound statements are statements that contain lists of statements enclosed in braces "`{ statements }`". See the following sections for examples.

• The enclosed statements should be indented one more level than the compound statement.

• The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.

• Braces are used around all statements, even single statements, when they are part of a control structure, such as a `if-else` or `for` statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

7.3 return Statements

A `return` statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

7.4 if, if-else, if else-if else Statements

The `if-else` class of statements should have the following form:

```
if ( condition) {
statements;
}
if ( condition) {
statements;
} else {
statements;
}
if ( condition) {
statements;
} else if ( condition) {
statements;
} else {
statements;
}
```

Note: `if` statements always use braces {}. Avoid the following error-prone form:

```
if ( condition) //AVOID! THIS OMITS THE BRACES {}!
statement;
```

7.5 for Statements

A `for` statement should have the following form:

```
for ( initialization; condition; update) {
statements;
}
```

An empty `for` statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form:

```
for ( initialization; condition; update);
```

When using the comma operator in the initialization or update clause of a `for` statement, avoid the complexity of using more than three variables. If needed, use separate statements before the `for` loop (for the initialization clause) or at the end of the loop (for the update clause).

7.6 while Statements

A `while` statement should have the following form:

```
while ( condition) {
statements;
}
```

An empty `while` statement should have the following form:

```
while ( condition);
```

7.7 do-while Statements

A `do-while` statement should have the following form:

```
do {
statements;
} while ( condition);
```

7.8 switch Statements

A `switch` statement should have the following form:

```
switch ( condition) {
case ABC:
statements;
/* falls through */
case DEF:
statements;
break;
case XYZ:
statements;
break;
default:
statements;
break;
}
```

Every time a case falls through (doesn't include a `break` statement), add a comment where the `break` statement would normally be. This is shown in the preceding code example with the `/* falls through */` comment.

Every `switch` statement should include a default case. The `break` in the default case is redundant, but it prevents a fall-through error if later another `case` is added.

7.9 try-catch Statements

A `try-catch` statement should have the following format:

```
try {
statements;
} catch (ExceptionClass e) {
statements;
}
```

A `try-catch` statement may also be followed by `finally`,
which executes regardless of whether or not the `try` block has completed successfully.

```
try {
statements;
} catch (ExceptionClass e) {
statements;
} finally {
statements;
}
```

8 - White Space

8.1 Blank Lines
Blank lines improve readability by setting off sections of code that are logically related.

Two blank lines should always be used in the following circumstances:
• Between sections of a source file
• Between class and interface definitions
One blank line should always be used in the following circumstances:
• Between methods
• Between the local variables in a method and its first statement
• Before a block (see section 5.1.1) or single-line (see section 5.1.2) comment
• Between logical sections inside a method to improve readability

8.2 Blank Spaces
Blank spaces should be used in the following circumstances:
• A keyword followed by a parenthesis should be separated by a space. Example:
```
while (true) {
...
}
```
Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.
• A blank space should appear after commas in argument lists.
• All binary operators except `.` should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment (“++”), and decrement (“--”) from their operands. Example:
```
a += c + d;
a = (a + b) / (c * d);
while (d++ = s++) {
n++;
}
prints("size is " + foo + "\n");
```
• The expressions in a `for` statement should be separated by blank spaces. Example:
```
for (expr1; expr2; expr3)
```
• Casts should be followed by a blank space. Examples:
```
myMethod((byte) aNum, (Object) x);
myMethod((int) (cp + 5), ((int) (i + 3))
+ 1);
```

9 - Naming Conventions
Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier—for example, whether it’s a constant, package, or class—which can be helpful in understanding the code.

Identifier Type Rules for Naming Examples
Packages The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981.
Subsequent components of the package name vary according to an organization’s own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project,

machine, or login names.

```
com.sun.eng
com.apple.quicktime.v2
edu.cmu.cs.bovik.cheese
```

Classes Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words—avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

```
class Raster;
class ImageSprite;
```

Interfaces Interface names should be capitalized like class names.

```
interface RasterDelegate;
interface Storing;
```

Methods Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

```
run();
runFast();
getBackground();
```

10 - Programming Practices

10.1 Providing Access to Instance and Class Variables

Don't make any instance or class variable public without good reason. Often, instance variables don't need to be explicitly set or gotten—often that happens as a side effect of method calls.

One example of appropriate public instance variables is the case where the class is essentially a data structure, with no behavior. In other words, if you would have used a `struct` instead of a class (if Java supported `struct`), then it's appropriate to make the class's instance variables public.

10.2 Referring to Class Variables and Methods

Avoid using an object to access a class (static) variable or method. Use a class name instead. For example:

```
classMethod(); //OK
AClass.classMethod(); //OK
```

Variables Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.

Variable names should be short yet meaningful. The choice of a variable name should be mnemonic— that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are `i`, `j`, `k`, `m`, and `n` for integers; `c`, `d`, and `e` for characters.

```
int i;
char c;
float myWidth;
```

Constants The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

Identifier Type Rules for Naming Examples

```
anObject.classMethod(); //AVOID!
```

10.3 Constants

Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a `for` loop as counter values.

10.4 Variable Assignments

Avoid assigning several variables to the same value in a single statement. It is hard to read. Example:

```
fooBar.fChar = barFoo.lchar = 'c'; // AVOID!
```

Do not use the assignment operator in a place where it can be easily confused with the equality operator. Example:

```
if (c++ = d++) { // AVOID! (Java disallows)
...
}
```
should be written as
```
if ((c++ = d++) != 0) {
...
}
```
Do not use embedded assignments in an attempt to improve run-time performance. This is the job of the compiler. Example:
```
d = (a = b + c) + r; // AVOID!
```
should be written as
```
a = b + c;
d = a + r;
```

10.5 Miscellaneous Practices

10.5.1 Parentheses

It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems. Even if the operator precedence seems clear to you, it might not be to others—you shouldn't assume that other programmers know precedence as well as you do.
```
if (a == b && c == d) // AVOID!
if ((a == b) && (c == d)) // USE
```

10.5.2 Returning Values

Try to make the structure of your program match the intent. Example:
```
if ( booleanExpression) {
return true;
} else {
return false;
}
```
should instead be written as
```
return booleanExpression;
```
Similarly,
```
if (condition) {
return x;
}
return y;
```
should be written as
```
return (condition ? x : y);
```

10.5.3 Expressions before '?' in the Conditional Operator

If an expression containing a binary operator appears before the `?` in the ternary `?:` operator, it should be parenthesized. Example:
```
(x >= 0) ? x : -x;
```

10.5.4 Special Comments

Use `XXX` in a comment to flag something that is bogus but works. Use `FIXME` to flag something that is bogus and broken.

11 - Code Examples

11.1 Java Source File Example

The following example shows how to format a Java source file containing a single public class. Interfaces are formatted similarly. For more information, see "Class and Interface Declarations" on page 3 and "Documentation Comments" on page 8

```
/*
* @(#)Blah.java 1.82 99/03/18
*
* Copyright (c) 1994-1999 Sun Microsystems, Inc.
* 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
* All Rights Reserved.
*
* This software is the confidential and proprietary information of Sun
* Microsystems, Inc. ("Confidential Information"). You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with Sun.
*/
package java.blah;
import java.blah.blahdy.BlahBlah;
/**
* Class description goes here.
*
* @version 1.82 18 Mar 1999
* @author Firstname Lastname
*/
public class Blah extends SomeClass {
/* A class implementation comment can go here. */
/** classVar1 documentation comment */
public static int classVar1;
/**
* classVar2 documentation comment that happens to be
* more than one line long
*/
private static Object classVar2;
/** instanceVar1 documentation comment */
public Object instanceVar1;
/** instanceVar2 documentation comment */
protected int instanceVar2;
/** instanceVar3 documentation comment */
private Object[] instanceVar3;
/**
* ... constructor Blah documentation comment...
*/
public Blah() {
// ...implementation goes here...
}
/**
* ... method doSomething documentation comment...
*/
public void doSomething() {
// ...implementation goes here...
}
```

11 - Code Examples
20

```
/**
* ...method doSomethingElse documentation comment...
* @param someParam description
*/
public void doSomethingElse(Object someParam) {
// ...implementation goes here...
}
}
```

10.2 Vedlegg 2 – CVS kommandoer

Dette vedlegget beskriver de vanligste CVS kommandoene og bruken av dem. Dokumentet er en noe redigert versjon hentet fra http://wwwinfo.cern.ch/asd/cvs/tutorial/cvs_tutorial_toc.html.

Look at the CVS basic command set

Most of the below commands should be executing while in the directory you checked out. If you did a cvs checkout malloc then you should be in the malloc sub-directory to execute most of these commands. cvs release is different and must be executed from the directory above.

cvs checkout (or cvs co)
To make a local copy of a module's files from the repository execute cvs checkout module where module is an entry in your modules file (see below). This will create a sub-directory module and check-out the files from the repository into the sub-directory for you to work on.

cvs update
To update your copy of a module with any changes from the central repository, execute cvs update. This will tell you which files have been updated (their names are displayed with a U before them), and which have been modified by you and not yet committed (preceded by an M).

It can be that when you do an update, the changes in the central copy clash with changes you have made in your own copy. You will be warned of any files that contain clashes by a preceding C. Inside the files the clashes will be marked in the file surrounded by lines of the form <<<< and >>>>. You have to resolve the clashes in your copy by hand. After an update where there have been clashes, your original version of the file is saved as ".#file.version".

If you feel you have messed up a file and wish to have CVS forget about your changes and go back to the version from the repository, delete the file and do an cvs update. CVS will announce that the file has been "lost" and will give you a fresh copy.

cvs commit
When you think your files are ready to be merged back into the repository for the rest of your developers to see, execute cvs commit. You will be put in an editor to make a message that describes the changes that you have made (for future reference). Your changes will then be added to the central copy.

When you do a commit, if you haven't updated to the most recent version of the files, CVS tells you this; then you have to first update, resolve any possible clashes, and then redo the commit.

cvs add and cvs remove
It can be that the changes you want to make involve a completely new file, or removing an existing one. The commands to use here are:

cvs add “filename”
cvs remove “filename”

You still have to do a commit after these commands to make the additions and removes actually take affect. You may make any number of new files in your copy of the repository, but they will not be committed to the central copy unless you do a cvs add.

CVS remove does not actually remove the files from the repository. It only removes them from the "current list" and puts the files in the CVS Attic. When another person checks out the module in the future they will not get the files that were removed. But if you ask for older versions that had the file before it was removed, the file will be checked out of the Attic.

cvs release
When you are done with your local copy of the files for the time being and want to remove your local copy use cvs release module. This must be done in the directory above the module sub-directory you which to release. It safely cancels the effects of cvs checkout. Usually you should do a commit first.

If you wish to have CVS also remove the module sub-directory and your local copy of the files then your cvs release -d module.

NOTE: Take your time here. CVS will inform you of files that may have changed or it does not know about (watch for the ? lines) and then with ask you to confirm this action. Make sure you want to do this.

cvs log
To see the commit messages for files, and who made them, use:

cvs log [“filename(s)”]

cvs diff
To see the differences between your version of the files and the version in the repository do:

cvs diff [“filename(s)”]

cvs tag
One of the exciting features of CVS is its ability to mark all the files in a module at once with a symbolic name. You can say “this copy of my files is version 3”. And then later say “this file I am working on looked better in version 3 so check out the copy that I marked as version 3.”

Use cvs tag to tag the version of the files that you have checked out. You can then at a later date retrieve this version of the files with the tag.

```
cvs tag tag-name ["filename(s)"]
```

Later you can do:

```
cvs co -r tag-name module
```

cvs rtag
Like tag, rtag marks the current versions of files but it does not work on your local copies but on the files in the repository. To tag all my libraries with a version name I can do:

```
cvs rtag LIBRARY_2_0 lib
```

This will recursively go through all the repository directories below lib and add the LIBRARY_2_0 tag to each file. This is one of the most useful features of CVS (IMHO). Use this feature if you about to release a copy of the files to the outside world or just want to mark a point in the developmental progression of the files.

cvs history
To find out information about your CVS repositories use the cvs history command. By default history will show you all the entries that correspond to you. Use the -a option to show information about everyone.

cvs history -a -o shows you (a)ll the checked (o)ut modules
cvs history -a -T reports (a)ll the r(T)ags for the modules
cvs history -a -e reports (a)ll the information about (e)verything

Kundestyrt prosjekt

Høst 2001

System-dokumentasjon

Versjon 1.03

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
05.11.2001	Opprettelse, skissering av innholdet	0.01	Martin
06.11.2001	La til info om brukergrensesnitt	0.05	Magnus
11.11.2001	Gjennomgang, og oppdatering	0.50	Martin
11.11.2001	Nå med kode!	0.60	Magnus
12.11.2001	Nå uten kode!! (og javadoc)	0.80	Martin
12.11.2001	La til installasjon	0.90	Magnus
12.11.2001	Utfyllende om installasjon og kjøring	0.91	Ole Kristian
13.11.2001	Fullført dokumentet og lest korrektur	1.0	Ole Kristian
13.11.2001	Korrekturlesing	1.01	Atle
13.11.2001	Feilmeldinger	1.03	Odd Christian

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dette dokumentet har som målsetting å lette bruk og eventuell videreutvikling av systemet ved å dokumenter de ulike aspektene av det implementerte systemet.

1.2 Avgrensning

Dette dokumentet dokumenterer kun akkurat det systemet som er implementert, og tar utgangpunkt i at strukturen på dataene ikke er endret i forhold til den originale CD-en.

1.3 Spesielle definisjoner

Se glossar [GLO]

1.4 Dokumentreferanser

[GLO] Glossar, Qpro16 2001

1.5 Dokumentoversikt

Dette dokumentet forklarer først hvordan programmet skal installeres og eventuelt rekompileres for videre utvidelser i kapittel 2.

For dagligdags bruk skildres en bruksveiledning i kapittel 3. Denne tar for seg hvordan regler skal spesifiseres og praktisk bruk av systemet. Her skildres også de feil som oftest oppstår og tiltak for å rette opp disse.

I kapittel 4 forklares selve systemdokumentasjon og hvilke former denne er oppgitt på.

2 Installasjon

Dette kapittelet forklarer hvordan systemet kan kjøres eller installeres på en maskin fra den vedlagte CDen. I tillegg forklarer den hvordan koden skal kompileres for videre utvikling. Beskrivelsene gjelder for en maskin med MS Windows operativsystem, men programmet kan kjøres på et hvilket som helst operativsystem som kan kjøre Java.

2.1 Kjøring av systemet fra CDen

Programmet kan enten kjøres direkte fra CDen, eller installeres lokalt på en maskin. Hvis det skal installeres på maskinen er det er fordel at Java SDK versjon 1.3 allerede er installert på denne. Eventuelt kan det velges å installere Java Runtime Environment sammen med programmet. Se kapittel 2.2, Installasjon, for hvordan installere programmet.

Systemet ligger i katalogen qpro16\ på CDen. Hvis java er installert på maskinen skal programmet normalt kunne startes ved å dobbelklikke på Qpro16.jar, eventuelt å gå til denne katalogen og skrive Qpro16 i kommandolinjeverktøyet.

Hvis jar-filer ikke er tilordnet java på den aktuelle maskinen vil den ovennevnte fremgangsmåten ikke fungere, og da kan det eventuelt kjøres ved å gå til katalogen qpro16\ og bruke kommandoen:

```
java -jar Qpro16.jar
```

Hvis det ikke er java installert på maskinen kan programmet startes med filen

```
run.bat
```

som kjører programmet ved hjelp av Java Runtime Environment som ligger på CDen.

Etter å ha brukt en av disse fremgangsmåtene kommer det grafiske brukergrensesnittet frem, og genereringen kan starte.

2.2 Installasjon fra CD

For at programmet skal fungere må Java SDKvære installert på maskinen som brukes. Eventuelt kan det brukes Java Runtime Environment. Denne inneholder java APIen og en javatolker. Med andre ord det som trengs for å kjøre et javaprogram, men ikke det som trengs for å kompilere et system. Java Runtime Environment ligger vedlagt på prosjekt-CDen.

For å installere kodegeneratoren kjører man bat-filen

```
install.bat
```

som ligger på prosjekt-CDen. Denne kjøres fra et kommandovindu, og man finner bat-filen i roten på CDen. Når installasjonen kjøres må katalogen hvor programmet skal installeres gies som inndata.

Kommandoen for å installere programmet blir for eksempel:

```
install c:\qpro16
```

Under installasjonen kan det velges om Java Runtime Environment skal taes med under installasjonen. Hvis det er java på maskinen fra før er dette ikke nødvendig.

Det som skjer under installasjonen er at filen Qpro16.jar, run.bat og de medfølgende regler og maler kopiert over til en underkatalog 'qpro16' til den valgte installasjonskatalogen. Hvis det er valgt å ta med Java Runtime Environment blir dette lagt i katalogen 'jre'. Så kan programmet startes etter tilsvarende fremgangsmåte som for kjøring fra CD. Se kapittel 2.1, Kjøring av systemet fra CD.

2.3 Kompilering

Hvis det har blitt foretatt forandringer i kildekoden er det nødvendig å kompilere systemet på nytt. Det kan da velges å kompilere kildekoden og kjøre den direkte, eller den kompilerte koden kan legges i en kjørbar jar-fil.

Først må man kompilere alle kodefilene. Dette gjøres slik man vanligvis kompilerer javafiler, men tredjeparts programpakkene `org` og `com` må ligge i classpathen under kompileringen. Disse pakkene ligger under `packages` katalogen på CD'en.

Eventuelt kan filen compile.bat som ligger i source-katalogen brukes. Dette forutsetter at kildekoden ligger organisert slik som på CDen, og at bat-filen ligger som på CDen i forhold til kildekoden. Ved bruk av bat-filen

Etter det kan det lages en jar-fil på følgende måte:

1. Filen `MainClass.mf` legges i den samme katalogen som det er kompilert til. I denne katalogen må også eksterne pakker (fra com og org) ligge. `MainClass.mf` - filen må inneholde referanse til startklassen: Hvis startklassen heter Start.class blir innholdet slik:

   ```
   Main-Class: Start
   ```

   Her er det viktig å ha et linjeskift etter denne linjen.

En ferdig slik fil som kan brukes ligger i source-katalogen på CDen.

2. Start et kommandovindu og gå til kompileringskatalogen.

3. Bruk jar-kommandoen til å opprette en JAR-fil.

   Denne kommandoen har syntaks

   ```
   jar [options] [files]
   ```

   I dette tilfellet vil da kommandoen bli:

   ```
   jar cmfv0 MainClass.mf Qpro16.jar GUI controller param
   error xmlparser org com Start.class
   ```

   der opsjonene står for:

   c – skape et nytt arkiv
   m – inkluderer manifest informasjon fra den spesifiserte manifest file
   f – spesifierer navnet på arkivfilen
   v – vil se all mulig informasjon fra prosessen
   0 – larger kun informasjonen, komprimerer ikke innholdet

   MainClass.mf angir startklassen (skal i manifestet)
   Qpro16.jar er målfilen
   Resten er pakkene og filene som man vil ha med

3 Brukerveiledning

Dette kapittelet gir en innføring i hvordan systemet skal brukes. Først hvordan maler og regler opprettes, og deretter hvordan disse blir brukt i systemet.

3.1 Oppretting av regler og maler

Før systemet kan startes må det være lagt inn regel- og malfiler. En beskrivelse av hvordan reglene og malene skal fungere finnes i Konstruksjonsdokumentet, kapittel 4. Her vil det imidlertid være en kokebokoppskrift på oppretting av disse filene.

Denne gjennomgangen vil bruke EJB som eksempel. Se vedlegg 1 og 2 i konstruksjonsdokumentet [KON] for kodeeksempler.

3.1.1 Malfilen

Malfilen er en flat tekstfil, og er veldig grei å opprette. For eksempel vil det i EJB måtte opprettes fire maler som er skjelettklasser. Det vil ofte være en del standard metoder og variabler som skal være med, og disse fører man rett inn i malene. Der det skal byttes ut info skriver man inn en variabel på formen `%VARIABEL%`, hvis '%' brukes som skilletegn (se reglene).

Malfilene plasseres så i katalogen '`templates`' under '`config`'.

3.1.2 Regelfilen

Regelfilen er en XML-fil som består av fem hovedelementer som alle må være med i filen. Disse fem er beskrevet i konstruksjonsdokumentet, kapittel 4, og det kapittelet må være tilgjengelig ved gjennomgang av oppskriften som følger under:

1. Først opprettes en ny fil og tagen `<RULEFILE>....</RULEFILE>` legges inn. Resten av elementene skal plasseres inne i denne tagen.
2. Deretter velges det hvilket skilletegn som skal være gyldig rundt variablene i malene.
3. Så må navnet til attributtet som inneholder IDen til hver node i XMI-treet angies.
4. Så legges det inn linker til hvilke maler denne regelfilen skal benytte for kodegenereringen.
5. Deretter lages inn interne maler som skal ligge i regelfilen.
6. Siste punkt er opprettingen av reglene. Disse kan være noe komplekse i starten. Det vil her komme en del eksempler på hvordan regler bygges opp.

Eksempel 1 (utdrag fra EJB-regelfil):
Det må angies navn for alle klasser. Dette navnet vil bli benyttet både i navnet på klassen, og nødvendigvis også som filnavn til kodefilen. Regelen vil se ut som følger:

```
<CLASSNAME type=”replace;filereplace” source=”
Foundation.Core.ModelElement.name” mandatory=”true” />
```

Navnet på regelen settes til `<CLASSNAME../>` fordi `%CLASSNAME%` vil bli brukt i malene der denne regelen skal bli brukt. Hvis de i malene hadde blitt brukt `%NAME%` måtte også denne regelen ha hatt navnet `<NAME../>`

Regelen er av type `'replace;filereplace'` fordi den både skal brukes til å forme de ferdige navnene på kodefilene, samt at den skal brukes til å bytte ut variabler i malene. Verdien til attributtet `'source'` er satt ut fra hva man vet finnes i XMI-treet. For å finne ut hva noden som inneholder den infoen denne regelen skal bruke heter må det letes i XMI-filen. I XMI 1.0 ligger navnet på en modell i noden `'Foundation.Core.ModelElement.name',` og dette er derfor brukt som verdi på attributtet `'source'`. Attributtet `'mandatory'` er satt til `'true'` fordi det må finnes en forekomst i malen som benytter denne regelen. Ellers vil ikke den ferdige koden være gyldig.

Eksempel 2(utdrag fra EJB-regelfil):
Her er et eksempel på hvordan det i reglene kan støttes innlegging av attributter:

```
<ATTRIBUTE type="multi"
source="Foundation.Core.Attribute"
template="ATTRIBUTE_TEMPLATE" mandatory="false"
separator="&#013;    ">
   <ATTRIBUTE_VISIBILITY type="replace" link=""
   source="Foundation.Core.ModelElement.visibility;
   xmi.value" target="%ATTRIBUTE_VISIBILTY%" />

   <ATTRIBUTE_TYPE type="replace"
   link="Foundation.Core.Classifier;xmi.idref"
   source="Foundation.Core.ModelElement.name"
   target="%ATTRIBUTE_TYPE%" />

   <ATTRIBUTE_NAME type="replace" link=""
   source="Foundation.Core.ModelElement.name"
   target="%ATTRIBUTE_NAME%"></ATTRIBUTE_NAME>

</ATTRIBUTE>
```

Dette er regelen for attributt i EJB. Attributter vil det som oftest være flere av, og derfor er regelen `<ATTRIBUTE ..>` av type `'multi'`. Det betyr at det kan være mange elementer i XMI-treet som skal behandles av denne regelen. Attributtet `'source'` har

verdien til den noden i XMI-treet som inneholder et undertre med info om en attributt. En regel med `'type=multi'` må alltid også ha attributtet `'template'`. I dette attributtet angis hvilken intern mal som skal benyttes. Verdien her refererer til et element under interne maler. I dette eksempelet ser vi også at `'mandatory'` er satt til 'false'. Det betyr at det ikke trenger å eksistere attributter i modellen for at den skal bli godkjent. Attributtet `'separator'` er også angitt. Verdien til dette attributtet vil bestemme hvilket tegn som skal skille hvert av attributtene. I dette tilfellet er det `'&#013;'`, som betyr linjeskift.

En regel av typen 'multi' vil alltid ha underregler. Vi ser her at `<ATTRIBUTE_VISIBILITY… />`, `<ATTRIBUTE_TYPE… />` og `<ATTRIBUTE_NAME… />` alle er underregler til `<ATTRIBUTE>`. Disse er alle av type `'replace'`, og skal bare hente info fra XMI-treet og erstatte i malen. Det som er litt spesielt med regelen `<ATTRIBUTE_TYPE.. />` er at den har et attributt som heter `'link'`. Dette attributtet brukes når det først må slåes opp i en node for å finne en verdi som peker til en annen node i treet, og deretter hente ut verdien i denne. Dette brukes i attributter fordi attributtypene i XMIen er representert med en intern kode, og koblingen mellom denne koden og hva det betyr er skilt ut fra selve attributtet og står et annet sted i XMI-filen. Det må derfor hentes ut en link til der typen er lagret. Den linken brukes så til å finne noden som inneholder navnet på typen til variabelen.

Regelfilen plasseres så i katalogen '`rules`' under '`config`'

3.2 Oppretting av arkitekturfiler

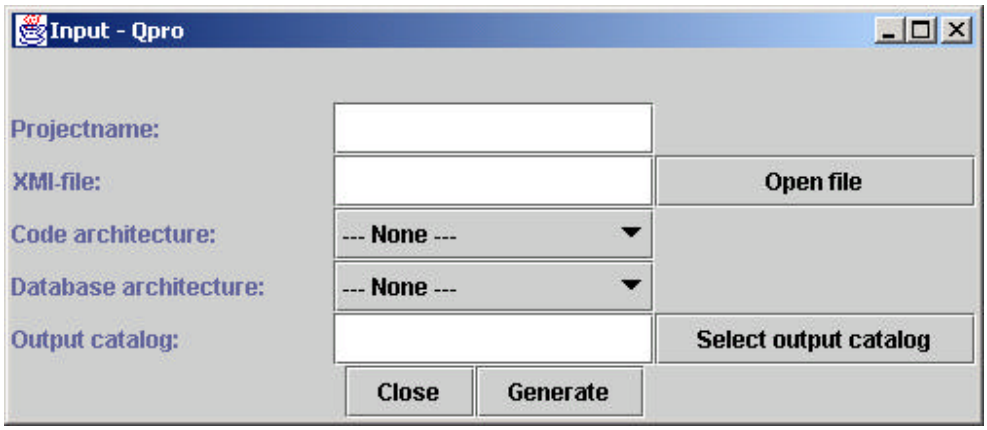
Det må utarbeides en XML-fil som inneholder alle arkitekturene systemet støtter. Filen skal hete 'archs.xml' og ligge i katalogen 'config'. Filen må inneholde navn på arkitektur, bane til regelfilen for arkitekturen, samt en id som må være unik. Det må også spesifiseres om det er en systemarkitektur eller en databasearkitektur. Filen kan se ut som følger:

```
<architectures>
   <sysarchitecture><!-- Angir at det er en
                        systemarkitektur -->
      <id>1</id>
      <name>ejb</name>
      <rulepath>config/rules/ejbRules.xml</rulepath>
   </sysarchitecture>
   <dbarchitecture><!-- Angir at det er en
                       databasearkitektur -->
      <id>101</id>
      <name>oracle</name>
      <rulepath>config/rules/oracleRules.xml</rulepath>
   </dbarchitecture>
</architectures>
```

Se forøvrig kapittel 3.1, Oppretting av regler og maler, for hvordan regelfiler opprettes.

3.3 Brukergrensesnittet

Brukergrensenittet som møter brukeren vil være grafisk og inneholde tekstfelt for å legge inn nødvendig informasjon. Teksten er engelsk i tilfelle verktøyet skal brukes av personer som ikke kan norsk. Brukergrensesnittet er laget med tanke på at systemet skal brukes til å generere kode ut i fra XMI-filer selv om det kan brukes mer generelt enn det. Brukergrensesnittet er vist i figur 1 under.



Figur 3.1 – Brukergrensesnitt

Forklaring til brukergrensesnittet:

Projectname	Her fylles navnet på prosjektet inn. Dette navnet vil brukes til å opprette loggfil.
XMI-file	Her angies banen til XML-filen som skal brukes som inndata under genereringen. Filnavnet kan skrives inn, eller filen kan velges ved å trykke på knappen 'Open file'. Da spretter det opp en fildialog som man kan bla i.
Code architecture	Her velges hvilken systemarkitektur det skal genereres kode for.
Database architecture	Her velges hvilken databasetype det skal genereres opprettelsesskript for.
Output catalog	Har angis hvilken katalog de ferdige filene skal lagres i. Katalogbanen kan skrives inn, eller katalogen kan velges ved å trykke på knappen 'Select output catalog'. Da spretter det opp en fildialog som man kan bla i. (Man må trykke to ganger på 'Open' for å velge katalog)
Close	Knapp for å avslutte systemet
Generate	Knapp for å starte genereringen. Etter å ha trykket denne kommer det opp et nytt vindu med en progresjonslinje sammen med info om hva som skjer.

Det vil ikke være mulig å starte genereringen før all nødvendig info er lagt inn. Det vil si at det eneste som ikke trenger å være fylt inn er den ene av arkitekturene.

3.4 Mulige feilmeldinger, og tiltak for å rette feilene

Hvis det under genereringen skulle dukke opp feilmeldinger vil disse bli vist på skjermen. Feilmeldingene har forklarende tekst, og feilene kan ligge i regler, maler eller i inndataene. Eventuelt kan en modell ikke inneholde de elementer som er påkrevd.

Ord i klammer, som f.eks. [link], er ord som varierer i feilmeldingene, ut ifra hvor feilen oppstod.

Feilmelding	"Wrong link: [malvariabel] [link]" "Wrong linkreference [link] in rule [rulenode]"
Handling	Sjekk at teksten i link-attributtet er på formen: *inputLinkNode;inputLinkAttribute* Sjekk source-attributtet i foreldrenoden til den aktuelle regelen. Source-attributtet peker på det aktuelle subtreet i inndataene. Sjekk at teksten i *link*-attributtet i aktuell regel peker på en node som finnes i det aktuelle subtreet. Sjekk at den eventuelle noden *link*-attributtet peker på, har det påkrevde attributtet (*inputLinkAttribute*).
Feilmelding	"Mandatory element [malvariabel] not in XMI"
Handling	Regelen som svarer til *malvariabel*en, sier at det skal finnes en node eller et subtre i inndatene som skal være utgangspunkt for innlegging av tekst inn på *malvariabel*ens sted i malen. Den noden som regelen referer til, finnes ikke. Det er enten noe galt med inndataene, eller så må mandatory-attributtet fjernes eller settes til false.
Feilmelding	"Can't find the textvalue for the xmi-node [id] referred to by the rule [rule]"
Handling	Sjekk at den type node som noden med identfikasjon *id* er ment å inneholde data. Kanskje må *rule* referere til en annen node.
Feilmelding	"Wrong format in restriction: [sourcerestriction]"
Handling	Sjekk at sourcerestriction-attributtet er brukt riktig, og har formatet: *NodeMedAttributt;AttributtMedVerdi;VerdiForAttributt*
Feilmelding	"Element referred to in restriction [sourcerestriction] doesn't exist: [element]"
Handling	*element* er en node i inndataene, og er referert til i et sourcerestriction-attributt. Nodenavnet i teksten til attributtet er feil, siden noden ikke finnes i subtreet referert til av sourceattributtet, og må endres.
Feilmelding	"Attribute referred to in restriction [sourcerestriction] doesn't exist:

	[attribute]"
Handling	Noden referert til i sourcerestriction finnes, men attributtet er ikke gitt på denne noden. Attributtnavnet i sourcerestriction-attributtet må derfor endres.
Feilmelding	"Invalid rulefile: [specification]"
Handling	Det mangler et eller annet i regelfilen, og *specification* forteller hva.
Feilmelding	"An error occured during building of the internal template [templateName]"
Handling	Sjekk at den interne malen i regelfilen med navn *templateName* er en OK mal. Hvis malen er tom er den ikke OK.
Feilmelding	"File [filepath] could not be found!"
Handling	Sjekk at *filepath* stemmer, og at filen er aksesserbar fra der programmet kjøres.
Feilmelding	"The name of the project must be specified."
Handling	Man må angi et navn på prosjektet som det skal genereres kode for.
Feilmelding	"Architecture file does't exist: [Bane til systemarkitekturfil]"
Handling	Regelfilen for spesifisert systemarkitektur eksisterer ikke. Man må opprette filen.
Feilmelding	"Architecture file cannot be read: [Bane til systemarkitekturfil]"
Handling	Regelfilen for spesifisert systemarkitektur kan ikke leses. Filen må gjøres skrivbar.
Feilmelding	"Architecture filename not specified."
Handling	Systemarkitektur må angis.
Feilmelding	"Database file does't exist: [Bane til databasearkitekturfil]"
Handling	Regelfilen for spesifisert databasearkitektur eksisterer ikke. Man må opprette filen.
Feilmelding	"Database file does't exist: [Bane til databasearkitekturfil]"
Handling	Regelfilen for spesifisert databasearkitektur kan ikke leses. Filen må gjøres skrivbar.
Feilmelding	"Database filename not specified."
Handling	Databasearkitektur må angis.
Feilmelding	"Cannot find the file: [bane til XMI-filen]“
Handling	Filen det refereres til er XMI-filen som inneholder modellen det skal genereres kode ut fra. Man må angi rett bane til filen.

Feilmelding	"Cannot parse XMI file. Reason: [Feilmelding fra parseren]“
Handling	Parseren kan ikke parse XMI filen. Grunnen blir også listet ut. Man må oppdatere XMI filen slik at den blir gyldig etter XML-standarden.
Feilmelding	"XMI filename not specified."
Handling	Navn på XMI-fil må angis.
Feilmelding	"Feil ved logging."
Handling	Det skjedde en feil under logging til fil. Trenger ikke ha noe å si for kjøringen. Kan skyldes at loggfil ikke kan opprettes pga manglende rettigheter.

4 Systemdokumentasjon

Systemet er videre dokumentert ved hjelp av javadoc som generes automatisk fra java-koden. Denne dokumentasjon beskriver alt som sees fra utsiden av de ulike klassene, noe som i praksis vil si alle offentlige metoder.

Alle metoder beskrives her med en beskrivelse av hva de gjør, hva de ulike parameterne som den tar inneholder, hva den eventuelt returnerer og hvilke unntak den kaster.

I tillegg er også kildekoden vedlagt, og ved å gå i denne kan alle indre løsninger i de ulike klassene betraktes.

For en utviklerer som skal ta over utviklingen av systemet anbefales det også at konstruksjonsdokumentet [KON] leses grundig før man starter.

5 Vedlegg

5.1 Vedlegg 1 – Javadoc og kildekode

Her følger javadoc og kildekode for Qpro16 kodegeneratoren. Begge delene har sin egen innholdsfortegnelse. Javadoc og kildekode ligger også på CDen under katalogene 'javadoc' og 'source'

JAVADOC starter på side 345

Kildekode starter på side 403

Innholdsfortegnelse for den genererte JAVADOC

The Qpro16 system

Package controller

Class CodeController

java.lang.Object
|
+--**controller.CodeController**

---

public class **CodeController**
extends java.lang.Object

Code Controller is responsible for building the code from the parsed XMI modell by using the method defined in the ProjectParameter interface.

Since:

25.10.2001

See Also:

ProjectParameter

CodeController

public **CodeController**(QproLog log)

Standard constructor.

Parameters:

log - Loggeobjekt

generate

public void **generate**(ProjectParameter params)

throws BuildException

Generates code using a ProjectParameter The main method of the code building . This method parses the XMI tree untill it finds a node that corresponds to an trigger. Then it uses 'useTemplate' method to use the corresponding template on the subtree of the matching node.

Parameters:

params - Collection of parameters that contain the configuration of the generation process

Throws:

BuildException - when there is an inconsistency between the rules, templates and XMI input.

Class ParamController

java.lang.Object
|
+--**controller.ParamController**

---

public class **ParamController**
extends java.lang.Object

ParamController validates the input to the system and creates the internal objects needed to represent them in the ProjectParameter.

Since:

25.10.2001

See Also:

ProjectParameter, QproParser, ProjectParameter, QproParser

ParamController

public **ParamController**(QproLog log)

Constructor for class. Takes a QproLog object to be able to log what happens

Parameters:

log - Log object

init

public ProjectParameter **init**(java.lang.String projectName,
java.lang.String pathToXMIFile,
java.lang.String srcDir,
Architecture chsSys,
Architecture chsDB)
throws ParamException

Takes all the parmaters that is to be validated as parameters. Returns the internal representation of the parameters as objects.

Parameters:

projectName - Name of the project

pathToXMIFile - Path to the XML file containing model in XMI-format

srcDir - Catalog to save generated code

chsSys - Chosen system architecture

chsDB - Chosen database architecture

Returns:

param.ProjectParameter

Throws:

ParamException - If validations failes, a ParamException is thrown

Class QproController

java.lang.Object
|
+--**controller.QproController**

---

public class **QproController**
extends java.lang.Object

The central controllobject of the system. This is the central controllobject in the system, and stand for all the communication between the controllerobjects of the system.

Since:
25.10.2001

See Also:
CodeController, ParamController

QproController

public **QproController**()
Standard constructor.

setValues

public void **setValues**(java.lang.String name,
java.lang.String xmiFile,
java.lang.String directory,
Architecture chsSys,
Architecture chsDB)

Called by the GUI when all the needed information has been collected. Starts ParamController and validates the parameters before the codegeneration is done by the CodeControlller.

Parameters:

name - The projectname

xmiFile - The XMI that describes the model, from which the code is generated

directory - The directory to put the generated code

chsSys - The chosen system architecture

chsDB - The chosen database architecture

Class QproLog

java.lang.Object
|
+--**controller.QproLog**

---

public class **QproLog**
extends java.lang.Object

Responsable for the loggwriting in the system

Since:
26.10.2001

QproLog

public **QproLog**(java.lang.String logFile,
java.lang.String srcDir)
throws LogException

InitCreates the logFile and the directory (if it does not exists).

Parameters:

logFile - containing the future name of the logFile

srcDir - conaining the source directory wherer the logFile is be put

Throws:

LogException - when an IOError or other unexpected error accoures.

log

public void **log**(java.lang.String content)

throws LogException

Writes the given content to the logfile

Parameters:

content - contains whatever that is to be written to the log

Throws:

LogException - if the Log has not been initialized, or IOERRORs accours.

Package error

Class BuildException

java.lang.Object
 |
 +--java.lang.Throwable
 |
 +--java.lang.Exception
 |
 +--error.QproException
 |
 +--**error.BuildException**

---

public class **BuildException**
extends QproException

An exception used by CodeController if any unknown error acc ourds during the building of the code.

Since:
26.10.2001

See Also:
Serialized Form

BuildException

public **BuildException**()

Default inheritaded contstructor

BuildException

public **BuildException**(java.lang.String msg)

Inheritaded constructor that takes an errrormsg and stores this as its e.getMessage() value.

Parameters:

msg - containing the error message of this Exception

See Also:

QproException

Class LogException

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--error.QproException
             |
             +--**error.LogException**

---

public class **LogException**
extends QproException

An exception used by QproLog if any exception accours during the logging in the system

Since:
26.10.2001

See Also:
Serialized Form

LogException

public **LogException**()

Default inheritaded contstructor

LogException

public **LogException**(java.lang.String msg)

Inheritaded constructor that takes an errrormsg and stores this as its e.getMessage() value.

Parameters:

msg - containing the error message of this Exception

See Also:

QproException

Class ParamException

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--error.QproException
             |
             +--**error.ParamException**

---

public class **ParamException**
extends QproException

An exception used by the Param package and the ParamController if any exception takes place during the validation or the building of the parameters.

Since:
26.10.2001

See Also:
Serialized Form

ParamException

public **ParamException**()

Default inheritaded contstructor

ParamException

public **ParamException**(java.lang.String msg)

Inheritaded constructor that takes an errrormsg and stores this as its e.getMessage() value.

Parameters:

msg - containing the error message of this Exception

See Also:

QproException

Class ParseException

java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--error.QproException
             |
             +--**error.ParseException**

---

public class **ParseException**
extends QproException

An excepetion used by the xmlparser pac kage if any error accours during the parsing of the file.

Since:
26.10.2001

See Also:
Serialized Form

ParseException

public **ParseException**()

Default inheritaded contstructor

ParseException

public **ParseException**(java.lang.String msg)

Inheritaded constructor that takes an errrormsg and stores this as its e.getMessage() value.

Parameters:

msg - containing the error message of this Exception

See Also:

QproException

Class QproException

java.lang.Object
 |
 +--java.lang.Throwable
 |
 +--java.lang.Exception
 |
 +--**error.QproException**

Direct Known Subclasses:
BuildException, LogException, ParamException, ParseException

---

public class **QproException**
extends java.lang.Exception

An general exception for the system, made to ease the errorhandling in latter extension of the system

Since:
26.10.2001

See Also:
Exception, Serialized Form

QproException

public **QproException**()

Default inheritaded contstructor

QproException

public **QproException**(java.lang.String msg)

Inheritaded constructor that takes an errrormsg and stores this as its e.getMessage() value.

Parameters:

msg - containing the error message of this Exception

Package GUI

Interface QproGUI

All Known Implementing Classes:

MainWindow

---

public interface **QproGUI**

Defines the grapical user interface of the system.

Since:

25.10.2001

See Also:

MainWindow

---

init

public void **init**(java.util.ArrayList sysArcs,
java.util.ArrayList DBArcs,
QproController controller)

This method generates architecture lists, adds actionlistener and starts the GUI.

Parameters:

sysArcs - Chosen system architecture

DBArcs - Chosen database architecture

controller - Takes the QproController that started the GUI.

---

updateState

public void **updateState**(java.lang.String text)

Method that shows text in the GUI while generating

Parameters:

text - The text to be shown

---

exit

public void **exit**()

Method to exit the system. Is used if the user chooses cansel or closes the GUI.

Class MainWindow

java.lang.Object
|
+--**GUI.MainWindow**

---

public class **MainWindow**
extends java.lang.Object
implements QproGUI

An implementation of the GUI interface of the Qpro system

Since:

25.10.2001

See Also:

QproGUI, MainWindowListener

MainWindow

public **MainWindow**()

Empty constructor for class.

Since:

25.10.2001

init

public void **init**(java.util.ArrayList sysArcs,
java.util.ArrayList DBArcs,
QproController controller)

This method generates architecture lists, adds actionlistener and starts the GUI.

Specified by:

init in interface QproGUI

Parameters:

sysArcs - Chosen system architecture

DBArcs - Chosen database architecture

controller - Takes the QproController that started the GUI.

---

updateState

public void **updateState**(java.lang.String text)

Method that shows text in the GUI while generating

Specified by:

updateState in interface QproGUI

Parameters:

text - The text to be shown

---

generate

public void **generate**()

Method to start the generation of code.

---

back

public void **back**()

Methos that opens the filedialog needed to choose XML-file

---

openXMIfileDialog

public void **openXMIfileDialog**()

---

openCatalogDialog

public void **openCatalogDialog**()

Methos that opens the filechooser needed to choose output catalog

---

exit

public void **exit**()

Method to exit the system. Is used if the user chooses cansel or closes the GUI.

Specified by:

exit in interface QproGUI

Class MainWindowListener

java.lang.Object
|
+--**GUI.MainWindowListener**

---

public class **MainWindowListener**
extends java.lang.Object
implements java.awt.event.ActionListener, java.awt.event.WindowListener

Listener for the GUI

Since:

25.10.2001

See Also:

MainWindow

---

action

public java.lang.String **action**

The action performed

MainWindowListener

public **MainWindowListener**(MainWindow qCon)

Constructor for class. Takes the GUI-object as input

Parameters:

qCon - GUI-object

actionPerformed

public void **actionPerformed**(java.awt.event.ActionEvent e)

This metod takes right action based on what the user does in the GUI.

Specified by:

actionPerformed in interface java.awt.event.ActionListener

Parameters:

e - Action that is performed.

---

windowActivated

public void **windowActivated**(java.awt.event.WindowEvent e)

windowActivated.

Specified by:

windowActivated in interface java.awt.event.WindowListener

---

windowClosed

public void **windowClosed**(java.awt.event.WindowEvent e)

windowClosed.

Specified by:

windowClosed in interface java.awt.event.WindowListener

---

windowClosing

public void **windowClosing**(java.awt.event.WindowEvent e)

windowClosing.

Specified by:

windowClosing in interface java.awt.event.WindowListener

---

windowDeactivated

public void **windowDeactivated**(java.awt.event.WindowEvent e)

windowDeactivated method comment.

Specified by:

windowDeactivated in interface java.awt.event.WindowListener

---

windowDeiconified

public void **windowDeiconified**(java.awt.event.WindowEvent e)

windowDeiconified.

Specified by:

windowDeiconified in interface java.awt.event.WindowListener

---

windowIconified

public void **windowIconified**(java.awt.event.WindowEvent e)

windowIconified.

Specified by:

windowIconified in interface java.awt.event.WindowListener

---

windowOpened

public void **windowOpened**(java.awt.event.WindowEvent e)

windowOpened.

Specified by:

windowOpened in interface java.awt.event.WindowListener

Package param

Interface ProjectParameter

All Known Implementing Classes:
ProjectParameterImpl

---

public interface **ProjectParameter**

A interface for the internal representation of all the parameters in the system. This interfaces defines all the methods that the architecture rules and te mplates must contain.

Since:
25.10.2001

See Also:
ProjectParameterImpl

---

	init(java.lang.String proName, Architecture chsSys,

init

public void **init**(java.lang.String proName,
Architecture chsSys,
Architecture chsDB,
java.lang.String srcDir,
com.sun.xml.tree.XmlDocument xmiRoot)
throws ParamException

Method that saves the params in intern variables, and gets the paths to the architecture files needed for the generation

Parameters:

proName - Name of the project

chsSys - Chosen system architecture

chsDB - Chosen database architecture

srcDir - Catalog where generated code will be saved

xmiDoc - The parsed XML document, containing the model

---

getXMIDocument

public com.sun.xml.tree.XmlDocument **getXMIDocument**()

Method that returns the parsed XML as a XmlDocument object

Returns:

Parsed XML docuement

---

getProjectName

public java.lang.String **getProjectName**()

Returns the name of the project.

Returns:

The project name.

---

getSrcDir

public java.lang.String **getSrcDir**()

Returns output directory for the generated code.

Returns:

The output directory.

---

getArchitecture

public Architecture **getArchitecture**(boolean database)

Returns the chosen architecture. You can get database- and systemarchitecture.

Parameters:

database - Boolean value used to choose either system architecture or database architecture

Returns:

The architecture

---

isTrigger

public boolean **isTrigger**(java.lang.String elementName,
boolean database)

Returns true or false if a element is trigger or not.

Parameters:

elementName - An element in the model

database - Boolean value used to choose either system architecture or database architecture

Returns:

If the element is trigger or not

---

getTemplatesByTrigger

public java.util.ArrayList **getTemplatesByTrigger**(java.lang.String elementName,
boolean database)

Method that returns an ArrayList containing all templates to be used on the an element.

Parameters:

elementName - An element in the model

database - Boolean value used to choose either system architecture or database architecture

Returns:

An ArrayList containing the templates to be used on the element

---

getRuleElement

public org.w3c.dom.Element **getRuleElement**(java.lang.String tag,
boolean database)

Returns an Elementobject containing a rule based on what tag you send in.

Parameters:

tag - Tag describing the name of a rule

database - Boolean value used to choose either system architecture or database architecture

Returns:

An element containing a rule

---

getInternalTemplateByName

public Template **getInternalTemplateByName**(java.lang.String name,
boolean database)

Returns an internal template from the ruleobject.

Parameters:

name - The name of an template

database - Boolean value used to choose either system architecture or database architecture

Returns:

The template

---

getIdAttributeName

public java.lang.String **getIdAttributeName**(boolean database)

Gets the name of the idattribute that vil be used to serach in the tree by id.

Returns:

String containing the name of the attribute containing an id in XML

Class Architecture

java.lang.Object
|
+--**param.Architecture**

---

public class **Architecture**
extends java.lang.Object

The objectrepresentation of an architecture with id, navn, og path to the corresponding rulefile.

Since:
04.11.2001

Architecture

public **Architecture**(int id,
java.lang.String name,
java.lang.String rulePath)

Constructor for the Architecture-class.

Parameters:
id - Identification-number
name - Name of the architecture
rulePath - Path to the rules-file

getPathToRuleFile

public java.lang.String **getPathToRuleFile**()
Returns the path to the rules-file
Returns:
Path to the rules-file

---

getArchitectureName

public java.lang.String **getArchitectureName**()
Returns the name of the architevture to be used
Returns:
Name of the architecture

---

getArchitectureID

public int **getArchitectureID**()
Returns the id of the architecture
Returns:
The id of the architecture

Class ProjectParameterImpl

java.lang.Object
|
+--**param.ProjectParameterImpl**

---

public class **ProjectParameterImpl**
extends java.lang.Object
implements ProjectParameter

A agregation class that implements the ProjectParameter inteface. Solves the methods by instances of the classes Rules and Architecture.

Since:

25.10.2001

See Also:

ProjectParameter, Rules

---

	getTemplatesByTrigger(java.lang.String elementName,

ProjectParameterImpl

public **ProjectParameterImpl**()

Emty constructor

init

public void **init**(java.lang.String proName,
Architecture chsSys,
Architecture chsDB,
java.lang.String srcDir,
com.sun.xml.tree.XmlDocument xmlDoc)
throws ParamException

Method that saves the params in intern variables, and gets the paths to the architecture files needed for the generation

Specified by:

init in interface ProjectParameter

Parameters:

proName - Name of the project

chsSys - Chosen system architecture

chsDB - Chosen database architecture

srcDir - Catalog where generated code will be saved

xmiDoc - The parsed XML document, containing the model

---

getXMIDocument

public com.sun.xml.tree.XmlDocument **getXMIDocument**()

Method that returns the parsed XML as a XmlDocument object

Specified by:

getXMIDocument in interface ProjectParameter

Returns:

Parsed XML docuement

---

getProjectName

public java.lang.String **getProjectName**()

Returns the name of the project.

Specified by:

getProjectName in interface ProjectParameter

Returns:

The project name.

---

getSrcDir

public java.lang.String **getSrcDir**()

Returns output directory for the generated code.

Specified by:

getSrcDir in interface ProjectParameter

Returns:

The output directory.

---

getArchitecture

public Architecture **getArchitecture**(boolean database)

Returns the chosen architecture. You can get database- and systemarchitecture.

Specified by:

getArchitecture in interface ProjectParameter

Parameters:

database - Boolean value used to choose either system architecture or database architecture

Returns:

The architecture

---

isTrigger

public boolean **isTrigger**(java.lang.String elementName,
boolean database)

Returns true or false if a element is trigger or not.

Specified by:

isTrigger in interface ProjectParameter

Parameters:

elementName - An element in the model

database - Boolean value used to choose either system architecture or database architecture

Returns:

If the element is trigger or not

---

getTemplatesByTrigger

public java.util.ArrayList **getTemplatesByTrigger**(java.lang.String elementName,
boolean database)

Method that returns an ArrayList containing all templates to be used on the an element.

Specified by:

getTemplatesByTrigger in interface ProjectParameter

Parameters:

elementName - An element in the model

database - Boolean value used to choose either system architecture or database architecture

Returns:

An ArrayList containing the templates to be used on the element

---

getRuleElement

public org.w3c.dom.Element **getRuleElement**(java.lang.String tag,
boolean database)

Returns an Elementobject containing a rule based on what tag you send in.

Specified by:

getRuleElement in interface ProjectParameter

Parameters:

tag - Tag describing the name of a rule

database - Boolean value used to choose either system architecture or database architecture

Returns:

An element containing a rule

---

getInternalTemplateByName

public Template **getInternalTemplateByName**(java.lang.String name,
boolean database)

Returns an internal template from the ruleobject.

Specified by:

getInternalTemplateByName in interface ProjectParameter

Parameters:

name - The name of an template

database - Boolean value used to choose either system architecture or database architecture

Returns:

The template

---

getIdAttributeName

public java.lang.String **getIdAttributeName**(boolean database)

Gets the name of the idattribute that vil be used to serach in the tree by id.

Specified by:

getIdAttributeName in interface ProjectParameter

Returns:

String containing the name of the attribute containing an id in XML

Class Rules

java.lang.Object
|
+--**param.Rules**

---

public class **Rules**
extends java.lang.Object

An objectrepresentation of the Rule-file.

Since:
07.11.2001

Rules

public **Rules**(java.lang.String pathToRuleFile)

throws ParamException

Constructor for the Rules-object. The constructor is building the rules-object based on the rules in the given rule-file.

Parameters:

pathToRuleFile - A string containing the path to the rule-file

isTrigger

public boolean **isTrigger**(java.lang.String elementName)

Method for checking whether a string is a trigger in the rules.

Parameters:

elementName - String to be checked

Returns:

Boolean value telling whether elementName is a trigger

getTemplatesByTrigger

public java.util.ArrayList **getTemplatesByTrigger**(java.lang.String elementName)

Returns an ArrayList containing the templates with the trigger elementName. null will be returned if no trigger with the given name exists .

Parameters:

elementName - The trigger

Returns:

ArrayList containing all the templates with the given trigger

See Also:

Template

getRuleElement

public org.w3c.dom.Element **getRuleElement**(java.lang.String tag)

Returns the root-element of the rule with the given tree.

Parameters:

tag - Name of the tag

Returns:

Root-element of the rule with the given tag

See Also:

Element

getInternalTemplateByName

public Template **getInternalTemplateByName**(java.lang.String name)

Returns the internal template with the given name.

Parameters:

name - Name of the internal template to be returned

Returns:

Internal template with the given name

getIdAttributeName

public java.lang.String **getIdAttributeName**()

Returns a string containing the name of the attribute containing an id in XML.

Returns:

String containing the name of the attribute containing an id in XML

Class Template

java.lang.Object
|
+--**param.Template**

---

public class **Template**
extends java.lang.Object

The objectrepresentation of a templatefile or an internal template in the rules. Got the method getNextToken that returns the next fragment of the template in question.

Since:
05.11.2001

Template

public **Template**(java.lang.String internalTemplate,
java.lang.String delimiter)

Constructor that builds the template-object for an internal template.

Parameters:

internalTemplate - String containing the internal template

delimiter - One-character-string that is used to mark a variable in the template

---

Template

public **Template**(java.lang.String pathToFile,
java.lang.String target,
java.lang.String delimiter)
throws ParamException

Constructor that builds the template-object for a template in a file.

Parameters:

pathToFile - Path to the template-file

target - Filename where to save the finished file

delimiter - One-character-string that is used to mark a variable in the template

Throws:

ParamException -

getNextToken

public TemplateToken **getNextToken**()

Method that returns the next token in the template. The token is either static text or a variable/parameter that is to be replaced. The character that marks the variables will not be returned.

Returns:

The next token in the template

See Also:

TemplateToken

---

getTarget

public java.lang.String **getTarget**(java.lang.String variable,
java.lang.String newValue)

Method that returns the target (filename) for the template. Replaces the variable in the filename with the given value. This method can only replace one variable in a filename. If there are more than one variable to be replaced the others will remain in the filename, and so will the character that marks the variable. This means that more than one variable in a target probably will cause the program to fail.

Parameters:

variable - Variable to be replaced

newValue - The value to replace the variable with

Returns:

The filename for the template

---

hasMoreTokens

public boolean **hasMoreTokens**()

Returns whether the template has more tokens. It will return true after the last token if the last token was not static, but invoking the getNextToken()-method is safe since it in that case will return an empty string.

Returns:

Boolean value that tells whether the template has more tokens

See Also:

TemplateToken

---

isInternal

public boolean **isInternal**()

Returns whether the template is internal.

Returns:

Boolean value that tells whether the template is internal

---

reset

public void **reset**()

Resets an internal template. After running this method the getNextToken()- method will return the first token in the template. Running this method on a template that is not internal will cause a ParamException to be thrown.

Class TemplateToken

java.lang.Object
|
+--**param.TemplateToken**

---

public class **TemplateToken**
extends java.lang.Object

This class is an encapsulation of an token in the template. This token is either a static text or a parameter that must be replaced using the related rules.

Since:

23.10.2001

See Also:

Template

TemplateToken

public **TemplateToken**(boolean isStatic,
java.lang.String tokenValue)

Constructor for the TemplateToken

Parameters:

isStatic -

tokenValue -

isStatic

public boolean **isStatic**()

Tells whether the TemplateToken is static or wheter it should be replaced.

Returns:

Boolean that tells whether the TemplateToken i static

---

getValue

public java.lang.String **getValue**()

Returns a String containing the text in the TemplateToken.

Returns:

String with the text in the TemplateToken

Package xmlparser

Interface QproParser

All Known Implementing Classes:

QproParserImpl

---

public interface **QproParser**

Defines all the methods of XML parsing and utils that is needed in the system.

Since:

26.10.2001

---

parseXML

public com.sun.xml.tree.XmlDocument **parseXML**(java.lang.String pathToXMLFile)
throws ParseException

Parses an XML file into an com.sun.xml.tree.XMLDocument object.

Parameters:

pathToXMLFile - a String containing a valid path to an XMI file

Returns:

com.sun.xml.tree.XmlDocument

Throws:

ParseException - containing a errormsg what went wrong.

---

writeTree

public java.lang.String **writeTree**(org.w3c.dom.Element root)

Returns a textual representation of the subtree in the element specified.

Parameters:

root - an Element containing the rootNode of the subtree

Returns:

String the tree in textual representation

Class QproParserImpl

java.lang.Object
|
+--**xml|parser.QproParserImpl**

---

public class **QproParserImpl**
extends java.lang.Object
implements QproParser

This class is an realization of the QproParser interface.

Since:
26.10.2001

QproParserImpl

public **QproParserImpl**()
Default contstructor

parseXML

public com.sun.xml.tree.XmlDocument **parseXML**(java.lang.String pathToXMLFile)
throws ParseException

The implementation of the parseXML method in the QproParser interface

Specified by:

parseXML in interface QproParser

See Also:

QproParser

---

writeTree

public java.lang.String **writeTree**(org.w3c.dom.Element e)

The implementation of the writeTree method in the QproParser interface

Specified by:

writeTree in interface QproParser

See Also:

QproParser

Innholdsfortegnelse for kildekoden

1.1.1 Start

```
import controller.QproController;

public class Start {


    public static void main(String[] args) {
        QproController qpro = new QproController();
    }
}
```

1.1.2 CodeController

```
package controller;

import com.sun.xml.tree.XmlDocument;
import error.BuildException;
import error.LogException;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import param.Architecture;
import param.ProjectParameter;
import param.Template;
import param.TemplateToken;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.NoSuchElementException;
import java.util.StringTokenizer;

/**
 * Code Controller is responsible for building the code from the parsed XMI modell
 * by using the method defined in the ProjectParameter interface.
 *
 * @author Qpro16 - konstruksjon
 * @version 0.1
 * @see param.ProjectParameter
 * @since 25.10.2001
 */
public class CodeController {
    private ProjectParameter params;
    private Architecture arch;
    private StringBuffer codeResult;
    private String targetFileName;
    private QproLog qproLog;

    /**Standard constructor.
     * @param log Loggeobjekt
     */
    public CodeController(QproLog log) {
        params = null;
        arch = null;
        codeResult = new StringBuffer();
        targetFileName = null;
        this.qproLog = log;
    }

    /**Generates code using a ProjectParameter
     * The main method of the code building . This method parses the XMI tree untill
     * it finds a node that corresponds to an trigger. Then it uses 'useTemplate'
     * method to use the corresponding template on the subtree of the matching node.
     * @param params Collection of parameters that contain the configuration of the generation process
     * @throws BuildException when there is an inconsistency between the rules, templates and XMI input.
     */
    public void generate(ProjectParameter params) throws BuildException {
        try {
            qproLog.log("****************** CodeController.generate() started ********************\n\n");
            this.params = params;
            //Retrieves the XMI-document, which is the basis for code generation
            XmlDocument doc = params.getXMIDocument();
            Element root = doc.getDocumentElement();
            //Generates code for both system- and databasearchitectures, if they exist
```

```
            qproLog.log("System architecture\n");
             arch = params.getArchitecture(false);
            if (arch !=null)
                generateByElement(root, false, -1);
             qproLog.log("Database architecture\n");
             arch = params.getArchitecture(true);
            if (arch !=null)
                generateByElement(root, true, -1);
             qproLog.log("****************** CodeController.generate() ended ******************\n\n");
        }catch (LogException e) {
            throw new BuildException(e.getMessage());
        }
    }

    /**Generates code starting at a rootnode.
     * A recursive method that searches for triggernodes in the XMI input.
     * Used by generate.
     * @param root The Node at which the generation begins.
     * @param isDatabase True if the desired output is databasescripts
     * @param level The level in the XMI input the recursion has reached
     * @throws BuildException when there is an inconsistency between the rules, templates and XMI input.
     * @throws LogException if the logging fails
     */
    private void generateByElement(Node root, boolean isDatabase, int level) throws BuildException,
LogException {
        level++;
        StringBuffer spaces = new StringBuffer();
        for (int i = 0; i < level; i++)
            spaces.append("  ");

        NodeList childNodes = root.getChildNodes();
        int length = childNodes.getLength();
        //Walks through all children of the rootnode
        for (int i = 0; i < length; i++) {
            Node node = childNodes.item(i);
             qproLog.log(spaces.toString() + node.getNodeName());
            //Only elementnodes and their children contain interesting information
            if (node.getNodeType() == node.ELEMENT_NODE) {
                Element e = (Element) childNodes.item(i);
                //Checks if the element from the XMI is defined as a trigger i the rulefile
                if (params.isTrigger(e.getNodeName(), isDatabase)) {
                    qproLog.log("TRIGGER found");
                    ArrayList templates = params.getTemplatesByTrigger(e.getNodeName(), isDatabase);
                int nrOfTemps = templates.size();
                //Generates code for all templates associated with the triggerelement
                for (int ii = 0; ii < nrOfTemps; ii++) {
                    Template temp = (Template) templates.get(ii);
                      useTemplate(e, temp, isDatabase);
                      temp.reset();
                    }
                }else
                //Recursive call that goes deeper into the XMItree
                    generateByElement(e, isDatabase, level);
            }//if !text_node
        }//for
    }

    /**This method uses a given template on the given subtree in the element according to the rules.
     * The method walks through the template, and encounters both static and
     * non-static tokens in the template. Static tokens are kept and later written directly
     * to a file. Non-static tokens are pointers to rules in the rulefile. Using the rules,
     * this method can decide what should be written to the file, instead of the non-static tokens.
     * @param root The root of the XMI subtree that contains the data to be used in conjunction with template
```

```
     * @param template The template with the static and non-static tokens
     * @isDatabase True if the desired output is databasescripts
     * @throws BuildException when there is an inconsistency between the rules, templates and XMI input.
     */
    private void useTemplate(Element root, Template template, boolean isDatabase) throws BuildException {
        //This method fills the instance variable codeResult with data from the template.
        //The template is divided into tokens, static and non-static, and this loop walks throug all of them
        while (template.hasMoreTokens()) {
            TemplateToken token = template.getNextToken();

            //TemplateToken token = template.getNextToken();
            //If the token is static, the text in the token is kept in codeResult
            //If not the token is static, a rule will be used to find the text to put in codeResult.
            //The rule is found using the value of the non-static token.
            Element rule = params.getRuleElement(token.getValue(), isDatabase);
            if (rule == null && !token.isStatic()) {
                try {
                    qproLog.log("\nWarning: There exists no rule for the " + token.getValue() + " token\n");
                } catch (LogException e) {
                    throw new BuildException(e.getMessage());
                }
            }

            if (token.isStatic()) {
                codeResult.append(token.getValue());
            } else if (rule != null) {
                String xmiTextValue = null;//The value to be put in codeResult

                //There are three essential attributes in the rule:
                String type_attr = rule.getAttribute("type");
                String source = rule.getAttribute("source");
                String link = rule.getAttribute("link");
                String mandatory = rule.getAttribute("mandatory");
                if (mandatory == null || mandatory.equals(""))
                    mandatory = "false";

                //Declares the xmi source of the wanted value
                String sourceNodeName = null;
                String sourceAttrName = null;
                int seperatorIndex_source = source.indexOf("/");
                sourceNodeName = null;
                sourceAttrName = null;
                if (seperatorIndex_source == -1)
                    sourceNodeName = source;
                else {
                    sourceNodeName = source.substring(0, seperatorIndex_source);
                    sourceAttrName = source.substring(seperatorIndex_source + 1);
                }

                //Uses the source and link attributes to find the node(s) in the XMI document
                //that contain the text to put in codeResult.
                //If the text to be put in codeResult is found in an attribute at the sourcenode
                //xmiTextValue is updated.
                NodeList xmiSourceNodes = null;
                if (link != null && !link.equals("")) {
                    int seperatorIndex_link = link.indexOf("/");
                    String linkNodeName = null;
                    String linkAttrName = null;
                    if (seperatorIndex_link == -1)
                        linkNodeName = link;
                    else {
                        linkNodeName = link.substring(0, seperatorIndex_link);
                        linkAttrName = link.substring(seperatorIndex_link + 1);
```

```
                }

                NodeList xmiLinkNodes = root.getElementsByTagName(linkNodeName);
                Element linkNode = (Element) xmiLinkNodes.item(0);
              if (linkNode == null)
                throw new BuildException("Wrong linkreference " + link + " in rule " + rule.getNodeName());
              String idRef = null;
              if (linkAttrName != null && !linkAttrName.trim().equals(""))
                  idRef = linkNode.getAttribute(linkAttrName);
              else {
                  Node textNode = linkNode.getFirstChild();
                if (textNode != null)
                    idRef = textNode.getNodeValue();
                }
                XmlDocument doc = params.getXMIDocument();
                Element refNode = getElementById(doc.getDocumentElement(), idRef, isDatabase);
              if (refNode == null)
                throw new BuildException("Wrong link: " + token.getValue() + " " + link);
                xmiSourceNodes = refNode.getElementsByTagName(sourceNodeName);
             } else
                xmiSourceNodes = root.getElementsByTagName(sourceNodeName);

            //Depending on the the type of the rule, the text to put in codeResult is retrieved
            StringTokenizer types = new StringTokenizer(type_attr, ";");
            while (types.hasMoreTokens()) {
                String type = types.nextToken();
              if (type.equals("replace") || type.equals("filereplace")) {
                  Node xmiNode = xmiSourceNodes.item(0);       //regner med at den nærmeste, og dermed
første(?), er den riktige noden
                if (xmiNode == null && mandatory == "true")
                  throw new BuildException("Mandatory element " + token.getValue() + " not in XMI");
                if (sourceAttrName == null) {
                    Node xmiTextNode = xmiNode.getFirstChild();
                  if (xmiTextNode == null) {
                      String id = "";
                    if (xmiNode.getNodeType() == Node.ELEMENT_NODE) {
                      Element el = (Element) xmiNode;
                        id = el.getAttribute(params.getIdAttributeName(isDatabase));
                      }
                    throw new BuildException("Can't find the textvalue for the xmi-node " + id + " referred to by
the rule " + rule.getNodeName());
                    }
                    xmiTextValue = xmiTextNode.getNodeValue();
                  } else
                    xmiTextValue = ((Element) xmiNode).getAttribute(sourceAttrName);//certain possibility for
classcastexc.

                String map = rule.getAttribute("map");
                if (map != null && !map.equals("")) {
                    xmiTextValue = mapTextValue(map, xmiTextValue, isDatabase);
                }

                //This is a very special case. This ruletype defines the final construction of the filename
                //to be used for this template.
                if (type.equals("filereplace"))
                    targetFileName = template.getTarget(token.getValue(),
xmiTextValue);//subStringReplace(template.getTarget(),token.getValue(),xmiTextNode.getNodeValue());
                else
                  codeResult.append(xmiTextValue);
              } else if (type.equals("multi")) {
                  String internalTemplateName = rule.getAttribute("template");
                  Template internalTemp = params.getInternalTemplateByName(internalTemplateName,
isDatabase);
```

```
                int length = xmiSourceNodes.getLength();
                //Counts the number of elements that are valid by restriction
                int nrOfValids =0;
                for (int i = 0; i < length; i++) {
                    Element xmiSubTree = (Element) xmiSourceNodes.item(i);
                    if (validByRestriction(xmiSubTree, rule.getAttribute("sourcerestriction")))
                        nrOfValids++;
                  }
                //Uses the internal template on all corresponding xmiNodes
                for (int i = 0; i < length; i++) {
                    Element xmiSubTree = (Element) xmiSourceNodes.item(i);
                    if (validByRestriction(xmiSubTree, rule.getAttribute("sourcerestriction"))) {
                        useTemplate(xmiSubTree, internalTemp, isDatabase);
                       internalTemp.reset();
                        String separator = rule.getAttribute("separator");
                      if (separator != null && !separator.equals("") && i < nrOfValids - 1)
                        codeResult.append(separator);
                    }
                 }//for ...use of internal templates
              }//if (type == ...)
            }//while (types.hasMore...)
        }//if (token.isStatic())
    }//while (template.hasMore...)

    //If not the template is internal, the codeResult should be written to file
    if (!template.isInternal()) {
        writeToFile(codeResult.toString(), targetFileName);
        codeResult.delete(0, codeResult.length());
       try {
            qproLog.log("----------Wrote code to "+ targetFileName +"----------");
        }catch (LogException e) {
          throw new BuildException(e.getMessage());
        }
    }
}

/**Writes a string to a file.
 * @param fileContent The string to be written to file
 * @param fileName The name of the file
 * @throws BuildException when the filewriting fails
 */
private void writeToFile(String fileContent, String fileName) throws BuildException {
    String dir = params.getSrcDir();
    String path = dir +"\\" + fileName;
   try {
        FileWriter writer = new FileWriter(path);
        writer.write(fileContent);
        writer.flush();
    }catch (IOException e) {
      throw new BuildException(e.getMessage());
    }
}


private Element getElementById(Node root, String idRef, boolean isDatabase) throws BuildException{
    Element resultElement = null;
    String idAttrName = params.getIdAttributeName(isDatabase);

   if (root.getNodeType() == root.ELEMENT_NODE) {
        String id = ((Element) root).getAttribute(idAttrName);
       if (id.equals(idRef))
            resultElement = (Element) root;
    }
```

```
        if (resultElement == null) {
            NodeList children = root.getChildNodes();
            Node currentNode = null;
            int i = 0;
            if (children != null) {
                while (i < children.getLength() && resultElement == null) {
                    currentNode = children.item(i);
                    resultElement = getElementById(currentNode, idRef, isDatabase);
                    i++;
                }
            }
        }
        return resultElement;
    }


    private String mapTextValue(String mapReference, String oldValue, boolean isDatabase) {
        int sepIndex = mapReference.indexOf(';');
        String mapSeriesTag = mapReference.substring(0, sepIndex);
        String mapTag = mapReference.substring(sepIndex + 1);

        String newValue = oldValue;
        Element mapRuleEl = params.getRuleElement(mapSeriesTag, isDatabase);
        NodeList mapNodes = mapRuleEl.getElementsByTagName(mapTag);
        for (int i = 0; i < mapNodes.getLength(); i++) {
            Node node = mapNodes.item(i);
            if (node.getNodeType() == node.ELEMENT_NODE) {
                Element map = (Element) node;
                if (map.getAttribute("old").equals(oldValue))
                    newValue = map.getAttribute("new");
            }
        }
        return newValue;
    }

    /**Checks if the element referred to in the given restriction, has got a desired value a given attribute.
     * @param subTree The portion of the tree that should conatain the element referred to in restriction
     * @param restriction Contains Elementname, attributename and desired value of the attribute
     *
     */
    private boolean validByRestriction(Element subTree, String restriction) throws BuildException {
        boolean valid = true;
        if (restriction != null && !restriction.equals("")) {
            StringTokenizer restrTokens = new StringTokenizer(restriction, ";");
            String elementName = null;
            String attrName = null;
            String attrValue = null;
            try {
                elementName = restrTokens.nextToken();
                attrName = restrTokens.nextToken();
                attrValue = restrTokens.nextToken();
            } catch (NoSuchElementException e) {
                throw new BuildException("Wrong format in restriction: " + restriction);
            }
            NodeList xmiElements = subTree.getElementsByTagName(elementName);
            if (xmiElements.item(0) == null || xmiElements.item(0).getNodeType() != Node.ELEMENT_NODE)
                throw new BuildException("Element referred to in restriction " + restriction + " doesn't exist: " +
elementName);
            Element restrEl = (Element) xmiElements.item(0);
            String actualValue = restrEl.getAttribute(attrName);
            if (actualValue == null)
                throw new BuildException("Attribute referred to in restriction " + restriction + " doesn't exist: " +
attrName);
```

```
            if (actualValue.compareTo(attrValue) !0)
                valid = false;
        }
        return valid;
    }

}
```

1.1.3 ParamController

```
package controller;

import com.sun.xml.tree.XmlDocument;
import error.LogException;
import error.ParamException;
import error.ParseException;
import org.w3c.dom.Element;
import param.Architecture;
import param.ProjectParameter;
import param.ProjectParameterImpl;
import xmlparser.QproParserImpl;

import java.io.File;

/**
 * ParamController validates the input to the system and creates the internal
 * objects needed to represent them in the ProjectParameter.
 *
 * @author Magnus Solbjørg
 * @version 0.5
 * @see param.ProjectParameter
 * @see xmlparser.QproParser
 * @since 25.10.2001
 * @see param.ProjectParameter
 * @see xmlparser.QproParser
 */
public class ParamController {

    /** Instantiates a new parser that will be used to parse the XMI formated model */
    public QproParserImpl theQproParserImpl = new QproParserImpl();
    private QproLog logger;

    /**
     * Constructor for class. Takes a QproLog object to be able to log what happens
     *
     * @param log Log object
     */
    public ParamController(QproLog log) {
        this.logger = log;

    }

    /**
     * Takes all the parmaters that is to be validated as parameters.  Returns the
     * internal representation of the parameters as objects.
     * @param projectName Name of the project
     * @param pathToXMIFile Path to the XML file containing model in Xmi format
     * @param srcDir Catalog to save generated code
     * @param chsSys Chosen system architecture
     * @param chsDB Chosen database architecture
     * @return param.ProjectParameter
     * @throws ParamException If validations failes, a ParamException is thrown
     */
    public ProjectParameter init(String projectName, String pathToXMIFile, String srcDir, Architecture chsSys,
Architecture chsDB) throws ParamException {
        XmlDocument xmlDoc = null;
        Element xmiRoot = null;  //For logging
        try {
            logger.log("Starting validation of params");
            //Validates project name
            if (projectName == "" || projectName == null) {
```

```
    logger.log("Prosjektnavn ikke gyldig");
    throw new ParamException("The name of the project must be specified.");
  }


  //Validates Architecture
  if (chsSys !=null && chsSys.getPathToRuleFile() !=null) {
      File chsSysFile = new File(chsSys.getPathToRuleFile());
    if (!chsSysFile.exists()) {
        logger.log("Architecture file does't exist: " + srcDir);
      throw new ParamException("Architecture file does't exist: " + srcDir);
    }else if (!chsSysFile.canRead()) {
        logger.log("Architecture file cannot be read: " + srcDir);
      throw new ParamException("Architecture file cannot be read: " + srcDir);
    }
  }else {
    logger.log("Systemarchitecture filename not specified.");
    if (chsDB == null || chsDB.getPathToRuleFile() == null)
      throw new ParamException("Architecture filename not specified.");
  }


  //Validates Database
  if (chsDB !=null && chsDB.getPathToRuleFile() !=null) {
      File chsDBFile = new File(chsDB.getPathToRuleFile());
    if (!chsDBFile.exists()) {
        logger.log("Database file does't exist: " + srcDir);
      throw new ParamException("Database file does't exist: " + srcDir);
    }else if (!chsDBFile.canRead()) {
      logger.log("Database file cannot be read: " + srcDir);
      throw new ParamException("Database file cannot be read: " + srcDir);
    }
  }else {
    logger.log("Database filename not specified.");
    if (chsSys == null || chsSys.getPathToRuleFile() == null)
      throw new ParamException("Database filename not specified.");
  }


  //Validates the XMI file
  if (pathToXMIFile !=null) {
      File XMIfile = new File(pathToXMIFile);
    if (!XMIfile.exists()) {
        logger.log("Cannot find the file: " + pathToXMIFile);
      throw new ParamException("Cannot find the file: " + pathToXMIFile);
    }else if (!XMIfile.canRead()) {
        logger.log("Cannot read the file: " + pathToXMIFile);
      throw new ParamException("Cannot read the file: " + pathToXMIFile);
    }else {
      // The file eksists and can be read. Lets parse it!
      try {
          xmlDoc = theQproParserImpl.parseXML(pathToXMIFile);
          xmiRoot = xmlDoc.getDocumentElement();
          logger.log("Parsed XMI tree:\n\n" + theQproParserImpl.writeTree(xmiRoot));
      }catch (ParseException pe) {
          logger.log("Cannot parse XMI file. Reason: " + pe.getMessage());
        throw new ParamException("Cannot parse XMI file. Reason: " + pe.getMessage());
      }
    }
  }else {
    logger.log("XMI filename not specified");
    throw new ParamException("XMI filename not specified");
  }
```

```
            ProjectParameterImpl pp = new ProjectParameterImpl();
            pp.init(projectName, chsSys, chsDB, srcDir, xmlDoc);

            logger.log("Validation of params is over.");
            return pp;

        }catch (LogException le) {
            throw new ParamException("Feil ved logging.");
        }
    }

}
```

1.1.4 QproController

```
package controller;

import GUI.MainWindow;
import GUI.QproGUI;
import com.sun.xml.tree.XmlDocument;
import error.BuildException;
import error.LogException;
import error.ParamException;
import error.ParseException;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import param.Architecture;
import param.ProjectParameter;
import xmlparser.QproParser;
import xmlparser.QproParserImpl;

import java.util.ArrayList;


/**The central controllobject of the system.
 * This is the central controllobject in the system, and stand for all the
 * communication between the controllerobjects of the system.
 * @see CodeController
 * @see ParamController
 * @author Qpro16 - konstruksjon
 * @version 0.1
 * @since 25.10.2001
 */
public class QproController {
    private final String START_STATUS_OK = "Generatoren har startet";
    private final String VALIDATE_STATUS_OK = "Validering/strukturering av parametre OK";
    private final String GENERATE_STATUS_OK = "Kode generert!";
    private final String ARCHPARSE_FAILED = "Feil i fil med arkitekturbeskrivelser";

    private final String LOG_EXTENSION = "txt";
    private final String ARCH_FILENAME = "config/archs.xml";

    private QproGUI theQproGUI;
    private ParamController theParamController;
    private CodeController theCodeController;
    private QproLog theQproLog;
    private ProjectParameter theProjectParameter;


    /**Standard constructor.
     *
     */
    public QproController() {
        theQproGUI = null;
        theParamController = null;
        theCodeController = null;
        theQproLog = null;
        theProjectParameter = null;
        initGUI();
    }

    private void initGUI() {
        theQproGUI = new MainWindow();
        ArrayList sysArchs = new ArrayList();
        ArrayList dbArchs = new ArrayList();
```

```
        try {
            getArchitectures(sysArchs, "sysarchitecture");
            getArchitectures(dbArchs, "dbarchitecture");
        }catch (ParamException e) {
            System.out.println(e.getMessage());
        }
        theQproGUI.init(sysArchs, dbArchs, this);
    }

    private void getArchitectures(ArrayList archs, String type) throws ParamException {
        QproParser parser = new QproParserImpl();
        XmlDocument doc = null;
        //Lager xml-tre utifra xml-filen, hvor arkitekturene er lagret
        try {
            doc = parser.parseXML(ARCH_FILENAME);
        }catch (ParseException e) {
            e.printStackTrace();
        }
        Element root = doc.getDocumentElement();
        NodeList list = root.getElementsByTagName(type);

        int length = list.getLength();
        //Går gjennom alle arkitektur-subtrær i xml-treet
        for (int i = 0; i < length; i++) {
            //Initialiserer variablene for en Architecture
            String idStr = null;
            int id = -1;
            String name = null;
            String rulePath = null;

            Node archNode = list.item(i);
            if (archNode.hasChildNodes()) {
                NodeList archChilds = archNode.getChildNodes();
                int subLength = archChilds.getLength();
                //Henter nedenfor ut all informasjon om en architecture ved å
                //gå gjennom alle barnenodene til en architecture-node
                for (int ii = 0; ii < subLength; ii++) {
                    Node infoNode = archChilds.item(ii);
                    Node textNode = infoNode.getFirstChild();
                    String nodeName = null;
                    String nodeValue = null;
                    //Navnet på noden ligger i barnenoden til architecture-noden, mens
                    //tekstverdien ligger i barnenoden til barnenoden.
                    if (textNode != null) {
                        nodeName = infoNode.getNodeName();
                        nodeValue = textNode.getNodeValue();//burde teste om det er textnode
                    }

                    if (nodeName == "name")
                        name = nodeValue;
                    else if (nodeName == "rulepath")
                        rulePath = nodeValue;
                    else if (nodeName == "id") {
                        idStr = nodeValue;
                        try {
                            id = Integer.parseInt(idStr);
                        } catch (NumberFormatException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }//if childNodes til architecture-node
            //Oppretter Architecture-objekt og legger det i riktig liste
```

```
            Architecture arch = new Architecture(id, name, rulePath);
            archs.add(arch);
        }
    }

    /**
     * Called by the GUI when all the needed information has been collected. Starts
     * ParamController and validates the parameters before the code generation is done
     * by the CodeControlller.
     * @param name The projectname
     * @param xmiFile The XMI that describes the model, from which the code is generated
     * @param directory The directory to put the generated code
     * @paramchsSys The chosen system architecture
     * @param chsDB The chosen database architecture
     */
    public void setValues(String name, String xmiFile, String directory, Architecture chsSys, Architecture chsDB) {
        //Oppretter log objektfor gjeldende generator prosjekt
        String logFileStr = name + "." + LOG_EXTENSION;
        try {
            theQproLog = new QproLog(logFileStr, directory);
            theQproGUI.updateState(START_STATUS_OK);
        } catch (LogException e) {
            e.printStackTrace();
            theQproGUI.updateState(e.getMessage());
        }

        //Setter igang ParamController sin validering og strukturering av parametrene
        ProjectParameter params = null;
        try {
            theParamController = new ParamController(theQproLog);
            params = theParamController.init(name, xmiFile, directory, chsSys, chsDB);
            theQproGUI.updateState(VALIDATE_STATUS_OK);
        } catch (ParamException e) {
            e.printStackTrace();
            theQproGUI.updateState(e.getMessage());
        }

        //Gir parametrene videre til CodeController, som starter å generere kode
        try {
            theCodeController = new CodeController(theQproLog);//TODO:fjern code parameter(debug)
            theCodeController.generate(params);
            theQproGUI.updateState(GENERATE_STATUS_OK);
        } catch (BuildException e) {
            e.printStackTrace();
            theQproGUI.updateState(e.getMessage());
        }
    }

}
```

1.1.5 QproLog

```
package controller;

import error.LogException;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Responsable for the loggwriting in the system
 * @author Martin Sleire Vatne
 * @version0.2
 * @since 26.10.2001
 */
public class QproLog {
   private FileWriter fw;

   /**
    * InitCreates the logFile and the directory (if it does not exists).
    * @param logFile containing the future name of the logFile
    * @param srcDir conaining the source directory wherer the logFile is be put
    * @throws LogException when an IOError or other unexpected error accoures.
    */
   public QproLog(String logFile, String srcDir) throws LogException {
      try {
         File theFile = new File(srcDir);
         if (!(theFile.exists())) {
            theFile.mkdir();
         }
         theFile = new File(theFile, logFile);
         theFile.createNewFile();
         fw = new FileWriter(theFile);
      } catch (IOException ioe) {
         throw new LogException("A IOException accourd during the initialization of the QproLog" +
ioe.getMessage());
      } catch (Exception e) {
         throw new LogException("Some unknown error accourd dureing the initialization of the QproLog" +
e.getMessage());
      }
   }

   /**
    * Writes the given content to the logfile
    * @param content contains whatever that is to be written to the log
    * @throws LogException if the Log has not been initialized, or IOERRORs accours.
    */
   public void log(String content) throws LogException {
      try {
         fw.write(content + "\n");
         fw.flush();
      } catch (IOException ioe) {
         throw new LogException("A IOException accourd during the initialization of the QproLog" +
ioe.getMessage());
      } catch (NullPointerException e) {
         throw new LogException("The QproLog has not been initialized" + e.getMessage());
      } catch (Exception e) {
         throw new LogException("Some unknown error accourd dureing the logging of" + content + " \n" +
e.getMessage());
      }
   }
}
```

1.1.6 BuildException

```
package error;


/**
 * An exception used by CodeController if any unknown error accourds during the building of the code.
 * @author Martin Sleire Vatne
 * @version 0.2
 * @since 26.10.2001
 */
public class BuildException extends QproException {

    /**
     * Default inheritaded contstructor
     */
    public BuildException() {
        super();
    }

    /**
     * Inheritaded constructor that takes an errrormsg and stores
     * this as its e.getMessage() value.
     * @param msg containing the error message of this Exception
     * @see QproException
     */
    public BuildException(String msg) {
        super(msg);
    }
}
```

1.1.7 LogException

```
package error;


/**
 * An exception used by QproLog if any exception accours during the logging in the system
 * @author Martin Sleire Vatne
 * @version 0.2
 * @since 26.10.2001
 */
public class LogException extends QproException {

    /**
     * Default inheritaded contstructor
     */
    public LogException() {
        super();
    }

    /**
     * Inheritaded constructor that takes an errrormsg and stores
     * this as its e.getMessage() value.
     * @param msg containing the error message of this Exception
     * @see QproException
     */
    public LogException(String msg) {
        super(msg);
    }
}
```

1.1.8 ParamException

```
package error;


/**
 * An exception used by the Param package and the ParamController if any exception takes place during
 * the validation or the building of the parameters.
 * @author Martin Sleire Vatne
 * @version 0.2
 * @since 26.10.2001
 */
public class ParamException extends QproException {

    /**
     * Default inheritaded contstructor
     */
    public ParamException() {
        super();
    }

    /**
     * Inheritaded constructor that takes an errrormsg and stores
     * this as itse.getMessage() value.
     * @param msg containing the error message of this Exception
     * @see QproException
     */
    public ParamException(String msg) {
        super(msg);
    }
}
```

1.1.9 ParseException

```
package error;


/**
 * An excepetion used by the xmlparser package if any error accours during the parsing of the file.
 * @author Martin Sleire Vatne
 * @version 0.2
 * @since 26.10.2001
 */
public class ParseException extends QproException {

    /**
     * Default inheritaded constructor
     */
    public ParseException() {
        super();
    }

    /**
     * Inheritaded constructor that takes an errrormsg and stores
     * this as its e.getMessage() value.
     * @param msg containing the error message of this Exception
     * @see QproException
     */
    public ParseException(String msg) {
        super(msg);
    }
}
```

1.1.10 QproException

```
package error;


/**
 * An general exception for the system, made to ease the errorhandling in latter extension of the system
 * @see java.lang.Exception
 * @author Martin Sleire Vatne
 * @version 0.2
 * @since 26.10.2001
 *
 */
public class QproException extends Exception {

    /**
     * Default inheritaded contstructor
     */
    public QproException() {
        super();
    }

    /**
     * Inheritaded constructor that takes an errrormsg and stores
     * this as its e.getMessage() value.
     * @param msg containing the error message of this Exception
     */
    public QproException(String msg) {
        super(msg);
    }
}
```

1.1.11 MainWindow

```
package GUI;


import controller.QproController;
import param.Architecture;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

/**
 * An implementation of the GUI interface of the Qprosystem
 *
 * @author Magnus Solbjørg
 * @version 0.5
 * @since 25.10.2001
 * @see GUI.QproGUI
 * @see GUI.MainWindowListener
 */
public class MainWindow implements QproGUI {


    private JFrame fGUI = new JFrame("Input - Qpro");
    //hovedpanelet

    JPanel nameRow = new JPanel();
    JPanel xmiRow = new JPanel();
    JPanel archRow = new JPanel();
    JPanel dbRow = new JPanel();
    JPanel outputRow = new JPanel();

    //private JLabel dummy = new JLabel("     g   ");

    private JLabel architecture = new JLabel("Code architecture: ");
    private JLabel database = new JLabel("Database architecture:");
    private JLabel name = new JLabel("Projectname: ");
    private JLabel XMIfile = new JLabel("XMI-file: ");
    private JLabel catalog = new JLabel("Output catalog: ");
    private JTextField tArchitecture = new JTextField(15);
    private JTextField tDatabase = new JTextField(15);
    private JTextField tName = new JTextField(15);
    private JTextField tXMIfile = new JTextField(15);
    private JButton XMIdialog = new JButton("Open file");
    private JTextField tCatalog = new JTextField(15);
    private JButton catalogDialog = new JButton("Select output catalog");
    private JPanel backPanel = new JPanel();
    private JButton back = new JButton("Back");
    private JButton exit = new JButton("Exit");

    private JPanel knapper = new JPanel();
    private JButton generate = new JButton("Generate");
    private JButton close = new JButton("Close");

    private JPanel state = new JPanel();
    private JTextArea stateMessage = new JTextArea(10, 40);

    private JFileChooser XMIchooser = new JFileChooser("./");
    private JFileChooser outputChooser = new JFileChooser();

    private int cat = outputChooser.DIRECTORIES_ONLY;
```

```
private JComboBox archList;

private JScrollPane archScrollPane = new JScrollPane(archList);

private JComboBox dbList;
private JProgressBar progress = new JProgressBar(0, 100);


private MainWindowListener listener;
private QproController parent;
private String XMIpath;
private ArrayList sysArch;
private ArrayList dbArch;

/**
 * Empty constructor for class.
 *
 * @since 25.10.2001
 */
public MainWindow(){


}

/**
 * This method generates architecture lists, adds actionlistener and starts the GUI.
 *
 * @param sysArcs Chosen system architecture
 * @param DBArcs Chosen database architecture
 * @param controller Takes the QproController that started the GUI.
 */
public void init(ArrayList sysArcs, ArrayList DBArcs, QproController controller) {
   this.sysArch = sysArcs;
   this.dbArch = DBArcs;
   this.parent = controller;
   //parent =controller;
   ArrayList sysArcsCombo = new ArrayList();
    Architecture archTemp;

   if (sysArcs.size() == 0) {
       sysArcsCombo.add("--- None available ---");
    }else {
       sysArcsCombo.add("--- None ---");
    }
   for (int i = 0; i < sysArcs.size(); i++) {
       archTemp = (Architecture) sysArcs.get(i);
       sysArcsCombo.add(archTemp.getArchitectureName());
    }

    ArrayList sysDBCombo = new ArrayList();
    Architecture DBTemp;
   if (DBArcs.size() == 0) {
       sysDBCombo.add("--- None available ---");
    }else {
       sysDBCombo.add("--- None ---");
    }
   for (int i = 0; i < DBArcs.size(); i++) {
       DBTemp = (Architecture) DBArcs.get(i);
       sysDBCombo.add(DBTemp.getArchitectureName());
    }

    archList = new JComboBox(sysArcsCombo.toArray());
    dbList = new JComboBox(sysDBCombo.toArray());
```

```
        listener = new MainWindowListener(this);
        start(true);

    }


    /**
     * Method that shows text in the GUI while generating
     *
     * @param text The text to be shown
     */
    public void updateState(String text) {
        System.out.println("GUIMELDING:" + text);
        state.setVisible(true);
        stateMessage.setText(stateMessage.getText() "\n" + text);

        progress.setValue(100);
        state.repaint();
        stateMessage.repaint();
        progress.repaint();
        fGUI.setVisible(true);
        fGUI.repaint();
    }

    /**
     *  Method to start the generation of code.
     */
    public void generate() {
       System.out.println("STARTER GENERATOREN");
       fGUI.getContentPane().removeAll();
        fGUI.getContentPane().setLayout(new GridLayout(2, 1));

        backPanel.setLayout(new BoxLayout(backPanel, BoxLayout.X_AXIS));
       back.addActionListener(listener);
        exit.addActionListener(listener);
       backPanel.add(back);
        backPanel.add(exit);

        backPanel.setVisible(true);
        fGUI.getContentPane().add(state);
        fGUI.getContentPane().add(backPanel);

       //fGUI.pack();
       fGUI.setVisible(true);
        fGUI.repaint();
        state.setVisible(true);
        stateMessage.setVisible(true);
        progress.setVisible(true);
        progress.setValue(5);
       //updateState("Starter...");


       if ((sysArch.size() >1 || dbArch.size() >1) && tName.getText() != null && tXMIfile.getText() != null
               && tCatalog.getText() != null) {
           Architecture archIndex = null;
           Architecture dbindex = null;
           if (archList.getSelectedIndex() > 0 && archList.getSelectedIndex() < archList.getItemCount())
               archIndex = (Architecture) sysArch.get((archList.getSelectedIndex() - 1));
           if (dbList.getSelectedIndex() > 0 && dbList.getSelectedIndex() < dbList.getItemCount())
               dbindex = (Architecture) dbArch.get((dbList.getSelectedIndex()) - 1);
           parent.setValues(tName.getText(), tXMIfile.getText(), tCatalog.getText(), archIndex, dbindex);
```

```
        } else {
            // returnerer barenull på arkitekturer hvis det ikke finnes noen
            parent.setValues("", "", "", null, null);
        }

    }

    /**
     * Methos that opens the filedialog needed to choose XMIfile
     */
    public void back() {
        fGUI.getContentPane().removeAll();
        start(false);
    }


    public void openXMIfileDialog() {
        int returnVal = XMIchooser.showOpenDialog(tXMIfile);

        tXMIfile.setText(XMIchooser.getCurrentDirectory().getAbsolutePath()+"\\"+
XMIchooser.getSelectedFile().getName());


    }

    /**
     * Methos that opensthe filechooser needed to choose output catalog
     */

    public void openCatalogDialog() {
        int returnVal = outputChooser.showOpenDialog(tCatalog);

        tCatalog.setText(outputChooser.getCurrentDirectory().getAbsolutePath());

    }

    /**
     * Method to exit the system. Is used if the user chooses cansel or closes the GUI.
     */
    public void exit() {
        System.exit(0);
    }


    /**
     * Builds the GUI.
     */
    private void start(boolean first) {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        }catch (Exception e) {
            System.exit(0);
        }


        if (first) {
            generate.addActionListener(listener);
            close.addActionListener(listener);
            fGUI.addWindowListener(listener);
            catalogDialog.addActionListener(listener);
            XMIdialog.addActionListener(listener);

        }
```

```
        state.setLayout(new BoxLayout(state, BoxLayout.Y_AXIS));
        knapper.setLayout(new BoxLayout(knapper, BoxLayout.X_AXIS));

        state.add(stateMessage);
        state.add(progress);

        state.setVisible(true);
       //knapper.add(Box.createRigidArea(new Dimension(5,5)));
       knapper.add(Box.createHorizontalGlue());
        close.setAlignmentX(knapper.LEFT_ALIGNMENT);
        generate.setAlignmentX(knapper.RIGHT_ALIGNMENT);
       knapper.add(close);
        knapper.add(generate);
       //knapper.add(Box.createRigidArea(new Dimension(5, 5)));

       fGUI.getContentPane().setLayout(new GridLayout(6, 3));

       fGUI.getContentPane().add(name);
        fGUI.getContentPane().add(tName);
        fGUI.getContentPane().add(new Canvas());


        fGUI.getContentPane().add(XMIfile);
        fGUI.getContentPane().add(tXMIfile);
       fGUI.getContentPane().add(XMIdialog);

       fGUI.getContentPane().add(architecture);
        fGUI.getContentPane().add(archList);
        fGUI.getContentPane().add(new Canvas());

        fGUI.getContentPane().add(database);
        fGUI.getContentPane().add(dbList);
        fGUI.getContentPane().add(new Canvas());


        fGUI.getContentPane().add(catalog);
        fGUI.getContentPane().add(tCatalog);
        fGUI.getContentPane().add(catalogDialog);

        fGUI.getContentPane().add(new Canvas());
       fGUI.getContentPane().add(knapper);
       fGUI.getContentPane().add(new Canvas());


        fGUI.pack();
        fGUI.setVisible(true);
        fGUI.repaint();
    }


}
```

1.1.12 MainWindowListener

```
package GUI;


import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

/**
 * Listener for the GUI
 *
 * @author Magnus Solbjørg
 * @version 0.5
 * @since 25.10.2001
 * @see GUI.MainWindow
 */
public class MainWindowListener implements ActionListener, WindowListener {
    /**
     * The GUI
     */
    private MainWindow parent;
    /**
     * The action performed
     */
    public String action;


    /**
     * Constructor for class. Takes the GUI object as input
     *
     * @param qCon GUI object
     */
    public MainWindowListener(MainWindow qCon) {
        parent = qCon;

    }

    /**
     * This metod takes right action based on what the user does in the GUI.
     *
     * @param e Action that is performed.
     */
    public void actionPerformed(ActionEvent e) {


        action = e.getActionCommand();

       if (action == "Generate") {
            System.out.println("Generate");
            parent.generate();

        } else if (action == "Close") {
            System.out.println("Closing...");
            parent.exit();
        } else if (action == "Open file") {
            parent.openXMIfileDialog();
        } else if (action == "Select output catalog") {
            parent.openCatalogDialog();
        } else if (action == "Back") {
            parent.back();
        } else if (action == "Exit") {
```

```
            parent.exit();
        }

    }

    /**
     * windowActivated.
     */
    public void windowActivated(WindowEvent e) {
    }

    /**
     * windowClosed.
     */
    public void windowClosed(WindowEvent e) {
    }

    /**
     * windowClosing.
     */
    public void windowClosing(WindowEvent e) {
        parent.exit();
    }

    /**
     * windowDeactivated method comment.
     */
    public void windowDeactivated(WindowEvent e) {
    }

    /**
     * windowDeiconified.
     */
    public void windowDeiconified(WindowEvent e) {
    }

    /**
     * windowIconified.
     */
    public void windowIconified(WindowEvent e) {
    }

    /**
     * windowOpened.
     */
    public void windowOpened(WindowEvent e) {

    }


}
```

1.1.13 QproGUI

```
package GUI;

import controller.QproController;

import java.util.ArrayList;

/**
 * Defines the grapical user interface of the system.
 * @author Magnus Solbjørg
 * @version 0.1
 * @since 25.10.2001
 * @see GUI.MainWindow
 */
public interface QproGUI {

    /**
     * This method generates architecture lists, adds actionlistener and starts the GUI.
     *
     * @param sysArcs Chosen system architecture
     * @param DBArcs Chosen database architecture
     * @param controller Takes the QproController that started the GUI.
     */
    public void init(ArrayList sysArcs, ArrayList DBArcs, QproController controller);

    /**
     * Method that shows text in the GUI while generating
     *
     * @param text The text to be shown
     */
    public void updateState(String text);

    /**
     * Method to exit the system. Is used if the user chooses cansel or closes the GUI.
     */
    public void exit();
}
```

1.1.14 Architecture

```
package param;


/**
 * The objectrepresentation of an architecture with id, navn, og path to the
 * corresponding rulefile.
 *
 * @author Ole Kristian Hoel
 * @version 1.0
 * @since 04.11.2001
 */
public class Architecture {
    private int architectureID;
    private String architectureName;
    private String pathToRuleFile;

    /**
     * Constructor for the Architectureclass.
     *
     * @param id Identificationnumber
     * @param name Name of the architecture
     * @paramrulePath Path to the rulefile
     */
    public Architecture(int id, String name, String rulePath) {
        this.architectureID = id;
        this.architectureName = name;
        this.pathToRuleFile = rulePath;

    }

    /**
     * Returnsthe path to the rulefile
     *
     * @return Path to the rulefile
     */
    public String getPathToRuleFile() {
        return pathToRuleFile;
    }

    /**
     * Returns the name of the architevture to be used
     *
     * @return Name of the architecture
     */
    public String getArchitectureName() {
        return architectureName;
    }

    /**
     * Returns the id of the architecture
     *
     * @return The id of the architecture
     */
    public int getArchitectureID() {
        return architectureID;
    }
}
```

1.1.15 ProjectPaameter

```
package param;

import com.sun.xml.tree.XmlDocument;
import error.ParamException;
import org.w3c.dom.Element;

import java.util.ArrayList;

/**
 * A interface for the internal representation of all the parameters in the system.
 * This interfaces defines all the methods that the architecture rules and
 * templates must contain.
 *
 * @author Magnus Solbjørg
 * @version 0.5
 * @since 25.10.2001
 * @see param.ProjectParameterImpl
 */
public interface ProjectParameter {

    /**
     * Method that saves the params in intern variables, and gets the paths to the architecture files needed
     * for the generation
     *
     * @param proName Name of the project
     * @param chsSys Chosen system architecture
     * @param chsDB Chosen database architecture
     * @param srcDir Catalog where generated code will be saved
     * @param xmiDoc The parsed XML document, containing the model
     */
    public void init(String proName, Architecture chsSys, Architecture chsDB, String srcDir, XmlDocument
xmiRoot) throws ParamException;

    /**
     * Method that returns the parsed XML as a XmlDocument object
     *
     * @return Parsed XML docuement
     */
    public XmlDocument getXMIDocument();

    /**
     * Returns the name of the project.
     *
     * @return The project name.
     */
    public String getProjectName();

    /**
     * Returns output directory for the generated code.
     *
     * @return The output directory.
     */
    public String getSrcDir();

    /**
     * Returns the chosen architecture. You can get database and systemarchitecture.
     *
     * @param database Boolean value used to choose either system architecture or database architecture
     * @return The architecture
     */
    public Architecture getArchitecture(boolean database);
```

```
    /**
     * Returns true or false if a element is trigger or not.
     *
     * @param elementName An element in the model
     * @param database Boolean value used to choose either system architecture or database architecture
     * @return If the element is trigger or not
     */
    public boolean isTrigger(String elementName, boolean database);

    /**
     * Method that returns an ArrayList containing all templates to be used on the an element.
     *
     * @param elementName An element in the model
     * @param database Boolean value used to choose either system architecture or database architecture
     * @return An ArrayList containing the templates to be used on the element
     */
    public ArrayList getTemplatesByTrigger(String elementName, boolean database);

    /**
     * Returns an Elementobject containing a rule based on what tag you use in.
     *
     * @param tag Tag describing the name of a rule
     * @param database Boolean value used to choose either system architecture or database architecture
     * @return An element containing a rule
     */
    public Element getRuleElement(String tag, boolean database);

    /**
     * Returns an internal template from the ruleobject.
     *
     * @param name The name of an template
     * @param database Boolean value used to choose either system architecture or database architecture
     * @return The template
     */
    public Template getInternalTemplateByName(String name, boolean database);

    /**
     * Gets the name of the idattribute that vil be used to serach in the tree by id.
     *
     * @return String containing the name of the attribute containing an id in XML
     */
    public String getIdAttributeName(boolean database);
}
```

1.1.16 ProjectParamaterImpl

```
package param;

import com.sun.xml.tree.XmlDocument;
import error.ParamException;
import org.w3c.dom.Element;

import java.util.ArrayList;

/**
 * A agregation class that implements the ProjectParameter inteface. Solves the
 * methods by instances of the classes Rules and Architecture.
 *
 * @author Magnus Solbjørg
 * @version 0.5
 * @since 25.102001
 * @see param.ProjectParameter
 * @see param.Rules
 */
public class ProjectParameterImpl implements ProjectParameter {
    private String srcDir;
    private String projectName;
    private XmlDocument xmiDoc;

    /**
     * Chosen system rules
     */
    private Rules chosenSysRules;
    /**
     * Chosen database rules
     */
    private Rules chosenDBRules;
    /**
     * Chosen system architecture
     */
    private Architecture chosenSystem;
    /**
     * Chosen system architecture
     */
    private Architecture chosenDB;


    /**
     * Emty constructor
     */
    public ProjectParameterImpl() {


    }

    /**
     * Method that saves the params in intern variables, and gets the paths to the architecture files needed
     * for the generation
     *
     * @param proName Name of the project
     * @param chsSys Chosen system architecture
     * @param chsDB Chosen database architecture
     * @param srcDir Catalog where generated code will be saved
     * @param xmiDoc The parsed XML document, containing the model
     */
    public void init(String proName, Architecture chsSys, Architecture chsDB, String srcDir, XmlDocument xmlDoc)
throws ParamException {
```

```
        this.srcDir = srcDir;
        this.projectName = proName;
        this.xmiDoc = xmlDoc;
        this.chosenSystem = chsSys;
        this.chosenDB = chsDB;

        if (chosenSystem != null)
            chosenSysRules = new Rules(chosenSystem.getPathToRuleFile());
        if (chosenDB != null)
            chosenDBRules = new Rules(chosenDB.getPathToRuleFile());


    }

    /**
     * Method that returns the parsed XML as a XmlDocument object
     *
     * @return Parsed XML docuement
     */
    public XmlDocument getXMIDocument() {
        return xmiDoc;
    }

    /**
     * Returns the name of the project.
     *
     * @return The project name.
     */
    public String getProjectName() {
        return projectName;
    }

    /**
     * Returns output directory for the generated code.
     *
     * @return The output directory.
     */
    public String getSrcDir() {
        return srcDir;
    }

    /**
     * Returns the chosen architecture. You can get database and systemarchitecture.
     *
     * @param database Boolean value used to choose either system architecture or database architecture
     * @return The architecture
     */
    public Architecture getArchitecture(boolean database) {
        if (database) {
            return chosenDB;
        } else {
            return chosenSystem;
        }
    }

    /**
     * Returns true or false if a element is trigger or not.
     *
     * @param elementName An element in the model
     * @param database Boolean value used to choose either system architecture or database architecture
     * @return If the element is trigger or not
     */
```

```
public boolean isTrigger(String elementName, boolean database) {

    if (database) {
        //Database architecture
        return chosenDBRules.isTrigger(elementName);
    }else {
        //System architecture
        return chosenSysRules.isTrigger(elementName);
    }

}

/**
 * Method that returns an ArrayList containing all templates to be used on the an element.
 *
 * @param elementName An element in the model
 * @param database Boolean value used to choose either system architecture or database architecture
 * @return An ArrayList containing the templates to be used on the element
 */
public ArrayList getTemplatesByTrigger(String elementName, boolean database) {
    if (database) {
        //Database architecture
        return chosenDBRules.getTemplatesByTrigger(elementName);
    }else {
        //System architecture
        return chosenSysRules.getTemplatesByTrigger(elementName);
    }

}

/**
 * Returns an Elementobject containing a rule based on what tag you send in.
 *
 * @param tag Tag describing the name of a rule
 * @param database Boolean value used to choose either system architecture or database architecture
 * @return An element containing a rule
 */
public Element getRuleElement(String tag, boolean database) {
    if (database) {
        //Database architecture
        return chosenDBRules.getRuleElement(tag);
    }else {
        //System architecture
        return chosenSysRules.getRuleElement(tag);
    }

}

/**
 * Returns an internal template from the ruleobject.
 *
 * @param name The name of an template
 * @param database Boolean value used to choose either system architecture or database architecture
 * @return The template
 */
public Template getInternalTemplateByName(String name, boolean database) {
    if (database) {
        //Database architecture
        return chosenDBRules.getInternalTemplateByName(name);
    }else {
        //System architecture
        return chosenSysRules.getInternalTemplateByName(name);
    }
```

```
    }

    /**
     * Gets the name of the idattribute that vil be used to serach in the tree by id.
     *
     * @return String containing the name of the attribute containing an id in XML
     */
    public String getIdAttributeName(boolean database) {
        if (database) {
            //Database architecture
            return chosenDBRules.getIdAttributeName();
        }else{
            //System architecture
            return chosenSysRules.getIdAttributeName();
        }

    }
}
```

1.1.17 Rules

```
package param;

import com.sun.xml.tree.XmlDocument;
import error.ParamException;
import error.ParseException;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import xmlparser.QproParser;
import xmlparser.QproParserImpl;

import java.util.ArrayList;
import java.util.Hashtable;

/**
 * An objectrepresentation of the Rulefile.
 *
 * @author Ole KristianHoel
 * @version 1.1
 * @since 07.11.2001
 */
public class Rules {
    private Hashtable triggers;
    private Element rules;
    private Hashtable templates;
    private String idAttributeName;

    /**
     * Holds templates
     */
    public Template theTemplate[];

    /**
     * Constructor for the Rulesobject.
     * The constructor is building the rulesobject based on the rules
     * in the given rulefile.
     *
     * @param pathToRuleFile A string containing the path tothe rule-file
     */
    public Rules(String pathToRuleFile) throws ParamException {
        XmlDocument xml;
        triggers = new Hashtable();
        templates = new Hashtable();
        QproParser parser = new QproParserImpl();

        //Parsing the rulesfile...
        try {
            xml = parser.parseXML(pathToRuleFile);
        }catch (ParseException pe) {
            throw new ParamException(pe.getMessage());
        }
        //...and get the root-element
        Element xmlRoot = xml.getDocumentElement();

        if (!xmlRoot.getTagName().equals("RULEFILE")) invalidRules("First tag in Rulesfile must be
<RULEFILE>.");


        /*
         * Get the character that separates the variables from the rest of the rules,
         * and check if it is valid.
```

```
        * (This is a simple test that won't catch all errors)
        */
       NodeList nl = xmlRoot.getElementsByTagName("VARIABLE_CHARACTER");
       if (nl.getLength() != 1) invalidRules("The Rules-file must contain exactly one <VARIABLE_CHARACTER>
tag.");
        Element varcharElm = (Element) nl.item(0);
        String varchar = varcharElm.getAttribute("char");

       if (varchar.length() < 1) invalidRules("The char-attribute in the <VARIABLE_CHARACTER>-tag must be
set!");
       if (varchar.length() > 1) invalidRules("The char-attribute in the <VARIABLE_CHARACTER>-tag must be only
one character");

       /*
        * Getting the name of the attribute in the xml that contains the
        * mapping between the internal type representation and the regular names.
        */
       nl = xmlRoot.getElementsByTagName("ID_ATTRIBUTE");
       if (nl.getLength() != 1)
           invalidRules("The Rules-file must contain exactly one <ID_ATTRIBUTE>-tag.");
       else {
           Element idAttrElm = (Element) nl.item(0);
          this.idAttributeName = idAttrElm.getAttribute("name");
        }

       /*
        * Get the templates and put them in a arraylist.
        * All templates in one arraylist must have the same trigger,
        * and there should not be more than one arraylist for each trigger.
        * Then the arraylists are put into a hash table with the trigger as key.
        */
       nl = xmlRoot.getElementsByTagName("TEMPLATES");
       if (nl.getLength() != 1) invalidRules("The Rules-file must contain excactly one <TEMPLATES>-tag.");
        Element templateTag = (Element) nl.item(0);
        NodeList templateNodes = templateTag.getChildNodes();

       //Go through all the templates
       for (int index = 0; index < templateNodes.getLength(); index++) {
          if (templateNodes.item(index).getNodeType() == Node.ELEMENT_NODE) {
              Element templateNode = (Element) templateNodes.item(index);
              String templateTrigger = templateNode.getAttribute("trigger");
              String templateSource = "config/templates/" + templateNode.getAttribute("source");
              String templateTarget = templateNode.getAttribute("target");

             if (templateTrigger.equals("")) invalidRules("Template "+ templateNode.getNodeName() + " has no
value for attribute 'trigger'");
             if (templateSource.equals("config/templates/")) invalidRules("Template "+
templateNode.getNodeName() +" has no value for attribute 'source'");
             if (templateTarget.equals("")) invalidRules("Template "+ templateNode.getNodeName() + " has no
value for attribute 'target'");

              Template template = new Template(templateSource, templateTarget, varchar);

              ArrayList someTemplates;
             //If there exist no arraylist with this trigger...
             if (!isTrigger(templateTrigger)) {
                //...create one and put the arraylist in the hash table
                someTemplates = new ArrayList();
                 triggers.put(templateTrigger, someTemplates);
             }else {
                //Else get the arraylist with this trigger
                someTemplates = getTemplatesByTrigger(templateTrigger);
             }
```

```
            //And then put the template in the template in the arraylist
            someTemplates.add(template);
          }
        }

        /*
         * Put the internal templates in a hash table with the name
         * of the template as key.
         */
        nl = xmlRoot.getElementsByTagName("INTERNAL_TEMPLATES");
        if (nl.getLength() != 1) invalidRules("The Rules file must contain excactly one <INTERNAL_TEMPLATES>
tag.");
        Element internalTemplateTag = (Element) nl.item(0);
        NodeList internalTemplateNodes = internalTemplateTag.getChildNodes();

        //Go through all the templates...
        for (int index = 0; index < internalTemplateNodes.getLength(); index++) {
          if (internalTemplateNodes.item(index).getNodeType() == Node.ELEMENT_NODE) {
            Element internalTemplateNode = (Element) internalTemplateNodes.item(index);
            String templateName = internalTemplateNode.getNodeName();


            NodeList templTextNodes = internalTemplateNode.getChildNodes();
            StringBuffer templText = new StringBuffer("");
            for (int i = 0; i < templTextNodes.getLength(); i++) {
              Node temp = templTextNodes.item(i);
              if (temp.getNodeType() == Node.TEXT_NODE) {
                templText.append(temp.getNodeValue());
              } else if (temp.getNodeType() == Node.ENTITY_REFERENCE_NODE) {
                Node partOfTemplate = temp.getFirstChild();
                if (partOfTemplate.getNodeType() == Node.TEXT_NODE) {
                  String text = partOfTemplate.getNodeValue();
                  templText.append(partOfTemplate.getNodeValue());
                } else {

                  throw new ParamException(
                      "An error occured during building of the internal template: " + templateName);
                }
              } else {

                throw new ParamException(
                    "An error occured during building of the internal template: " + templateName);
              }
            }

            String templateContent = templText.toString();
            /*
             *
             Node templateContentNode = internalTemplateNode.getFirstChild();
             String templateContent = templateContentNode.getNodeValue();
             */
            Template template = new Template(templateContent, varchar);

            //...and put it in the hash table
            templates.put(templateName, template);
          }
        }

        //The rules are stored in a tree
        nl = xmlRoot.getElementsByTagName("RULES");
        if (nl.getLength() != 1) invalidRules("The Rules file must contain excactly one <RULES> tag.");
        rules = (Element) nl.item(0);
      }
```

```
/**
 * Private method for throwing a ParamException including text
 * that says the Rules-file is invalid.
 *
 * @param message String containing the message to be printed
 * @throws ParamException
 */
private void invalidRules(String message) throws ParamException {
    throw new ParamException("Invalid Rules-file!\n" + message);
}

/**
 * Method for checking whether a string is a trigger in the rules.
 *
 * @param elementName String to be checked
 * @return Boolean value telling whether elementName is a trigger
 */
public boolean isTrigger(String elementName) {
    return triggers.containsKey(elementName);
}

/**
 * Returns an ArrayList containing the templates with the trigger elementName.
 * <code>null</code> will be returned if no trigger with the given name exists .
 *
 * @param elementName The trigger
 * @return ArrayList containing all the templates with the given trigger
 * @see Template
 */
public ArrayList getTemplatesByTrigger(String elementName) {
    return (ArrayList) triggers.get(elementName);
}

/**
 * Returns the root-element of the rule with the given tree.
 *
 * @param tag Name of the tag
 * @return Root-element of the rule with the given tag
 * @see Element
 */
public Element getRuleElement(String tag) {
    NodeList nl = rules.getElementsByTagName(tag);
    Element ruleElement = (Element) nl.item(0);
    return ruleElement;
}

/**
 * Returns the internal template with the given name.
 *
 * @param name Name of the internal template to be returned
 * @return Internal template with the given name
 */
public Template getInternalTemplateByName(String name) {
    return (Template) templates.get(name);
}

/**
 * Returns a string containing the name of the attribute containing an id in XML.
 *
 * @return String containing the name of the attribute containing an id in XML
 */
public String getIdAttributeName() {
```

```
        return this.idAttributeName;
    }
}
```

1.1.18 Template

```
package param;

import error.ParamException;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;

/**
 * The objectrepresentation of a templatefile or an internal template in the
 * rules. Got the method getNextToken that returns the next segment of the
 * template in question.
 *
 * @author Ole Kristian Hoel
 * @version 1.0
 * @since 05.11.2001
 */
public class Template {
    private StringTokenizer st;
    private String target;
    private String delimiter;
    private boolean isInternal;
    private String internalTemplate;
    private StringBuffer fileContentBuffer;

    private boolean staticState = true;

    /**
     * Constructor that builds the template object for an internal template.
     *
     * @param internalTemplate String containing the internal template
     * @param delimiter One character string that is used to mark a variable in the template
     */
    public Template(String internalTemplate, String delimiter) {
        this.delimiter = delimiter;
        this.internalTemplate = internalTemplate;
        this.st = new StringTokenizer(internalTemplate, delimiter, true);
        this.target = "";
        this.isInternal = true;

    }

    /**
     * Constructor that builds the template object for a template in a file.
     *
     * @param pathToFile Path to the template file
     * @param target Filename where to save the finished file
     * @param delimiter One character string that is used to mark a variable in the template
     * @throws ParamException
     */
    public Template(String pathToFile, String target, String delimiter) throws ParamException {
        this.delimiter = delimiter;
        this.target = target;
        this.isInternal = false;
        try {
            BufferedReader file = new BufferedReader(new FileReader(pathToFile));
            String line = file.readLine();
            fileContentBuffer = new StringBuffer();
```

```
        //The file is read line by line
        while (line != null) {
            fileContentBuffer.append(line + "\n");
            line = file.readLine();
        }
        String fileContent = fileContentBuffer.toString();
        this.st = new StringTokenizer(fileContent, delimiter, true);

    } catch (FileNotFoundException fnfe) {
        //Skrive om catch-setningene
        throw new ParamException("File " + pathToFile + " could not be found");
    } catch (IOException ioe) {
        throw new ParamException(ioe.getMessage());
    }


}

/**
 * Method that returns the next token in the template.
 * The token is either static text or a variable/parameter that
 * is to be replaced.
 * The character that marks the variables will not be returned.
 *
 * @return The next token in the template
 * @see TemplateToken
 */
public TemplateToken getNextToken() {
    String temp = st.nextToken();
    while (temp.equals(delimiter)) {
        //The delimiter means we should change state
        staticState = !staticState;
        //Get the next token, so the delimiter is not returned
        //If the delimiter was the last character,
        //an empty string will be returned
        if (st.hasMoreTokens()) {
            temp = st.nextToken();
        } else {
            temp = "";
        }
    }

    return new TemplateToken(staticState, temp);

}

/**
 * Method that returns the target file (filename) for the template.
 *
 * Replaces the variable in the filename with the given value.
 * This method can only replace one variable in a filename.
 * If there are more than one variable to be replaced the others
 * will remain in the filename, and so will the character that marks the variable.
 *
 * This means that more than one variable in a target probably will cause the
 * program to fail.
 *
 * @param variable Variable to be replaced
 * @paramnewValue The value to replace the variable with
 * @return The filename for the template
 */
public String getTarget(String variable, String newValue) {
    String filename = "";
```

```
        Template t = new Template(target, delimiter);

        while (t.hasMoreTokens()) {
            TemplateToken tt = t.getNextToken();
            //If the token is static, the text will be added
            if (tt.isStatic()) {
                filename += tt.getValue();
            } else {
                //If the variable should be replaced, it will be
                if (tt.getValue().equals(variable)) {
                    filename += newValue;
                //Or else it is just added
                } else {
                    filename += delimiter + tt.getValue() + delimiter;
                }
            }
        }
        return filename;
    }

    /**
     * Returns whether the template has more tokens.
     *
     * It will return true after the last token if that
     * token was not static, but invoking the getNextToken() method
     * is safe since it in that case will return an empty string.
     *
     * @return Boolean value that tells whether the template has more tokens
     * @see TemplateToken
     */
    public boolean hasMoreTokens() {
        return st.hasMoreTokens();
    }

    /**
     * Returns whether the template is internal.
     *
     * @return Boolean value that tells whether the template is internal
     */
    public boolean isInternal() {
        return isInternal;
    }

    /**
     * Resets an internal template. After running this method the getNextToken()
     * method will return the first token in the template.
     *
     * Running this method on a template that is not internal will cause
     * a ParamException to be thrown.
     *
     */
    public void reset() {
        if (this.isInternal()) {
            this.st = new StringTokenizer(internalTemplate, delimiter, true);
        } else {
            String fileContent = fileContentBuffer.toString();
            this.st = new StringTokenizer(fileContent, delimiter, true);
        }

    }
}
```

1.1.19 TemplateToken

```
package param;


/**
 * This class is an encapsulation of an token in the template. This token is
 * either a static text or a parameter that must be replaced using the related
 * rules.
 *
 * @author Qpro 16
 * @version 0.1
 * @see param.Template
 * @since 23.10.2001
 */
public class TemplateToken {
    private boolean isStatic;
    private String tokenValue;

    /**
     * Constructor for the TemplateToken
     *
     * @param isStatic
     * @param tokenValue
     */
    public TemplateToken(boolean isStatic, String tokenValue) {
        this.isStatic = isStatic;
        this.tokenValue = tokenValue;

    }

    /**
     * Tells whether the TemplateToken is static or wheter it should be replaced.
     *
     * @return Boolean that tells whether the TemplateToken i static
     */
    public boolean isStatic() {
        return isStatic;
    }

    /**
     * Returns a String containing the text in the TemplateToken.
     *
     * @return String with the text in the TemplateToken
     */
    public String getValue() {
        return tokenValue;
    }
}
```

1.1.20 QproParser

```
package xmlparser;

import com.sun.xml.tree.XmlDocument;
import error.ParseException;
import org.w3c.dom.Element;

/**
 * Defines all the methods of XML parsing and utils that is needed in the system.
 *
 * @author Martin Sleire Vatne
 * @version 0.2
 * @since 26.10.2001
 */
public interface QproParser {

    /**
     * Parses an XML file into an com.sun.xml.tree.XMLDocument object.
     * @param pathToXMLFile a String containing a valid path to an XMI file
     * @throws error.ParseException containing a errormsg what went wrong.
     * @return com.sun.xml.tree.XmlDocument
     */
    public XmlDocument parseXML(String pathToXMLFile) throws ParseException;

    /**
     * Returns a textual representation of the subtree in the element specified.
     *
     * @param root an Element containing the rootNode of the subtree
     * @return String the tree in textual representation
     */
    public String writeTree(Element root);

}
```

1.1.21 QproParserImpl

```
package xmlparser;

import com.sun.xml.parser.Resolver;
import com.sun.xml.tree.XmlDocument;
import error.ParseException;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

import java.io.File;
import java.io.IOException;

/**
 * This class is an realization of the QproParser interface.
 * @author Martin Sleire Vatne
 * @version 0.2
 * @since 26.10.2001
 */
public class QproParserImpl implements QproParser {

    private StringBuffer tree;

    /**Default contstructor*/

    public QproParserImpl() {
        tree = null;
    }

    /**
     * The implementation of the parseXML method in the QproParser interface
     * @see QproParser
     */
    public XmlDocument parseXML(String pathToXMLFile) throws ParseException {
        File ruleFile = new File(pathToXMLFile);
        try {
            InputSource src = Resolver.createInputSource(ruleFile);
            return XmlDocument.createXmlDocument(src, true); //not validatin'
        } catch (SAXException saxe) {
            throw new ParseException("Parsererror!\n Invalid XML input\n" + saxe.getMessage());
        }catch (IllegalStateException ise) {
            throw new ParseException("Parsererror!\n Illegal Parser State\n" + ise.getMessage());
        }catch (IOException ioe) {
            throw new ParseException("Parsererror!\n Invalid filepath or other IO error\n" + ioe.getMessage());
        }catch (Exception e) {
            throw new ParseException("Parsererror!\n A strange and mysterious error took place\n" +
e.getMessage());
        }

    }

    /**
     * The implementation of the writeTree method in the QproParser interfac
     * @see QproParser
     */

    public String writeTree(Element e) {
        makeTree((Node) e, 0);
        return tree.toString();
```

```
    }

    /**
     * A private utility method that recursive writes the tree, used by the writeTree method.
     * @param e an Element containing the root node of the current subtree
     * @param level an int containing the level of the rootelement in the total tree
     * @see writeTree
     */
    private boolean makeTree(Node e, int level) {
        if (e == null) return true;

        //Element n;
        NamedNodeMap nnm;
        NodeList nl, nlT;
         Element eT, eN;
        nl = e.getChildNodes();
        if (level == 0) {
             tree = new StringBuffer();
            printNode((Node) e);
         }

        for (int i = 0; i < nl.getLength(); i++) {
            for (int j = 0; j <= level; j++) {
                tree.append("\t");
            }
             Node n = nl.item(i);
            if (n.hasChildNodes()) {
                //har barn
                printNode(n);
                makeTree(n, level + 1); //sets the current element as root and makes the subtree (DFS)

            } else {
                 printNode(nl.item(i));
            }


        }


        return true;
    }

    /**A private utility method that prints all the relevant infomation of an node in the tree
     * Used by makeTree to make the textual representation of a tree.
     * @param n containing the node which is to be described in text.
     * @see makeTree
     */
    private void printNode(Node n) {
        switch (n.getNodeType()) {
            case Node.TEXT_NODE:
                tree.append("TEXT-NODE with value: [" + n.getNodeValue().trim() + "]\n");
                break;
            case Node.ATTRIBUTE_NODE:
                tree.append("ATTRIBUTE-NODE named [" + n.getNodeName() + "] with value [" +
n.getNodeValue().trim() + "]\n");
                break;
            case Node.CDATA_SECTION_NODE:
                tree.append("CDATA SECTION NODE with value [" + n.getNodeValue() + "]\n");
                break;
            case Node.COMMENT_NODE:
                tree.append("COMMENT-NODE with value [" + n.getNodeValue() + "]\n");
                break;
            case Node.DOCUMENT_FRAGMENT_NODE:
```

```
                tree.append("DOCUMENT FRAGMENT NODE\n");
                break;
            case Node.DOCUMENT_NODE:
                tree.append("DOCUMENT NODE\n");
                break;
            case Node.DOCUMENT_TYPE_NODE:
                tree.append("DOCUMENT TYPE NODE with name [" + n.getNodeName() + "]\n");
                break;
            case Node.ELEMENT_NODE:
                tree.append("ELEMENT NODE with name [" + n.getNodeName() + "] and [" +
n.getAttributes().getLength() + "] attributes\n");
                break;
            case Node.ENTITY_NODE:
                tree.append("ENTITY NODE with name [" + n.getNodeName() + "]\n");
                break;
            case Node.ENTITY_REFERENCE_NODE:
                tree.append("ENTITY REFERENCE NODE with name [" + n.getNodeName() + "]\n");
                break;
            case Node.NOTATION_NODE:
                tree.append("NOTATION NODE with name [" + n.getNodeName() + "]\n");
                break;
            case Node.PROCESSING_INSTRUCTION_NODE:
                tree.append("PROCESSING INTSTRUCTION NODE \n");
                break;
            default
                tree.append("#ERROR!!! UNKNOWN NODE !! \n");
                break;
        }
    }
}
```

Kundestyrt prosjekt

Høst 2001

Prosjektevaluering

Versjon 1.00

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
05.11.2001	Opprettelse, skissering av innhold	0.01	Martin
06.11.2001	Utfyllelse av innhold, klargjøring til gjennomlesing av resten av gruppen.	0.10	Martin
07.11.2001	Skisserte tidsforbruk kapittelet må vente på siste timelister før det kan fullføres.	0.15	Martin
07.11.2001	La til om prosjektfasene	0.20	Magnus
07.11.2001	Førsteutkast til fagevaluering	0.30	Martin
07.11.2001	Skrevet litt om bla. testing og tilgjengelige ressurser	0.31	Atle
08.11.2001	Momenter til prosjektresultat lagt til	0.32	Martin
09.11.2001	Tidsforbruket oppdatert	0.40	Martin
09.11.2001	Skrevet ferdig om test	0.41	Atle
11.11.2001	Fikset småting – bl.a. korrektur	0.42	Odd Christian
11.11.2001	Akseptansetest	0.50	Atle
12.11.2001	Konklusjon	0.51	Atle
13.11.2001	Korrekturlesing	0.71	Atle
14.11.2001	Feilretting	1.00	Magnus

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dette dokumentet er resultatet av vurderingsdelen av den siste praktiske fasen i prosjektet, og ønsker derfor å oppsummere og evaluere hele prosjektgjennomføringen og prosjektets endelige resultat.

1.2 Avgrensning

Dette dokumentet evaluere kun de viktigste aspektene av prosjektet.

1.3 Spesielle definisjoner

Se glossar [GLO].

1.4 Dokumentreferanser

Prosjektdirektivet, Qpro16 2001	[PRO]
Forstudiet, Qpro16 2001	[FOR]
Kravspesifikasjon, Qpro16 2001	[KRA]
Konstruksjon, Qpro16 2001	[KON]
Implementasjon, Qpro16 2001	[IMP]
Systemdokumentasjon, Qpro16 2001	[SYS]
Glossar, Qpro16 2001	[GLO]

1.5 Dokumentoversikt

Evalueringen starter med å beskrive den originale oppgaven og dets utvikling igjennom prosjektets levetid i kapittel 2. I kapittel 3 følger en evaluering av selve arbeidet og alle interne aspekter ved prosjektet.

Kapittel 4 evaluerer det faget dette prosjektet hører til, før man i kapittel 5 evaluerer selve resultatet av prosjektet, og hvilke mål som har blitt oppfylt.

Det hele oppsummeres i kapittel 6 med gruppens totale konklusjon over prosjektet og dets gjennomføring.

2 Oppgaven

Dette kapittelet tar for seg utviklingen av oppgaven opp igjennom prosjektets levetid.

2.1 Original oppgavetekst

Den originale oppgaveteksten var gitt som:

Et "light" verktøy for generering av EJB-baserte systemer.

Input til verktøyet er en beskrivelse av en datamodel/UMLmodel og andre nødvendige parametre i XML (gjerne importert fra Rose, Select eller lignende).

Output fra verktøyet skal som minimum være:
- - SQL create-script for databasen
- - et ferdig generert EJB prosjekt for databasen

Dette kan hjelpe i alle fall oppstarten av mange prosjekter. Bønnene vil da være klare for å implementere forretningslogikk. Prosjektet vil kunne utvides med for eksempel generering av et tilhørende MVC-pattern i for eksempel jakarta-struts.

Det vil bli lagt vekk på at systemet kan fungere som et sett av verktøy sammen med andre verktøy i en verktøykjede og at all generering kan tilpasses prosjektenes behov.

2.2 Spesifiseringssprossen

I prosjektdirektivet [PRO] ble oppgaven spesifisert til:
"… å designe en generell løsning for automatisk generering av komponentklasser ut ifra en samling entiteter og relasjoner…".

I forstudiet [FOR] ble oppgaven ytterligere spesifisert, og satt i sammenheng med nåsituasjon og visjon til kunden. Her ble nye moment dratt inn i oppgaven, og oppgaven ble spisset fra en komplett løsning i designfasen til en demoversjon i implementasjonsfasen.

Da det viste seg at det fantes mange ulike eksisterende løsninger på problemet ble det i forstudiet også lagt vekt på å vurdere om kunden burde gå for en eksisterende løsning eller konstruere en ny.

Etter forstudiets konklusjon om å konstruere en egen løsning spesifiseres oppgaven ytterligere i kravspesifikasjonen [KRA] til å omhandle en generell kodegenerator som på grunnlag av valgte regler og maler generer kode fra en XML-modell. Det legges også

vekt på at det skal være mulig å definere egne regler og maler slik at eventuelle nye arkitekturer kan støttes.

2.3 Kundens endrende syn

I starten av prosjektet virket det som om kunden ikke helt visste hva oppgaven skulle ta tak i og hvordan det skulle løses. Det kunne også virke som kunden ikke var klar over at det fantes eksisterende løsningene på problemstillingen i for eksempel Rational Rose.

Gjennom oppstartsfasen på prosjektet ble det klart at den opprinnelige oppgaveteksten kunne løses fullt ut med ulike eksisterende løsninger og dette medførte at kunden stadig kom med nye vinklinger av oppgaven.

I forstudiet ble kundens ønsker og syn på oppgaven kraftig endret fra en kodegenerator for Container Managed Persistence Beans i EJB til en generell mal og regel basert kodegenerator. Det ble i tillegg lagt større vekt på utforskning av regel og mal basert kodegenerering enn kostnader.

2.4 Vår oppfatning

Etter hvert som prosjektet har skredet frem har gruppen kommet frem til at prosjektets oppgave er å lage en generell mal og regel basert kodegenerator som kan generere ønsket kode avhengig av hva som står spesifisert i regelfilene.

Dette står i kontrast til den ensidige knytningen til EJB baserte systemer som den opprinnelige oppgaveteksten beskriver, men er i tråd med kundens endrende syn og oppgavespesifiseringen underveis.

3 Prosjektarbeidet

Dette kapittelet beskriver hvordan selve prosjektarbeidet har fungert, og de ulike aspektene rundt dette.

3.1 Intern organisering og struktur

Gruppen organiserte seg på et tidlig stadium i prosessen med å samle ansvarsoppgaver i roller. Dette var roller som kun hadde overordnet ansvar angående løsningen på selve oppgaven, men som definerte ansvaret for standardarbeidet som er avgjørende for selve prosjektgjennomføringen.

Den interne strukturen mellom rollene ble forholdsvis flat, men det ble definert en overordnet ansvarlig rolle som Prosjektleder. De andre rollene som ble definert var Teknisk ansvarlig, Økonomisk ansvarlig, Kvalitetsansvarlig og Ambassadør. Se [PRO] for nærmere definering av rollene og deres ansvarsområder samt fordelingen av rollene.

Denne overordende rollefordelingen og strukturen har fungert veldig bra, og vært uendret igjennom hele prosjektet. Sammen med det grundige prosjektdirektivet har dette vært med på å forenkle det daglige prosjektarbeidet i stor grad. Prosessen med å fordele arbeid har til en hver tid vært enkel og effektiv, og den oversiktlige ansvarsfordelingen har dessuten gjort det enklere for gruppemedlemmene å finne frem til nødvendig informasjon via riktig person.

3.2 Internt samarbeid og samhandling

Dette kapittelet omhandler hvordan gruppen har arbeidet sammen gjennom prosjektets gang.

3.2.1 I starten

Ingen av gruppens fem medlemmer kjente hverandre godt fra før, og dette medførte at det første kundemøtet som også var det første gruppemøtet ble kunstig og ikke veldig produktivt. Rett etter kundemøtet satte gruppen imidlertid i gang med å organisere seg.

Først ble alle gruppemedlemmene sine ambisjoner med prosjektet fastlagt og disse var så like at gruppens felles ambisjon ble fastlagt forholdsvis kjapt. Deretter ble de ulike rollene fastlagt og fordelt etter ønske og kompetanse til gruppens ulike medlemmer. Denne tidlige organiseringen var med på at gruppen tidlig kom i gang med å produsere dokumenter og fikk en standardisert god kommunikasjon ut mot kunde og veileder.

3.2.2 Møtevirksomhet

Gruppens interne møter har fungert godt. Hver fredag avholdt gruppen et statusmøte, og utenom disse ble spontane møter avholdt når det var behov for det. På alle møtene har prosjektleder hatt rollen som ordstyrer for å holde gruppen på tema og få møtene gjennomført så effektivt så mulig.

Statusmøtene hver fredagene hadde et standard innhold der gruppen så hva som var gjort i uken som hadde gått, og ble enige om hvor og hvordan man skulle gå videre i prosessen. Disse møtene ble alltid avsluttet med et fellesmåltid med grøt og saft i skolekantinen.

I starten av hver fase har gruppen i felleskap bestemt innholdet i dokumentene og aktivitetene som skal produseres og fordelt de spesifikke delansvarene ut til gruppens medlemmer. I denne fastleggingsprosessen har ’6 thinking hats’ metoden blitt brukt iherdig til kreativt samarbeid for å komme opp med best mulig innhold i fasen. Den spesifikke ansvarsfordelingen i de ulike fasene ble gjort på grunnlag av ønsker og kompetansen til gruppens medlemmer. Da det ikke var spesielle ønsker var rollefordelingen retningsledende.

I tillegg til det faglige samarbeidet har gruppen også funnet på en god del sosiale aktiviteter ved siden av grøtspisingen, her nevnes spesielt spillkveldene, kinoturene og golfingen som sosialt samlende aktiviteter.

3.2.3 Konflikthåndtering

Helt fra starten av prosjektet ble gruppen enige om å si ifra dersom det var noe som plaget en av gruppemedlemmene eller en konflikt var i ferd med å oppstå. Til tross for dette har ingen synlige konflikter oppstått. Selv under de prøvelser og veiledning som gruppen ble utsatt for under gruppedynamikk seminaret i regi av Luftkrigsskolen fungerte gruppen godt, og fikk skryt for å være ekstremt løsningsorientert.

Grunnene til denne mangelen på konflikter i prosjektarbeidet er mange. Men den kanskje viktigste grunnen er at alle gruppens medlemmer hadde samme målsetting og ambisjon med prosjektet. Dette faktumet bidro til at alle var innstilt på å legge like mye arbeid i prosjektet, og på denne måten unngikk gruppen en god del konflikter. At gruppen har vært veldig løsningsorientert, har virket konfliktdempende og kanskje også hindret potensielle konflikter å komme til overflaten.

En annen viktig grunn til lavt konfliktnivå var at alle gruppens medlemmer hadde adskilte ansvarsområder, der den personen med ansvaret var eksperten på området og hadde det siste ordet når det skulle taes avgjørelser. Alle var leder og ekspert innefor sitt område. Dermed ble alle konfliktområdene løst via diskusjon der alle fikk muligheten til å legge frem sine synspunkt. Den som hadde ansvaret for det området tok da avgjørelsen på bakgrunn av denne diskusjonen. Slike diskusjoner førte flere ganger til at nye valg ble tatt, men det var gjort på et veloverveid grunnlag. Alle fikk også lov til å uttale seg når det var et problem som var oppe til diskusjon, alle ble hørt, og alle argumenter ble vurdert. Diskusjonene i gruppen har alltid endt i enighet og det har aldri vært nødvendig

for prosjektleder å ta avgjørelser der gruppen ikke var enig. Dette er i tråd med gruppens definisjon av prosjektleders rolle. Hans oppgave var ikke å fordele oppgaver gjennom beordring, men å oppmuntre til ansvar og initiativ, og legge forholdene mest mulig til rette for de andre i gruppen.

Den siste grunnen til det lave konfliktnivå som nevnes her er gruppens homogenitet. Alle gruppens medlemmer ligger på samme faglige nivå, og er genuint interessert i det fagfeltet som det jobbes med. Dette har vært med på å hjelpe lagfølelsen i gruppa og kompetansedelingen da alle gruppens medlemmer 'snakker samme språk'.

Gruppens homogenitet har også muliggjort den store bruken av parallellitet som måtte gjøres i deler av denne prosjektgjennomføringen. Gruppens medlemmer har bidratt til å diskutere andre områder av prosjektet enn akkurat den de holder på med, og kommet med nye ideer og betraktninger som har vist seg verdifulle.

3.3 Samarbeidet med kunden

Kunde og oppdragsgiver for prosjektet har vært EDB ASA representert ved Ketil Aasarød. Kontaktpersonen er selv sivilingeniør i data, og har selv hatt faget kundestyrt prosjekt i sin tid.

Samarbeidet med kunden har gjennom prosjektet fungert veldig godt. I starten gikk det litt sakte å få tilbakemeldinger fra kontaktpersonen, men dette bedret seg etter hvert. Avgrensingen og finformuleringen av oppgaven gikk også noe sakte da oppdragsgiver ikke var klar over alle de eksisterende løsningene som fantes på den opprinnelige oppgaven.

Det faktum at EDB ASA og Ketil Aasarød var lokalisert i Trondheim muligjorde faste ukentlige møter med kunden. Dette har medført at gruppen har fått innspill fra kunden tidlig i alle faser, og at mange mulige misforståelse og feiltolkninger fra prosjektgruppen har blitt rettet opp på et tidlig stadium. Det må også nevnes at oppdragsgiver installerte og satte opp en webserver til gruppen slik at en del tid ble spart på dette.

Den nære kontakten med kunden, spesielt i forhold til oppgavespesifisering og kravspesifikasjon, har i stor grad vært med på å sikre at det prosjektet som nå leveres er det kunden vil ha.

3.4 Prosjektfasene

Dette kapittelet beskriver hvordan gruppen opplevde de ulike fasene i prosjektet.

3.4.1 Planlegging

Planleggingsfasen var i starten preget av gruppens medlemmer ikke kjente hverandre. I starten ble det en del posisjonering i forhold til hverandre, men dette gikk over når rollene ble definert og gruppens medlemmer ble bedre kjent.

Etter at innholdet var fastlagt ble arbeidet delt i noenlunde like bolker mellom gruppens medlemmer og utført parallelt og forholdsvis uavhengig av hverandre.

Etter hvert som fasen skred frem viste det seg at arbeidsmengden i fasen var betraktelig større enn ventet, men grunnet den effektive arbeidsdelingen ble prosjektdirektivet produsert i rute.

3.4.2 Forstudiet

Før forstudiet startet hadde gruppen vært på et kurs for å lære ”6 thinking hats”-teknikken. Denne ble også brukt i oppstartsfasen av forstudiet, og førte til at vi fort fikk et godt skjelett på innholdet i forstudiet. Gruppen fikk begynt å titte på forskjellige teknologier, og det virket mer interessant enn det vi hadde gjort i planleggingsfasen.

Mye tid gikk med til en markedsundersøkelse for å finne ut hva slags produkter i markedet som kunne være en løsning på prosjektets problemstilling. Denne undersøkelsen ble veldig stor, og tok mye tid. Man følte til tider at det strittet litt imot.

Den siste prosessen fram mot en konklusjon var ganske vanskelig. Etter å allerede ha brukt mye tid på forstudiet skulle en god konklusjon taes. Det at forstudiet egentlig pekte litt annerledes en det kundens ønsker gjorde, medførte at det ble enda vanskeligere. Men det bli til slutt utarbeidet gode argumenter, og det ble konkludert med at kundes ønske skulle arbeides videre med.

Det at det gikk så pass mye tid på forstudiet medførte at kravspesifikasjonen ble påbegynt i parallell et stykke ut i forstudiefasen. Tre stykker jobbet videre med forstudiet og to startet opp med kravspesifikasjonen. Dette viste seg å være en god løsning.

3.4.3 Kravspesifikasjonsfasen

Her kjørte vi også ”6 thinking hats” på innholdet. Et god innholdsfortegnelse ble skissert opp, og dette medførte at arbeidet gikk ganske fort og greit. Som sagt var det i hovedsak to stykker som jobbet med kravene og det viste seg at tidsforbruket var såpass lavt at det var tilstrekkelig. Noen av de andre gruppemedlemmene gikk imidlertid igjennom kravspesifikasjonen når den var omtrent ferdig slik at flere synspunkter ble tatt med i betraktningen.

Det var litt problematisk å formulere tydelig krav, og etter råd fra hovedveileder ble det brukt en del tid på å få dem tydelig nok.

3.4.4 Konstruksjonsfasen

Også her ble det først kjørt ”6 thinking hats” på innholdet. For å komme i gang med modelleringen brukte vi en merkelappsmetode for å få etablert alle viktige klasser i systemet. Det var mange teknologier å settes seg inn i, og det medførte en del lesing, men det var også interessant. Arbeidsoppgavene var klart fordelt, og etter en litt tung start med å sette seg inn i teknologiene, gikk det veldig greit å få opp designet. En i gruppa implementerte et bevis av konseptet, og syntes det var gøy. Dette ble nyttig også i implementasjonsfasen.

Som sagt gikk det veldig greit med konstruksjonen helt til den detaljerte systembeskrivelsen skulle skrives. Da ble det en del kjedelige timer for å få det på plass. Noen gikk også over til å starte implementeringen mot slutten av fasen.

3.4.5 Implementasjonsfasen

Implementasjonsfasen fikk en ”flying start” i og med at vi hadde modellert så godt, og kunne generere kodeskjelettet direkte fra Rational Rose. Det var i hovedsak 3-4 som stod for implementeringen, og oppgavene var klart fordelt mellom dem. Dette var tydeligvis et arbeid som var gøy, for implementeringen gikk fort. Det eneste som hele tiden tok litt tid var å få til regler, maler og en generatormodul som jobbet godt sammen. Det meste kom fort på plass, og etter en del feilfinning var systemet klart. Totalt antall kodelinjer ble på ca. 2700. De erfaringene som ble gjort i forbindelse med regler og maler kommer sannsynligvis godt med for videre utvikling av systemet.

3.4.6 Vurdering og Testing

For å få et godt skjelett på vurderingsdokumentet ble det kjørt ”6 thinking hats”. Innholdsfortegnelsen ble fastlagt, og innfyllingen startet. Det var litt tungt arbeid, men veldig nyttig for å tenke gjennom hvordan arbeidet har vært.

Når det gjelder testen var det ganske ullent å angripe, men etter hvert løsnet det. Å lage testplaner når lite var implementert er vanskelig fordi det finnes lite standardløsninger å ty til. Test må konstrueres ganske spesifikt mot det produktet som skal testes. Løsningen med at hver programmerer lager sin egen testklasse som tester sin egen modul fungerte svært bra. Dette gjorde at programmerer måtte tenke på at modulen skulle testes. Testing tar tid, og avdekket en del skjulte feil. Disse ble rapportert tilbake til programmerer, rettet, ført inn i endringsloggen før modulen ble testet på ny. Ettersom vi hadde kodeskjelettet på forhånd var modulene allerede ganske godt integrert med hverandre. Selve testen av de ulike modulene må derfor sies å ha gått rimelig greit.

Kodeinnspeksjonen ble foretatt av de to som hadde programmert minst. Sjekklista fra kravspesifikasjonen var litt for grundig, og ikke alt var like aktuelt, men de viktigste punktene ble plukket ut og sjekket. På grunn av tidsbrist ble det kun foretatt en formell kodeinspeksjon av de tre viktigste klassene. Alle som hadde programmert fikk i oppgave å rydde opp i sin egen kode.

Integrasjonstest ble utført av kvalitetsansvarlig i delvis samarbeid med de respektive programmererne etter hvert som det var uklare momenter i implementasjonen. Det ble foretatt kjøretester som avdekket noen feil. Integrasjonen av moduler til komponenter har gått veldig greit. Dette skyldes i stor grad at skjelettet var ferdig på forhånd og at hver enkelt programmerer hadde fått tildelt sine moduler og klasser som skulle implementeres.

3.4.7 Presentasjon

Presentasjonsfasen ble litt preget av være den siste fasen i prosjektet fordi den ble en del av sluttspurten til prosjektet. Fasen startet med en felles kreativ sesjon med ”6 thinking hats” metoden for å komme frem til hva presentasjonen skulle inneholde og hvordan den skulle utføres.

Presentasjonen ble avholdt den 15. november klokken 17.15 i rom B209 i Elektrobygget ved NTNU - Gløshaugen.

3.5 Risikohåndtering

Her vil det bli tatt opp hvordan risikohåndtering er gjennomført i prosjektet. Både planlegging, risikoer som inntraff og tiltak vil bli beskrevet.

3.5.1 Praktisk gjennomføring

Før hver fase satte vi opp risikoer som vi mente kunne inntreffe. Dette var nyttig slik at vi kunne være bevisst på å prøve å unngå disse. Det ble også satt opp tiltak til hver risiko, og hvem som skulle være ansvarlig for å gjennomføre tiltakene.

3.5.2 Inntrufne risikoer og gjennomførte tiltak

Av risikoer som inntraff var det i hovedsak tidsrisikoen. Denne inntraff til gjengjeld på flere av fasene. Vi fulgte da opp med å innføre tiltakene som var satt opp, og dette medførte at det likevel gikk ganske smertefritt.

Da det i høst var Uka var også dette satt opp som risiko. En av gruppemedlemmene var involvert som frivillig under Uka, og han måtte bruke en del tid på det. Det var også diverse tilstelninger under uka som medlemmene var med på, og alt dette førte til at det av og til ble brukt litt mindre tid på prosjektet enn planlagt. Som nevnt ble tiltakt satt inn, og det ble ikke et for stort problem til at vi fikk gjennomført det vi skulle.

Et problem, som vi hele tiden var bevisst på, var at undersøkelsene i forstudiet kunne bli preget av subjektive oppfatninger. For å unngå dette prøvde vi å få synspunkter fra flere i gruppa på de samme undersøkelsene, samt å innhente informasjon fra andre eksterne kilder.

3.5.3 Uforutsette risikoer som oppsto

Som uforutsette risikoer var det en del softwareproblemer som oppstod. Vi hadde valgt å benytte Microsoft Word som verktøy for dokumentskriving, men programmet var ikke alltid like villig når det var store dokumenter på opp mot 100 sider som skulle redigeres. Til tross for en del småproblemer gikk imidlertid ingen data tapt.

Midt i prosjektperioden gikk maskinen med alle dokumentene til gruppa ned. Dette skyldtes at operativsystemet på maskinen rett og slett sluttet å fungere. Dette var først litt krise i og med at det var en stund siden siste backup var tatt, men etter å ha reinstallert maskinen viste det seg imidlertid at alle data på diskene var intakte. Etter dette ble det tatt hyppigere backup.

Det eneste tapet av data skjedde da Rational Rose feilet, og hele modellen for systemet ble borte. Vi hadde da brukt modellen til alt vi egentlig skulle bruke den til, men den hadde vært grei å ha i ettertid. Det viste seg at en litt eldre versjon lå på en annen plass, og det skulle ikke så mye til før den var oppdatert.

3.6 Tidsforbruk

Estimering av tidsforbruk i et IT prosjekt er et veldig komplekst fagfelt, som fortsatt er veldig usikkert og vanskelig å få til skikkelig i praksis. Dette kapittelet tar for seg planleggingsprosessen og hvilke planer som ble lagt i prosjektet. Til slutt vises det faktiske forbruk at timeverk i prosjektet. Den gjeldende planen står i prosjektdirektivet og alle de tidligere planene som har eksistert ligger vedlagt [PRO] .

3.6.1 Opprinnelig plan

Den opprinnelige planen ble lagt i starten av prosjektet på grunnlag av daværende forståelse av oppgaven og dets arbeidsfordeling. I tillegg ble faktisk timeforbruk i tidligere prosjekter gransket og lagt til grunne.

Det ble regnet med runde tall og man så for seg at forstudiet, konstruksjonsfasen og implementasjonsfasen ble de største fasene. I tillegg så man for seg at gruppen produserer ca 125 timer per full arbeidsuke og dermed 1500 timer i løpet av prosjektets levetid.

I tilegg så man for seg et en noenlunde ren vannfallsmodell, der de ulike fasene etterfulgte hverandre uten noen stor grad av overlapp. Det eneste unntaket fra dette var alle den eksperimenteringen som skulle foregå i fasene før implementeringen.

Ukenr	1	2	3	4	5	6	7	8	9	10	11	12
Fase Startdato	27.aug	03.sep	10.sep	17.sep	24.sep	01.okt	08.okt	15.okt	22.okt	29.okt	05.nov	12.nov
Planlegging												
Forstudie			300t - 20 %									
Kravspek					225t -15 %							
Konstruksjon												
Implementasjon									300t - 20 %			
Test & vurdering											100t - 7 %	
Presentasjon												

Figur 3.1 – Opprinnelig plan av timeforbruk og fremdrift i prosjektet

3.6.2 Brukstillefelleestimering

Midt i prosjektets levetid ble en metode for å estimere tidsbruk på grunnlag av brukstilfeller tatt i bruk etter påtrykk for hovedveileder. Denne metoden beregnet gjenværende tid på konstruksjon og implementasjon av system beskrevet ved bruk av et spesielt brukstilfelleformat. Metoden baserte seg på kompleksiteten til brukstilfellene og omgivelsesfaktorene for å estimere hvor mye tid som kreves.

Denne ble brukt i kravspesifikasjonen [KRA] og metoden gav oss et estimat på 488,48 timer på konstruksjon og implementasjon for de brukstilfellene som vi skulle konstruere.

Metoden var enkel og rask å bruke, men da dette prosjektet har nesten all intern komplikasjon i et brukstilfelle (generere kode fra XMI) følte vi at metoden ikke var helt egnet til å estimere vårt tidsbruk.

3.6.3 Endret planer

Igjennom prosjektet har gruppen vært nødt til å endre den opprinnelige planen to ganger for å planlegge ut fra reelt tidsforbruk og fremgang. Disse endringene har blitt foretatt i felleskap etter at statusmøtene har avdekket store avvik fra den gjeldene planen når det gjaldt fremdrift og timeforbruk.

3.6.4 Faktisk forbruk

Har valgt å skille mellom fremdrift og timeforbruk i beskrivelsen av det faktiske forbruket.

Fremdrift

Den faktiske fremdriften til prosjektet skilte seg etter hvert radikalt fra den opprinnelige fremdriftsplanen. På grunn av at noen av fasene i tok mye lengre tid å bli ferdige med enn andre måtte det taes i bruk mye parallellitet i gruppens arbeid. Bruken av parallellitet

fungerte veldig bra, og har vært en av hovedgrunnene til at prosjektet kom i mål på den måten det til slutt gjorde.

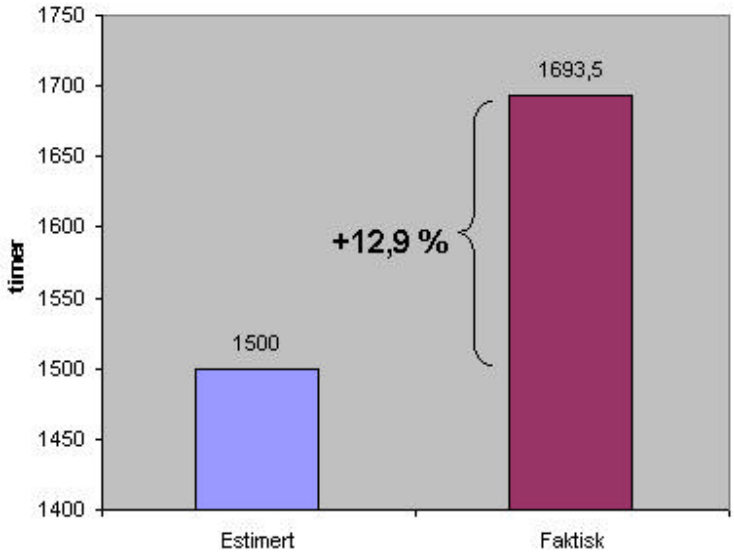
Spesielt forstudiet og konstruksjonsfasen var mye mer omfattende enn forventet, og dette medførte at fasene tok mye tid. I slutten av disse fasene var derimot arbeidet mest fokusert på små utbedringer, mens essensen i hovedsak var fastlagt. Gruppen har da startet etterfølgende faser i parallell på grunnlag av essensen i foregående faser, og dette har fungert godt.

Uke	1	2	3	4	5	6	7	8	9	10	11	12
Fase Startdato	27.aug	03.sep	10.sep	17.sep	24.sep	01.okt	08.okt	15.okt	22.okt	29.okt	05.nov	12.nov
Planlegging	**2 uker**											
Forstudie		**4,5 uker**										
Kravspek					**2,5 uker**							
Konstruksjon							**4 uker**					
Implementasjon									**2,5 uker**			
Test & vurdering											**1,5 uke**	
Presentasjon												**1 uke**

Figur 3.2 – Faktisk fremdrift i prosjektet

Timeforbruk

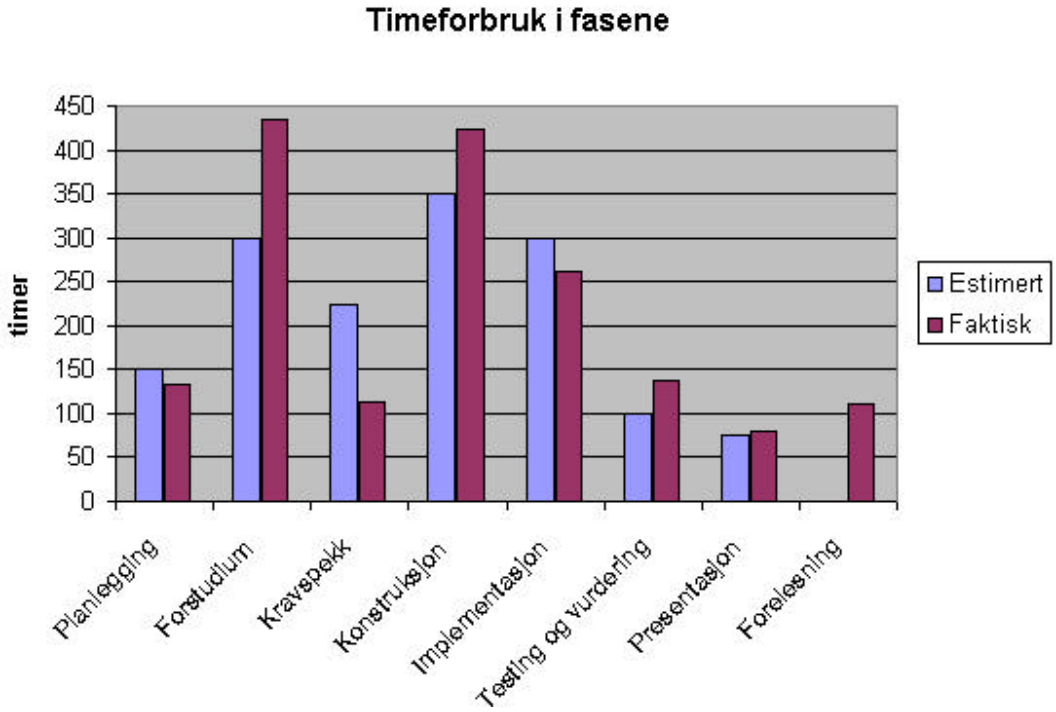
Det totale faktiske timeforbruket viste seg å bli noe større enn de opprinnelige planene. Prosjektgruppen fullførte prosjektet med et tidsforbruk som lå 12,9% over det estimerte.



Figur 3.3 – Totalt timeforbruk i prosjektet

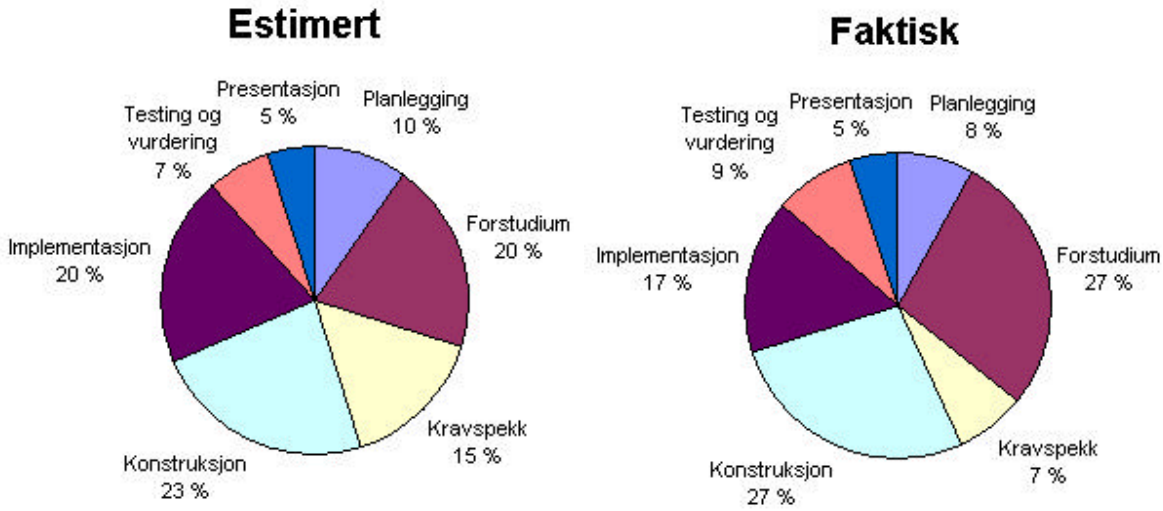
Dette skyldes i stor grad gruppens ønske om å implementere noe reelt som faktisk fungerte, isteden for å avgrense oppgaven mer i gjennomføringen av implementasjonsfasen.

I fordelingen av timene i de ulike fasene ble det derimot større forskjeller. Forstudiet og konstruksjonsfasen ble mye større enn forventet, mens kravspesifikasjonen ble mye mindre enn først forventet.



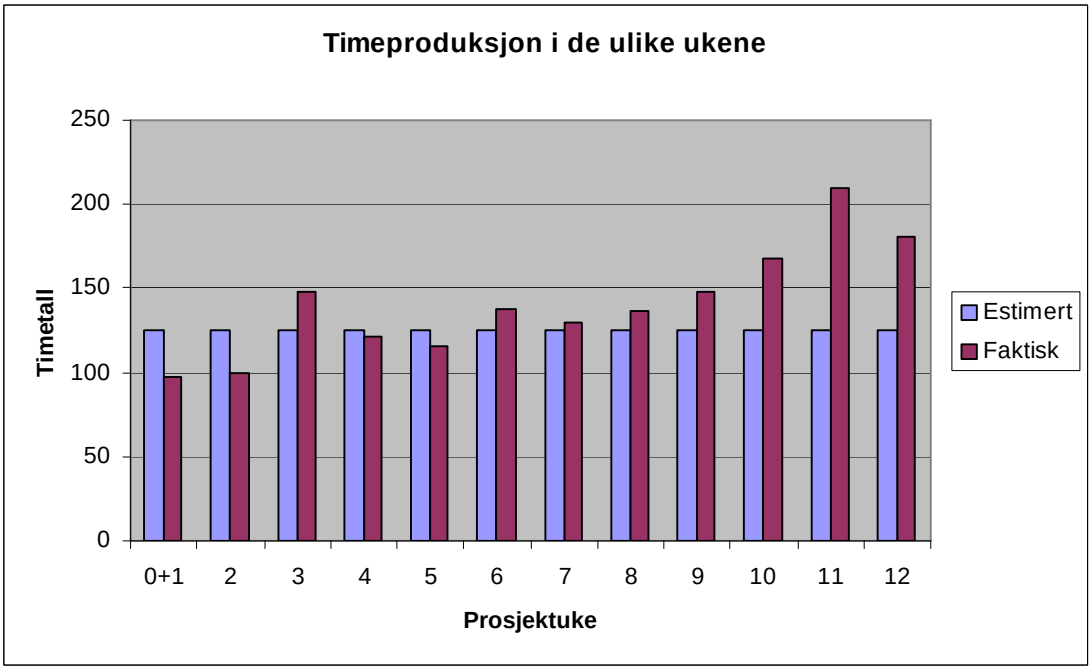
Figur 3.4 – Timeforbruk i de ulike fasene

Sett på en litt annen måte kan dette illustreres som gjort i figur 3.4. Her er forelesningene trukket ut da disse ikke henger direkte sammen med de ulike fasene.



Figur 3.5 - Estimert og reelt timeforbruk i de ulike fasene av prosjektet

Timeproduksjonen i prosjektets ulike uker varierte en del, men generelt sett jobbet gruppen særdeles jevnt, og totalt over hele prosjektet skiller det kun 15 timer mellom de ulike gruppemedlemmene sin timeproduksjon. Som vi etter hvert ventet oss, ble timeproduksjonen høyere mot slutten av prosjektet.



Figur 3.6 – Timeproduksjon i de ulike ukene av prosjektet.

4 Faget SIF8080 - 'Kundestyrt prosjekt'

Dette kapittelet evaluerer faget SIF8080 'Kundestyrt prosjekt' som dette prosjektet har vært en del av ved Institutt for Datateknikk og Informasjonsvitenskap (IDI) ved Norges Teknisk-Naturvitenskapelige Universitet (NTNU).

4.1 Introduksjonen til faget

Introduksjonsforelesningen til faget var god og informativ, og man fikk en god innføring i hva man hadde i vente. Positivt med utdeling av kompendiet, og gruppelistene på papir slik at man hadde noe konkret å forholde seg til.

Kompendiet inneholdt mesteparten av den nødvendige informasjonen, men virket uoversiktlig og kunne med hell vært omorganisert. I tillegg er kompendiet veldig tynt i fasene 'Konstruksjon' og 'Implementasjon' i forhold til de andre.

Det første møtet med kunden kunne med hell vært utsatt noen dager slik at gruppen hadde hatt tid til å organisere seg først. Da kunne dette første møtet blitt mye mer produktivt, og gruppen hadde hatt muligheter til å forberede seg på den gitte oppgaven. Slik som det ble nå var det kun monolog fra kunden til gruppen, og møtet gav ikke noe stor gevinst.

4.2 Samarbeid med hjelpeveileder

Samarbeidet med hjelpeveileder har fungert veldig godt, og selve konseptet med en student fra siste årskurs som hjelpeveileder er særdeles bra.

4.3 Samarbeid med hovedveileder

Samarbeidet med hovedveileder har fungert forholdsvis bra, med faste ukentlige møter. Hovedveileder kunne med hell fått mer tid til faget, slik at han kunne i større grad bidratt med klarere tilbakemeldinger og bedre forberedelser til møtene.

4.4 Tilgjengelige ressurser

De ressursene som er gjort tilgjengelige for gruppen er i stor grad en datamaskin, skriverkvoten til hver enkelt og gratis kopiering og innbinding av rapportene til sensor, kunde og faglærer.

4.4.1 Datamaskin

Gruppa er veldig fornøyd med den datamaskinen som er blitt tildelt. Vi fikk ganske raskt skaffet oss administratorrettigheter slik at det var mulig å installere de programmene vi trengte.

Datamaskinen har ikke hatt noen problemer med å møte prosjektets krav til minne, ytelse eller lagringsplass. Datasalen der maskinen har stått plassert har vært flittig brukt av gruppens medlemmer, det vil si ikke bare den tildelte maskinen.

4.4.2 Skriverkvote

Skriverkvoten på IDI er på 500 sider per bruker per semester, noe som vil tilsvare 2500 sider for hele prosjektgruppen. Dette har imidlertid ikke vært noe problem ettersom skriverkvoten bare fungerer på realfagsbygget og ikke i IT-bygget eller på VM-brakka.

4.4.3 Kopiering og innbinding

Preleveransen ble bundet inn hos Tapir Bokhandel. Den ferdige rapporten ble kopiert opp og bundet inn i fire eksemplarer (sensor1, sensor2, hovedveileder, kunde) gratis hos IDI. Vi forsøkte å få rapporten i farger, men det slo gruppa raskt fra seg da det viste at det ville koste ca. 6000 kroner for 5 fargeeksemplarer hos Tapir Trykk.

4.5 Kurset i gruppedynamikk

I dette faget lå et obligatorisk kurs i gruppedynamikk. Dette kurset var delt i to og dette kapittelet evaluerer begge de to delene.

4.5.1 Del I: ’6 thinking hats’

Denne delen var et seminar der gruppen lærte kreativ tenkning ved bruk av ”6 thinking hats” metoden. Metoden var særdeles lett å lære og har vist seg å være veldig i nyttig i den videre prosjektgjennomføringen da gruppen har brukt denne i oppstarten av de fleste fasene siden.

4.5.2 Del II: ’Gruppedynamikk’ ved Luftkrigsskolen

Denne delen var en ut-i-skogen-dag med luftkrigsskolen der gruppen ble satt på flere praktiske oppgaver og utfordringer. Dagen var veldig lærerik og gav mange av gruppens medlemmer et klarere syn på hvem de var i gruppeprosessen. Oppgavene og utfordringene var også med på å bringe medlemmene i gruppen nærmere hverandre, og bli bedre kjent. I tillegg ble det gitt en personlig tilbakemelding fra alle til alle i gruppen, noe som også var nyttig.

4.6 Forelesningene

Forelesningene i faget har vært av litt blandet kvalitet. Selv om emnene generelt sett er bra og nyttige for prosjektgjennomføringen kan mye forbedres på selve forelesningene. Dette går hovedsakelig på pedagogiske momenter, og det er ikke forbausende at det er de eksterne foredragsholderne som har stått for de beste forelesningene.

4.7 Faglig utbytte

Det faglige utbyttet i faget har vært stort, og dette har helt klart vært det mest relevante faget i utdannelsen til nå i forhold til den arbeidssituasjon som vi ender opp i når vi er ferdig utdannet. Kanskje spesielt på de litt mer menneskelige aspektene har utbyttet vært stort. Det å jobbe som en selvstendig gruppe over så lang tid med såpass høyt arbeidspress har vært en lærerik prosess, som gruppens medlemmer føler har gitt dem mye.

Det tekniske utbyttet av faget er veldig oppgavespesifikk, men for vår del har det vært stort. Gruppen har fått et grundig innblikk i nye teknologier som EJB og XML.

4.8 Belastning

Faget er i utgangspunktet satt til å ta 310 timer for alle gruppemedlemmene igjennom hele prosjektet, dette utgjør om lag 26 timer i uken. Først så det ut vi skulle lande akkurat på det antallet timer som var satt opp, men på grunn av interessen for å få laget en fungerende implementasjon gikk vi noe over dette estimatet.

Med en full arbeidsuke blir det litt knapp tid til å henge med i de andre fagene, spesielt med de litt tyngre tekniske valgfagene uten å ta mye av helgen i bruk. Dette har medført at man har måttet nøye seg med å kun gjøre øvingene, og heller satse på å ta igjen lesingen etter prosjektet.

Denne oppgaven har også vært noe stor for en gruppe på bare fem personer, og mye mer kunne vært utrettet dersom man hadde vært en person til på gruppen.

4.9 Mulige forbedringer

I tillegg til de forbedringene som tidligere er nevnt ser gruppen følgende andre mulige forbedringer:

?? Bedre siling på oppgavene, vår oppgave fantes det i grunnen en ferdig løsning for, noe som er litt demotiverende.
?? Mer oversiktlig kompendium, og litt mer informasjon om noen av de mer tekniske fasene som konstruksjon og implementasjon.
?? Mer pedagogiske forelesninger, legg gjerne mer vekt på praktiske eksempler.
?? Bør være seks medlemmer på gruppene, litt for lite kan utføres med kun 1500 timer på et helt prosjekt.

4.10 Totalvurdering

Dette har vært et av de beste fagene gruppens medlemmer har hatt til nå, selv om belastningen har vært forholdsvis stor har utfordringene vært artige og relevante.

5 Resultatet av prosjektet

Dette kapittelet evaluerer resultatet av prosjektet ut ifra kundens og gruppens mål, samt beskriver hvordan prosjektet kan videreutvikles.

5.1 Levert produkt

En generell regelbasert kodegenerator, der alle innstillinger kan forandres ved hjelp av regler og maler. Denne kodegeneratoren er dermed så generell at den kan generere kode fra en vilkårlig modellrepresentasjon i XML dersom det er definert gyldige regler og maler for ønsket format på resultatkoden.

5.1.1 Systemtest

I dette avsnittet har vi testet det ferdige produktet opp mot kundens krav til systemet. Dette gjøres ved å gå igjennom testplanen i kravspesifikasjonen. Alle tester med kritiske eller høye krav må oppfylles for at systemet skal bli godkjent.

Tegnforklaring:

✔ Markerer at testen er godkjent.

✘ Markerer at testen underkjent.

Test mot inndata krav

TestID	Beskrivelse	Kommentar	Prioritet	Godkjenning
Inndata Test 1	Kan programmet ta en generell UML modell som inndata?	Taes inn som XMI eksportert fra Rose. Siden XMI 1.0 ikke helt så entydig må man forandre på reglene hvis man for eksempel vil ha inn XMI fra TogetherSoft Control Center eller fra Select Enterprise.	Kritisk	✔
Inndata Test 2	Kan programmet velge i hvilke kataloger kildekoden som genereres skal legges i?	Dette fungerer greit. En liten bug er at man må trykke to ganger på Open for å velge katalog i GUI.	Høy	✔
Inndata Test 3	Kan programmet velge hvilken arkitektur det skal genereres kode for?	Valg av arkitektur velges i GUI under ”Code Architecture”.	Høy	✔

Inndata Test 4	Dukker det opp en feilmelding i programmet hvis inndata er av et uforståelig format?	Meldingen ”Cannot Parse XMI file” dukker opp dersom man forsøker å parse en fil som ikke er XMI.	Middels	✔
Inndata Test 5	Kan programmet importere XMI fra Rose?	Programmet takler XMI eksportert fra Rational Rose. Relasjoner blir ikke håndtert, og må sees på som en videre utvidelse av programmet.	Kritisk	✔
Inndata Test 6	Kan programmet importere XMI fra Select?	Selve programmet takler import av XMI fra Select, men for at dette skal fungere må det gjøres endringer i reglene for XMIen som skal importeres.	Middels	✘
Inndata Test 7	Er det mulig innen en rimelig tidsfrist å manuelt endre eller skape inndata til kodegeneratoren ?	Det er faktisk mulig å ta en hvilken som helst XML-input til programmet så lenge man definerer en egen regelfil for denne.	Middels	✔

Test mot parse- og genereringsprosess krav

TestID	Beskrivelse	Kommentar	Prioritet	Godkjenning
Parse Test 1	Kan programmet generere kode for EJB 2.0?	EJB 2.0 koden som blir generert kompilerer. Man må bare huske på å ha satt classpath for Java 2 Enterprise Edition.	Kritisk	✔
Parse Test 2	Kan programmet generere kode for COM språk?	Programmet er laget for å kunne generere kode for alt som det kan lages regler for. Det er eksperimentert med regler for å generere Visual Basic, men disse er ikke fullført.	Middels	✘

Parse Test 3	Tar det under 5 minutter å generere all kode for en normal modell med 10 data aksessobjekter og 10 brukstilfeller?	Genereringen tar svært kort tid. Det tar kun noen få sekunder å generere en modell med 10 dataaksessobjekter. Tidsforbruket er dermed ikke noe problem.	Middels	✔
Parse Test 4	Tar det under 2 minutter å generere et subsett (SQL, data aksessobjekter, forretningsobjekter, GUI)?	Man kan velge å generere kun SQL eller kun Kode. Genereringen tar kun kort tid og vil ikke være noe problem.	Middels	✔
Parse Test 5	Kan programmet automatisk generere ulike SQL dialekter for ulike databasesystemer?	Programmet genererer ulike SQL dialekter ved at man velger dette i GUI. Alternativene som er implementert er for Oracle og MS SQL Server.	Middels	✔
Parse Test 6	Er det mulig å enkelt kunne lage sine egne regler og maler?	Programmet har utstrakt støtte for å kunne lage egne maler og regler slik at ønskede nye språk kan genereres.	Lav	✔

Test mot utdata krav

TestID	Beskrivelse	Kommentar	Prioritet	Godkjenning
Utdata Test 1	Kan programmet generere SQL kode som oppretter databasen?	Det er blitt laget regler for å opprette databasen i Oracle og MS SQL. For å støtte andre databasearkitekturer må det lages egne regler for disse.	Kritisk	✔
Utdata Test 2	Kan programmet generere aksessobjekter mot tabellene i databasen?	Programmet kan generere aksessobjekter. Det er laget regler for EJB 1.0 og 2.0 og Visual Basic.	Kritisk	✔
Utdata Test 3	Kan programmet generere aksessobjekter for forretningsprosesser?	Dette punktet er støttet av designet men ikke av implementasjonen.	Middels	✘

Utdata Test 4	Kan programmet generere brukergrensesnitt for forretningsprosessene?	Dette punktet er støttet av designet men ikke av implementasjonen.	Lav	✘
Utdata Test 5	Kan programmet generere SQL kode støttet av MS SQL server?	Det er laget regler for å generere SQL kode som oppretter databasen på en MS SQL Server.	Kritisk	✔

Test mot maskinvare- og programvarekrav

TestID	Beskrivelse	Kommentar	Prioritet	Godkjenning
Maskin Test 1	Kan programmet kjøres på en 350 MHz Pentium med 256 MB minne?	Prototypen som er laget har ingen store krav til verken ytelse, minne eller lagringsplass.	Høy	✔
Maskin Test 2	Kan programmet kjøres i et MS Windows operativsystem?	Programmet kjører uten problemer på MS Windows operativsystem.	Kritisk	✔
Maskin Test 3	Kan programmet kjøres i et UNIX/Linux operativsystem?	Ettersom programkoden er skrevet i java og ikke inneholder noen maskinvareavhengigheter skal det kunne eksporteres til alle maskinvareplattformer som kan kjøre java applikasjoner.	Lav	✔

Test mot utviklingsmiljø krav

TestID	Beskrivelse	Kommentar	Prioritet	Godkjenning
Miljø Test 1	Er programmet utviklet i henhold til Sun SDK 1.3?	Programkoden er utelukkende utviklet i Sun SDK 1.3. Alle regelfiler er spesifisert ved hjelp av XML.	Høy	✔
Miljø Test 2	Er det blitt benyttet tredjeparts pakker?	Det er blitt benyttet en ekstern XML. Denne pakken er blitt godkjent av kunden.	Kritisk	✔

Test mot generatorkode og dokumentasjon krav

TestID	Beskrivelse	Kommentar	Prioritet	Godkjenning
Kode Test 1	Er klassene fornuftige?	Kunden har sett på designet av generatoren og synes det så fornuftig ut.	Høy	✔
Kode Test 2	Er generatorkoden kommentert med JavaDoc?	All programkode som tilhører generatoren er godt kommentert med forklarende JavaDoc.	Høy	✔
Kode Test 3	Følger det med en installasjons- og brukerdokumentasjon?	Installasjons- og brukerdokumentasjon finnes i et eget dokument som heter Systemdokumentasjon. For mer detaljert informasjon henvises det til konstruksjons-dokumentet.	Middels	✔

Konklusjonen blir at de fleste testmomentene er oppfylt. Det er kun to tester med middels og en med lav prioritet som ikke passerer. Kravet til godkjenning av produktet i sin helhet sier at alle høye og kritiske krav skal være oppfylt, samtidig som en helhetsvurdering taes med i denne betraktningen. Totalt sett godkjennes derfor produktet prosjektets akseptansetest.

5.2 Kundens mål

For at kunden skulle kunne avgjøre om hans mål med prosjektet var oppfylt ble det kjørt en akseptansetest (demo) på det ferdige produktet. Både at det virket, og det som ligger bak ble vist fram. Kunden var enig i at dette var en bra løsning på oppgaven til prosjektet, og at hans mål var oppfylt. Kunden har også vært med under hele prosjektprosessen, og har i så måte kunne være med å styre det i den retningen som var ønskelig. Dette har nok også vært med på å gjøre at man kom fram til et tilfredstillende resultat.

Det som også må nevnes er at for at produktet skal kunne taes i bruk i forbindelse med framtidige prosjekter i EDB ASA, må også noen av de videre utvidelsene som er skissert opp i implementasjonsdokumentet [IMPL] implementeres. For eksempel må det lages regler for ”deployment descriptor” for å kunne automatisk legge ut EJB på en server. Det er dokumentert hvordan og hvor lang tid disse utvidelsene vil ta, slik at EDB ASA forholdsvis enkelt skal kunne foreta utvidelsene.

5.3 Gruppens mål

I forkant av prosjektet satte gruppen seg sammen og ble enige om en del mål og ambisjoner som de hadde med prosjektet. Viktige momenter for gruppen var at medlemmene ønsket å tilegne seg ny kunnskap om teknologier som kunne brukes til å løse prosjektet. Også det å skulle jobbe gjennom et større prosjekt over lengre tid ble sett på som en utfordring som skulle bli interessant å komme i gang med. Dette ville innebære å lære seg en slik arbeidsprosess, samt å føle på kroppen hvordan det å jobbe tett sammen med andre. Det ble også fort klarlagt at gruppen ønsket å oppnå en god karakter, og dermed ble listen for lagt for hvor mye arbeid som skulle legges ned i prosjektet.

5.4 Videre utvikling av produktet

Systemet som er implementert oppfyller alle de kritiske og høye kravene fra kravspesifikasjonen. Her taes det utgangspunkt i at videre utvikling av produktet omhandler forbedringspotensialet som er beskrevet i implementasjonsdokumentet [IMP], kapittel 6, Videre utvidelser, og konstruksjonsdokumentet [KON], kapittel 11, Mulige utvidelser. Den videre utviklingen vi her legger opp til er det som prosjektgruppen anser som nødvendig for å ende opp med et produkt av industriell kvalitet. Med industriell kvalitet menes et produkt det vil lønne seg å bruke og som vil gi kunden fortrinn i forhold til konkurrenter som bruker produkter som allerede er på markedet.

I de følgende avsnittene angis tidsestimat for forskjellige forbedringer, og bakgrunnen for estimatene.

5.4.1 Mer spesifisering av regler og maler

Tidsestimat, forbedring regelformat: 50 timer
Tidsestimat, nye regler/maler: 100 timer

Det er to ting som må gjøres i forbindelse med regler og maler. Det ene er å forbedre formatet for reglene, og det andre er å utvikle regler og maler for flere arkitekturer man trenger. De formatendringene vi har foreslått i implementasjonsdokumentet vil ikke ta mye tid, men det er her tatt høyde for at nye behov melder seg ved utvikling av maler/regler for nye teknologier. Utvikling av regler/maler kan være vanskelig å komme i gang med, og kan ta mye tid de første gangene. Her er det tatt utgangspunkt i at det skal lages regler og maler for ”deployment descriptor” for EJB og klasser og grensesnitt for COM.

5.4.2 Verktøy for å spesifiserer regler og maler

Tidsestimat, regel- og malverktøy: 500 timer

I dag må regler skrives i XML for hånd, og malene må skrives som flate tekstfiler. Et verktøy som lager regler på det spesifiserte formatet i XML, basert på brukerens valg i en

GUI, må lages av utviklere som har svært god kjennskap til regelformatet. Malene må genereres i sammenheng med reglene, og med stor kontroll fra brukeren. Tidsestimatet baserer seg på at noen med like detaljert kunnskap som prosjektgruppen utvikler utvidelsen. Dette kan føre til at utvidelsen ikke kan implementeres med det første.
Utvidelsen står beskrevet i [KON] kapittel 11.2, Verktøy for oppretting/editering av regelfiler.

5.4.3 Støtte for flere versjoner av XMI

Tidsestimat, konverterteringsverktøy: 150 timer

Slik det er i dag, kan regelfilene måtte endres for å fungere sammen med nye versjoner av XMI. En løsning på dette problemet er et konverteringsverktøy for XMI, som konverterer forskjellige versjoner av XMI til en bestem versjon. Utviklingen av dette verktøyet kompliseres av at det dukker opp nye versjoner av XMI med jevne mellomrom, og at det kan være tunge strukturelle endringer mellom disse versjonene.
Utvidelsen står beskrevet i [KON] kapittel 11.1, Konverteringsverktøy for XMI.

5.4.4 Komplett tidsestimat

Tidsestimat, forbedring regelformat:	50 timer
Tidsestimat, nye regler/maler:	100 timer
Tidsestimat, regel- og malverktøy:	500 timer
Tidsestimat, konverterteringsverktøy:	150 timer
SUM	800 timer

Ut i fra vårt tidsestimat trengs det 550 timer for å utvide systemet vi har designet og implementert, til å bli et produkt av "industriell kvalitet".

6 Konklusjon

Totalt sett er gruppemedlemmene veldig fornøyd med faget kundestyrt prosjekt. Dette skyldes sannsynligvis i stor grad at vi har hatt en svært god sammensetning i gruppa. Gruppa har vært homogen på den måten at alle har vært engasjert og interessert i å medvirke til å få et godt resultat, noe som har gitt utslag i ikke alltid helt normale arbeidstider. Det virker som om vi har klart å beholde en stabil og jevn arbeidsmengde ved hjelp av god planlegging før hver fase. Ofte har det vært slik at innholdsfortegnelsen er blitt skapt og mye av ansvaret allerede fordelt før fasen er kommet skikkelig i gang. Dette har i alle fall gjort at det ikke har vært noen som har vært helt uten arbeid. Det ble brukt endel timer mer enn normert for faget, men dette skyldes som sagt forstudiet og konstruksjonen ble såpass omfattende og at gruppa ønsket å lage et produkt som virkelig fungerte. Ellers må det sies at vi har lært mye både om teknologier og om hvordan det er å jobbe sammen i et større prosjekt.

Gruppa føler at denne rapporten kan komme til å få betydelig nytteverdi for EDB ASA når det gjelder forenkling av kodegenereringen ved oppstart av nye prosjekter. Ikke bare fordi vi faktisk har laget en fungerende prototyp, men også fordi vi har lagt ned mye arbeid i forstudiet hvor eksisterende og relevant programvare er beskrevet i detalj, slik at hvis EDB ASA velger å kjøpe en ferdig programvarepakke i fremtiden, vil de slippe å starte med blanke ark, men vil kunne dra nytte av det grunnarbeidet som er gjort. Til slutt vil vi bare nok en gang takke hovedveileder Reidar Conradi, hjelpeveileder Rune Rystad og kunden Ketil Aasarød.

Kundestyrt prosjekt

Høst 2001

Glossar

Versjon 1.03

Gruppe 16

Endringslogg:

Dato	Endring	Versjon	Endret av
06.09.2001	Opprettelse av dokumentet	0.1	Martin
17.10.2001	Ord fra kravspesifikasjon lagt til	0.2	Atle
18.10.2001	Flere ord fra kravspesifikasjonen lagt til	0.3	Martin
19.10.2001	Ord fra forstudiet lagt til	0.4	Atle
02.11.2001	Ord fra konstruksjonsfasen lagt til	0.5	Atle
08.11.2001	Oppdatering av diverse	0.6	Magnus
11.11.2001	Oppdaterte resten av ordene	1.0	Atle
12.11.2001	Topptekst og bunntekst	1.01	Ole Kristian
13.11.2001	Endret skrivefeil	1.03	Magnus

Innholdsfortegnelse:

1 Innledning

1.1 Mål

Dette dokumentet skal fungere som en forklaring til alle fremmendord brukt i de ulike dokumentet, og på denne måte føre til øket forståelse for leseren.

1.2 Avgrensning

Dette dokumentet forklarer kun ord og eller uttrykk som er brukt i prosjektdokumentene.

1.3 Dokumentoversikt

Kapittel 2 inneholder en alfabetisk liste over alle ord og uttrykk som er tatt med.

2 Ordliste

Her vil du finne ordene forklart med våre egne ord samt eventuelle linker der det finnes mer dypt gående informasjon.

2.1 Ordforklaringer

Alfabetisk oversikt følger nedenfor.

2.1.1 ADA

Programmeringsspråk utviklet for det amerikanske Forsvarsdepartementet. Et svært omfattende språk nært beslektet med Pascal. Ada er oppkalt etter Augusta Ada Byron (1815-1852), datter av Lord Byron. Hun var en matematiker og kollega av Charles Babbage, datamaskinens oppfinner.

Mer informasjon på http://www.adahome.com/.

2.1.2 Aktivitetsdiagram

Et aktivitetsdiagram viser flyten fra aktivitet til aktivitet. En aktivitet er en abstrahering på høyt nivå, og vil gjerne inneholde flere enkelthandlinger.

2.1.3 Aktør

Utøver, handlende eller opptredende person.

2.1.4 ANSI – American National Standards Institute

Amerikansk standardiseringsorganisasjon, grunnlagt i 1918, som koordinerer den frivillige utviklingen av standarder i privat og offentlig sektor.

Mer informasjon på http://www.ansi.org/.

2.1.5 ANT

Apache Ant er et Java basert bygge verktøy. Det ligner på UNIX make.

Mer informasjon på http://jakarta.apache.org/ant/.

2.1.6 API – Application Programming Interface

En samling rutiner/funksjoner som programvareapplikasjoner bruker for å utføre ulike operasjoner. I for eksempel Windows er det en API som brukes av applikasjoner for å håndtere vinduer, menyer og ikoner. I stedet for at hver enkelt applikasjon skal ha sine egne rutiner for å håndtere dette, kan applikasjonene gjøre dette ved hjelp av kall til en felles API.

2.1.7 Arkitektur

Designet til et datasystem/dataprogram. Arkitekturen setter standarder for alle enheter som er tilkoblet og programvaren som kjører. Den er bygget ut i fra hvilken oppgave systemet/programmet skal utføre.

2.1.8 ASCII – American Standard Code of Information Interchange

En måte å kode tegnsett på en datamaskin på, hvor hvert enkelt tegn (f.eks. en bokstav) gis en verdi fra 0-255. Såkalt 7-bits ASCII benytter kun verdiene 0-127, altså kun de 7 første bitene i en byte. Den åttende biten er en paritetsbit som brukes til feilkontroll. Det finnes ulike variasjoner av ASCII. Den viktigste variasjonen er Extended ASCII, opprinnelig fra IBM. Extended ASCII benytter 8 i stedet for 7 databits, og bruker ASCII-verdier over 127 for å beskrive utenlandske tegn og spesialtegn. Tegnene fra 0-127 er identiske med standard 7-bits ASCII.

2.1.9 ASP – Active Server Pages

En spesifikasjon for Web-sider som inneholder enten Visual Basic eller JavaScript-kode og hvor webtjeneren genererer HTML-siden i det brukeren åpner dokumentet. En ASP-fil inneholder både skript og HTML-kode, og filene har endelsen .asp. Når webtjeneren ser filer med denne endelsen, vet den at filen skal prosesseres før den sendes.

2.1.10 BAT

Seriell fil. En fil bestående av instruksjoner som utføres i den rekkefølge de er skrevet i. Brukes særlig til å utføre rutineoperasjoner. I operativsystemet DOS er en BAT-fil en tekstfil som brukeren selv kan modifisere og som inneholder DOS-kommandoer. Kjennetegnes ved å ha filendelsen .BAT.

2.1.11 BMP – Bean Managed Persistence

Handler om persistens for entititetsbønner. Hvis du ønsker mer kontroll en i CMP, for eksempel gjøre ting mer effektivt enn den automatisk genererte koden, da må du bruke BMP. Der kan du skrive all database aksesslogikk som en del av din egen bønne. Ulempen er at du må forstå databasestrukturen og SQL og kode endel mer enn med CMP. Se CMP.

2.1.12 Brukergrensesnitt

Grensesnitt der skjermbildet inneholder vinduer og grafiske symboler. Brukeren kan, i tillegg til å skrive instruksjoner via tastaturet, også gi instruksjoner via en mus som styrer en markør på skjermen. Med markøren kan brukeren flytte eller markere objektene på skjermen, klikke på ikoner, starte opp programmer, osv.

2.1.13 Brukstilfelle

En transaksjon eller sekvens av handlinger utført av en bruker. Brukstilfellene blir studert for å bestemme hvilke objekter som er nødvendige for å utføre dem og hvordan de kommuniserer med andre objekter.

2.1.14 Brukstilfellediagram

Et brukstilfellediagram er en dynamisk beskrivelse av hva et system skal gjøre, men ikke hvordan. Det blir ofte brukt for å modellere kravene til et system.

2.1.15 C

Høynivå programmeringsspråk utviklet av Bell Laboratories. Inneholder de fleste moderne data- og kontrollstrukturer som man kan forvente å finne i et profesjonelt programmeringsspråk. C er ikke avhengig av å måtte kjøres på noen spesifikk maskinplattform, men er opprinnelig utviklet med tanke på UNIX.

2.1.16 C++

En objektorientert programmeringsspråk skapt av Bjarne Stroustrup. C++ bygger på C og har blitt veldig populært på grunn av måten det kombinerer klassisk C med objektorientert programmering.

2.1.17 C#

Objektorientert programmeringsspråk fra Microsoft Corporation som er basert C++ med elementer av Visual Basic og Java. For eksempel så tilbyr C# i likhet med Java automatisk søppelhåndtering, noe som C++ mangler. C# er utviklet parallelt med Microsoft .NET plattform og støtter forøvrig både XML og SOAP. Uttales C-sharp.

2.1.18 CMP – Container Managed Persistence

Handler om persistens for entititetsbønner. Dette er det enkleste opplegget for å oppnå persistens, hvor du bare ber en Container ta seg av alt. Den kan ta seg av forretningslogikk, spesifisere hvordan entitetsbønner mapper til felter i en database og slappe av mens bønna genererer SQL spørringer under kjøretid. Se BMP.

2.1.19 COM – Common Object Model

COM referer både en spesifikasjon og en implementasjon utviklet av Microsoft som gir et rammeverk for å integrere komponenter. Dette rammeverket støtter samarbeid og gjenbruk av distribuerte objekter som kommuniserer via COM. COM støtter også samarbeid av komponenter implementer i forskjellige språk vha binærkode. COM komponenter må kjøre i Windows miljø, og kan implementeres i mange ulike språk herunder VB, C++, Java, C# mfl.

Mer informasjon på http://www.microsoft.com/com/.

2.1.20 CORBA – Common Object Request Broker Architecture

Kommunikasjonsdelen av Object Management Architecture (OMA) fra the Object Management Group (OMG). CORBA er en standard som håndterer kommunikasjon/meldingsutveksling mellom objekter i et distribuert, multiplattform miljø. Ved hjelp av Corba kan altså objekter i ett program kommunisere med objekter i andre programmer, selv om de to programmene er skrevet i to forskjellige programmeringsspråk og kjører på ulike maskinvare- og operativsystemplattformer. Corba 1.1 ble introdusert i 1991 og versjon 2.0 kom i desember 1994.

2.1.21 CRC – Class Responsibility Collaboration

Et metode for objektorientert design som bruker vanlige 3x5 indeks kort. Utviklet av Ward Cunningham ved Textronix. Et kort for hver klasse inneholder ansvar (kunnskap og tjenester) og samarbeid (interaksjon med andre objekter). Kortene gir en uformell intuitiv måte for gruppemedlemmer å arbeide sammen om et objektorientert design.

2.1.22 CSS – Cascading Style Sheets

Cascading Style Sheets. En HTML-spesifikasjon som gir webutviklere og webbrukere mer kontroll over hvordan Web-sider vises i webleseren. Spesifikasjonen ble utviklet av W3C - World Wide Web Consortium. Med CSS kan både brukere og webutviklere lage stilark (”style sheets”) som beskriver hvordan siden skal se ut, med for eksempel typografisk informasjon (skrifttyper, størrelser, osv.) og hvordan de ulike elementene på siden skal se ut. Dette stilarket kan så knyttes til en hvilken som helst webside. CSS versjon 2 inneholder også støtte for XML, nedlastbare skrifttyper, osv.

Mer informasjon på http://www.w3.org/Style/CSS/.

2.1.23 CVS – Concurrent Versions System

Et system for versjonskontroll på UNIX. Opprinnelig utviklet som en serie med shellskript på midten av 1980-tallet av Dick Grune i 1986. Det ble omgjort til et C program i 1989 av Brian Berliner med litt hjelp fra Jeff Polk. CVS tar hånd om

endringene mellom en kildekodeversjon og en annen og lagrer alle endringene i en fil. CVS støtter samarbeid mellom hver enkelt av programmererne ved å merge filene.

Mer informasjon på http://www.cvshome.com/.

2.1.24 DCOM – Distributed COM

Tidligere kjent som Network OLE. Microsoft sin teknologi for distribuerte objekter. En versjon av COM som tillater deling av binære objekter via nettverk. DCOM har fulgt med Windows pakken siden NT4 og er Microsoft sitt svar på Corba.

2.1.25 DBMS – Data Base Management System

En samling programmer som brukes til å administrere og vedlikeholde en database. Det finnes flere arkitekturer som et DBMS kan legge seg opp til. Eks er relasjonsdatabaser, nettverksdatabaser, hierarkiske databaser, inverterte listedatabaser og objektorienterte databaser.

2.1.26 DDL – Data Description Language

Et språk brukt for å definere data og deres relasjoner til andre data. Det blir ofte brukt for å lage datastrukturer i en database. Store DBMS bruker SQL.

2.1.27 Deployment

Er et ord får å legge kode og/eller data ut på en server. For eksempel en webserver.

2.1.28 Deployment Descriptor

Er en samling data som beskriver hvordan en kjørbar kode skal legges ut på en server. F.eks. i sammenheng med EJB er beskrivelsen lagret i en XML fil, som så brukes i prosessen med å legge koden ut på serveren.

2.1.29 DFD – Data Flyt Diagram

Et modelleringsspråk for å angi flyt i et system.

2.1.30 DLL – Dynamic Linked Library

En implementeringsmåte for biblioteksrutiner under operativsystemene OS/2 og Windows. Biblioteksrutinene ligger lagret som filer med endelsen .DLL, og Windows/OS/2 laster inn DLL-filen når programmet som kjøres trenger den.

2.1.31 DOM – Document Object Model

Et vanlig programmerings grensesnitt (API) for aksessering av HTML og XML dokumenter fra en webleser. Det ble utviklet for å formalisere dynamisk HTML, som tillater animasjon, interaksjon og dynamisk oppdatering av websider. DOM tilbyr en språk- og platformnøytral objektmodell for websider, men fordi den er såpass generell kan den brukes av alle programmer som aksesserer dokumenter. I 1998 lanserte W3C DOM Level 1.

2.1.32 DTD – Document Type Definition

Et språk som beskriver innholdet til et SGML dokument. DTD blir brukt sammen med XML og DTD definisjonene kan integreres i XML dokumentet eller i en separat fil.

2.1.33 EAR – J2EE Application

EAR er en type fil som inneholder en applikasjon. For eksempel, i fasen for utviklingen av komponenter, lagrer en utvikler av Enterprise Java Beans bønnene i en JAR-fil. I sammensetningen av hele applikasjonen kombineres disse filene og setter de sammen til en J2EE applikasjon og lagrer det i en EAR-fil.

2.1.34 EJB – Enterprise JavaBeans

Er en serverbasert komponent modell fra SUN som tilbyr portabilitet over applikasjons-servere, samt tjenester som transaksjoner, sikkerhet osv. til komponentene. EJB leverer et konsistent grensesnitt til alle applikasjoner uansett servertype. Alle komponentene må implementeres i Java.

Mer informasjon på http://java.sun.com/products/ejb/.

2.1.35 Enitetsklasse

En entitetsklasse er en objektsrepresentasjon av en entitet, det vil si et dataobjekt i systemet. Denne inneholder informasjon og har ingen annen logikk en det som har med endringer av denne entiteten.

2.1.36 ER – Entity Relationship

En ER modell er n konseptuel datamodell som viser verden som entiteter og deres relasjoner. En viktig komponent av modellen er Entity-Relationship diagrammet som brukes til å visualisere dataobjekter

2.1.37 EXE

En kjørbar fil med filendelsen .EXE. Disse filene kalles også programfiler. I DOS kan du starte opp en EXE-fil ved å taste navnet på filen (uten .EXE-endelsen). I Windows starter du opp filen ved å dobbeltklikke på filnavnet (i Windows Utforsker eller Filbehandling.

2.1.38 Feilhåndtering

Feilhåndtering er å ta seg av feil som oppstår på en fornuftig måte. F.eks. utføre en handlig etter hvilken feil som oppstår.

2.1.39 Grenseklasse

En grenseklasse er en klasse som utgjør en grense i systemet. En slik grense kan være ut mot brukeren eller mot en pakke i systemet.

2.1.40 Grensesnitt

Programvaregrensesnitt definerer språkene, kodene og meldingene som programmer kan bruke til å kommunisere med hverandre. Et eksempel er SMTP og LU 6.2 kommunikasjonsprotokollene.

2.1.41 GUI – Graphical User Interface

Se Brukergrensesnitt.

2.1.42 GUID – Global Unique Identifier

Et unikt nummer som blir brukt for å identifisere et COM objekt. Dette blir laget ved å addere tid og klokkeslett med enhetens interne serienummer.

2.1.43 HP – Hewlett Packard

Vanlig brukt forkortelse for Hewlett Packard, USAs nest største dataselskap og leverandør av datamaskiner, periferiutstyr og måleutstyr. HP ble grunnlagt i 1939 av William Hewlett og David Packard i en garasje i California. Deres første produkt var en lydoscillator for måling av lyd. HPs første store kunde var Walt Disney Studios som kjøpte åtte oscillatorer for å utvikle og teste et nytt lydsystem til filmen Fantasia.

Mer informasjon på http://www.hp.com/.

2.1.44 HTML – Hyper Text Markup Language

Navnet på språket som benyttes for å lage dokumenter på World Wide Web. Det aller meste vi ser av Web-sider når vi surfer på Internett er laget i HTML. HTML er et

standardisert språk basert på SGML, og HTML-baserte dokumenter kan opprettes og redigeres med en rekke ulike verktøy. Se SGML.

Mer informasjon på http://www.w3.org/MarkUp/.

2.1.45 HTTP – Hyper Text Transport Protocol

Den underliggende protokollen som brukes av World Wide Web. HTTP definerer hvordan meldinger formateres og overføres, og hvordan webtjenere og weblesere skal svare på ulike kommandoer.

2.1.46 IBM – International Business Machines

Verdens største dataselskap, produsent av datamaskiner. Lanserte i 1981 sin IBM PC, som er den maskinen dagens PC-standard fortsatt bygger på. Derav begrepet IBM-kompatibel.

Mer informasjon på http://www.ibm.com/.

2.1.47 IDL – Interface Definition Language

Et språk brukt for å beskrive grensesnittet til en metode. For eksempel er alle objekter i CORBA modellen definert med en IDL som beskriver tjenestene objektet tilbyr og hvordan data kan utveksles.

2.1.48 Interface

Se Grensesnitt.

2.1.49 Intern mal

En intern mal er en mal som er definert i reglene for en arkitektur. Denne malen er definert der fordi den skal brukes på mange forskjellige maler. Man unngår ved denne sentraliseringen dobbeltlagring og letter oppdateringen av nettopp slike maler.

2.1.50 ISO – International Organisation for Standardization.

Internasjonal Standardiseringsorganisasjon, som har hovedkvarter i Geneve, Sveits. Består av nasjonale standardiseringsorganer fra rundt 130 land. ISO ble grunnlagt i 1947. Forkortelsen ISO står ikke for ”International Standards Organization” slik mange tror (navnet på organisasjonen er ”International Organisation for Standardization”, forkortelsen burde da vært IOS), men er et ord avledet av det greske ordet ”isos”, som betyr ”lik”. ISO er mest kjent for bla. å ha definert standarder for kvalitetssikring og datakommunikasjon, bla. den såkalte OSI-modellen, men har også definert en rekke

andre standarder. Vi nevner bla. ISO-standarden for fotografisk film (ISO film speed code), standardisering av telefon og bankkort, papirstørrelser, skruegjenger osv.

2.1.51 J2EE – Java 2 Enterprise Edition

En platform fra SUN for å bygge webbaserte serverapplikasjoner. J2EE tar seg av alle tjenester mellom brukerens leser og serverens databaser og informasjonssystemer. J2EE støtter JDBC for kommunikasjon med databaser, JNDI for kataloger, JTA for transaksjoner, JMS for meldingstjenester, JavaMail for emailsystemer og JavaIDL for CORBA kommunikasjon. I desember 1999 kom versjon 1.2 og var den første formelle beskrivelsen.

2.1.52 JAR – Java Archive

Et filformat brukt for å distribuere en Java applikasjon. Det inneholder alle resursene som kreves for å installere og kjøre Java programmet i en enkelt komprimert fil. JAR blir også brukt for å distribuere javabønner.

2.1.53 Java

Et C++-lignende objektorientert programmeringsspråk utviklet av Sun Microsystems. Java-kildekode kompileres til et format som kalles bytekode, som deretter kan kjøres av en Java-interpreter. Java-interpretere finnes for de fleste operativsystemer, inkludert Windows, Unix og Mac, og kompilert Java-kode kan derfor kjøre på de fleste datamaskiner. Java inneholder en rekke funksjoner som gjør at språket egner seg spesielt godt for WWW. Blant annet kan nyere Web-lesere laste ned små såkalte Java-applets og kjøre disse inne i webleseren, i teorien uavhengig av maskinvareplattform.

Mer informasjon på http://www.java.sun.com/.

2.1.54 Javax

Er en pakke som inneholder diverse javakomponenter.

2.1.55 JDK – Java Development Kit

Et sett av utviklingsverktøy fra SUN. Det inkluderer Java Viritual Machine (JVM), compiler, debugger og andre verktøyer for utvikling av Java applets og applikasjoner. Hver ny versjon av JDK tilbyr nye egenskaper og forbedringer til språket. Når Java applikasjoner blir utviklet under en ny versjon må også tolkeren oppdateres til samme versjon.

2.1.56 JSP – Java Server Pages

En fri spesifikasjon for å bruke Java Servlet API til å generere websider. JSP-spesifikasjonen ble skrevet av ledende industri som en del av utviklingen av Java. JSP er en HTML side med Java kode innebygget. Java koden utføres av webserveren eller applikasjonsserveren når noen henter fram siden. JSP kan også kalle EJBer for videre prosessering. JSP er SUN sitt svar på Microsoft ASP (Active Server Pages).

2.1.57 JTA - Java Transaction API

JTA spesifiserer et standart Java interface mellom en transaksjonsmanager og partene involvert i et distribuert transaksjonssystem: resurs manager, applikasjonsserveren og applikasjonene som utfører transaksjonene.

JTA spesifikasjonen ble utviklet av Sun i samarbeid med ledende industri innenfor transaksjonsprosesser og databasesystemer.

2.1.58 JTS – Java Transaction Service

JTS spesifiserer implementasjonen av en transaksjonsmanager som støtter Java Transaction API(JTA).

2.1.59 JVM – Java Virtual Machine

JVM. Programmer skrevet i Java-språket kjører i en såkalt JVM, som opptrer ovenfor Java-programmet som om det var en selvstendig datamaskin. Web-lesere som f.eks. Netscape og Microsoft Internet Explorer inneholder en JVM, som kjører og tolker Java-programmene. Java-programmer som kjører inne i en JVM har ingen adgang til vertsdatamaskinens operativsystem, noe som betyr økt sikkerhet. JVMer eksisterer for flere ulike maskinvareplattformer, dette betyr at én og samme Java-applikasjon uten modifikasjoner kan kjøre på flere ulike maskiner, uavhengig av maskinvare og operativsystem.

2.1.60 Kall

Se Metodekall.

2.1.61 Katalogstruktur

En katalogstruktur vil si hvordan man organiserer kataloger på et lagringsmedium. En struktur består ofte av en eller flere rotkataloger med underkataloger.

2.1.62 Klasse

En klasse er det som bli et objekt ved kjøring av javakode. Som regel lagrer man én klasse per fil, men der er også mulig med flere per fil.

2.1.63 Klassediagram

Et klassediagram viser klassene og grensesnittene et system består av, samt relasjonene og samspillet mellom dem.

2.1.64 Komponentdiagram

Et komponentdiagram viser et relasjoner mellom programvarekomponenter. Komponenter i UML kan for eksempel være filer, grensesnitt, klasser og lignende som samles i pakker. Pakker kan også nøstes innenfor andre pakker.

2.1.65 Komponentmal

Se maler.

2.1.66 Komponentregler

Se regler.

2.1.67 Kontrollklasse

En kontrollklasse er en klasse som kontrollerer at alt går riktig for seg. Det være seg validering av data eller kjøring av programmet.

2.1.68 Maler

Maler er tekstfiler som danner grunnlaget for kodegenereringen. Malene ligner mye på den ferdige koden, men data fra modellen skal plasseres inn i malene før de blir til ferdig kode.

2.1.69 Marshalling

Oversettermekanisme som oversetter beskjedene som skal sendes av gårde fra en maskin til et forståelig språk for en annen maskin.

2.1.70 MDA – Model Driven Architecture

Model Driven Architecture (MDA) "provides an open, vendor-neutral approach to the challenge of interoperability, building upon and leveraging the value of OMG's established modeling standards: Unified Modeling Language (UML); Meta-Object Facility (MOF); and Common Warehouse Meta-model (CWM). Platform-independent Application descriptions built using these modeling standards can be realized using any major open or proprietary platform, including CORBA, Java, .NET, XMI/XML, and Web-Based platforms."

2.1.71 Metodekall

Et metodekall er når f.eks. et objekt kaller(starter) en metode for å få utfør en handling.

2.1.72 MS – Microsoft

Verdens største produsent av programvare for PCer. Grunnlagt i 1975 av Paul Allen og Bill Gates, to college-studenter som sammen skrev den første Basic-interpreteren for Intels 8080-mikroprosessor. Microsoft står blant annet bak operativsystemene MS-DOS og Microsoft Windows, Windows 95 og Windows NT.

Mer informasjon på http://www.microsoft.com/.

2.1.73 MTS – Microsoft Transaction Server

En transaksjonsserver fra Microsoft for Windows som støtter transaksjoner på LAN, internet og intranet. Den blir brukt i mellomlaget mellom klient og databaseserver. MTS støtter tofaselåsing som er inkludert i Distributed Transaction Coordinator (DTC). MTS er vert for ActiveX Server komponenter. En komponent er skrevet for en bruker, men MTS skalerer dette opp til en prosess for mange brukere. MTS er ikke en separat komponent, men en del av operativsystemet.

2.1.74 Moduler

En modul er en del av et system som utfører en oppgave.

2.1.75 MVC – Model View Controller

MVC er en mye brukt og robust arkitektur for GUIer. MVC paradigmet er en måt å dele opp en applikasjon, eller deler av en applikasjon, i tre deler: modell, det man ser og kontroller.

2.1.76 .NET

Mer informasjon på http://www.microsoft.com/net/.

2.1.77 Objekt

Et objekt er noe som blir opprettet ved kjøring av kode. Et objekt virker slik at det kan ha interne metoder og variabler, og kan samhandle med andre objekter for å få utført ønsket handling.

2.1.78 Objektdiagram

Et objektdiagram er en statisk diagramtype som viser objekter og deres relasjoner på et gitt tidspunkt. Objekter vil gjerne være instanser av klasser.

2.1.79 OOAD – Objekt Orientert Analyse og Design

Analyse er dreier seg om å utforske et problem ved å se på det som en gruppe av samarbeidende objekter. Et objekt er definert av sin klasse, dataelementer og oppførsel. Design handler om å transformere en objektorientert modell til spesifikasjoner som må til for å lage systemet.

2.1.80 OQL – Object Query Language

Et spørrespråk som støtter komplekse datatyper (multimedia, dokumenter etc.) som blir lagret som objekter. OQL er definert av ODMG og er en utvidelse av SQL-92 spørrespråket. Standard SQL kan fortsatt benyttes og OQL serveren prosesserer forespørslene.

2.1.81 Pakke

En pakke er en samling av f.eks. kode som passer sammen. En pakke kan også inneholde data.

2.1.82 Parsing

Parsing er å gjøre noe om til et annet format. I systemet vårt bli XML parset til et tre som er representert i et javaobjekt.

2.1.83 PDF – Portable Document Format

Filformat for dokumenter som kan inneholde både tekst og grafikk, og som kan hentes frem uavhengig av maskinplattform. Utviklet av Adobe Systems. For å lese PDF-filer trenger man Adobe Acrobat Reader som kan hentes gratis fra Internett.

2.1.84 RAM – Random Access Memory

Maskinens internminne. Navnet kommer av at prosessoren i datamaskinen kan lese programinstruksjoner eller data fra vilkårlige posisjoner i dette minnet, og altså ikke behøver å lese instruksjonene/dataene sekvensielt. Programmer eller data leses fra en disk(ett) eller annet permanent lagringsmedium og inn i maskinens internminne. Prosessoren leser så instruksjonene fra internminnet og utfører disse. Når maskinen slås av, forsvinner informasjonen som ligger lagret i internminnet (RAM).

2.1.85 Reengineering

Reengineering vil si å ta eksisterende kode og arbeide videre med den for å lage et forbedret eller nytt system. Ofte er det ønskelig å ta eksisterende kode inn i et modelleringsverktøy, endre modellen der og så generere koden på nytt.

2.1.86 Regler

Reglene bestemmer hvordan generatoren skal benytte maler og modell for å danne den ferdige koden. Reglene er beskrevet i konstruksjonen, kapittel 4.

2.1.87 RPC – Remote Procedure Call

Et programmeringsgrensesnitt som tillater et program å bruke tjenestene til et annet program på en annen maskin. Det kallende programmet sender et kall til programmet på den andre maskinen, der kallet blir kjørt og resultatene blir sendt tilbake til det kallende programmet.

2.1.88 RUP – Rational Unified Process

Programvare fra Rational Software Corporation som gir rettledning, maler og eksempler til gruppemedlemmene i utviklingsprosessen. RUP har god støtte for UML.

Mer informasjon på http://www.rational.com/products/rup/.

2.1.89 Samarbeidsdiagram

Et samarbeidsdiagram viser organiseringen av objekter og flyten av meldinger i en interaksjon.

2.1.90 SAX – Simple API to XML

En hendelsorientert programmeringsgrensesnitt mellom XML parseren og en XML applikasjon. Et objektorientert grensesnitt blir tilbudt av DOM.

2.1.91 Scenario

Et scenario er en beskrivelse av et hendelsesforløp.

2.1.92 Sekvensdiagram

Et sekvensdiagram viser tidsrekkefølgen for meldinger mellom objekter, og dermed hvordan de samarbeider.

2.1.93 SGML - Standard Generalized Markup Language

Standard for lagring av dokumenter på strukturert form i en database. Utarbeidet av ISO og beskrevet i ISO 8879. Se HTML.

2.1.94 SOAP – Simple Object Access Protocol

En meldingsbasert protokoll basert på XML for å aksessere tjenester på web initiert av Microsoft, IBM og andre. Protokollen bruker XML syntaks for å sende tekst kommandoer over Internet ved hjelp av HTTP.

2.1.95 SQL - Structured Query Language

Standard språk for å søke/hente informasjon fra en database, gjerne lagret sentralt på en databasetjener. Den originale versjonen ble utviklet av IBM i 1974-75, og SQL ble først introdusert kommersielt i 1979 av Oracle.

2.1.96 Stub

En liten programvarerutine som tilbyr en funksjon. Stubs blir for eksempel brukt på klientmaskiner og motsvarende på servermaskinen. Disse er i stand til å forstå en felles protokoll, Remote Procedure Call (RPC) eller lignende.

2.1.97 SUN

SUN Microsystems er en stor produsent av UNIX baserte arbeidsstasjoner og servere. Selskapet ble grunnlagt i 1982 av Andreas Bectolsheim, Vinod Khosla og Scott McNeally. Deres første datamaskiner SUN-1 og SUN-2 var en stor suksess i universitetsmiljøet.

Mer informasjon på http://www.sun.com/.

2.1.98 Swing

En javapakke for å utvikle brukergrensesnitt. Swing gir muligheter for menyer, dialogbokser, tekstfelt etc. Swing er skrevet i Java og er derfor platformuavhengig i motsetning til Java Abstract Window Toolkit (AWT) som tilbyr platformspesifik kode. Swing har også et mer avansert grensesnitt som for eksempel tilbyr metoder for å forandre bilder på knapper. Swing er inkludert i Java Foundation Classes (JFC) som følger med i Java Development Kit (JDK).

2.1.99 Template

Se maler.

2.1.100 Tilstandsdiagram

Et tilstandsdiagram viser en tilstandsmaskin og legger vekt på hva som skal til for å gå fra en tilstand til en annen.

2.1.101 Traversering

Traversering vil si å søke seg gjennom et tre. Det medfører at man besøker node for node i treet.

2.1.102 UML - Unified Modelling Language

Et språk som brukes for å spesifisere, visualisere, konstruere og dokumentere programvaresystemer. UML skal forenkle programdesignprosessen. Språket ble utviklet av Ivar Jacobson, Grady Booch og Jim Rumbaugh hos Rational Software, og er nå en akseptert standard fra OMG (Object Management Group).

2.1.103 UNIX

Flerbruker operativsystem utviklet av Dennis Ritchie og Ken Thompson ved AT&T Bell Laboratories fra 1969 til 1973. UNIX ble opprinnelig utviklet for bruk på minidatamaskiner, og har blitt videreutviklet opp gjennom årene, og finnes nå i en rekke ulike varianter fra ulike leverandører. Noen av de mest kjente variantene er AIX (fra IBM), BSD UNIX, System V, A/UX, SunOS og HP-UX (fra Hewlett-Packard). Også Linux er basert på Unix. Ettersom UNIX er skrevet i programmeringsspråket C, er operativsystemet svært lett å flytte over på nye maskinvareplattformer.

2.1.104 Use Case

Se Brukstilfelle.

2.1.105 Utplasseringsdiagram

Et utplasseringsdiagram viser konfigurasjonen av eksekverbare prosesseringsnoder og komponentene som inngår.

2.1.106 Utvikler

En utvikler er en som utvikler systemer. Det kan f.eks. være javaprogrammer, hjemmesider, COM komponenter m.m.

2.1.107 Validering

Validering vil si å godkjenne noe ut fra predefinerte regler.

2.1.108 VB – Visual Basic

Høynivå programmeringsspråk for Windows, laget av Microsoft. V.B. er svært brukervennlig, til tross for at det har svært avanserte muligheter. Dette har gjort det til et meget populært programmeringsspråk for Windows. Mange av Microsofts applikasjoner (f.eks. Word) har et innebygd språk (Visual Basic for Applications) som er en forenklet utgave av VB.

2.1.109 Verifisering

Verifisering er å kunne bevises at noe er sant. For eksempel kan det være viktig å verifisere at en melding virkelig kommer fra den personen det står den kommer fra.

2.1.110 W3C – World Wide Web Consortium

Organisasjon grunnlagt i oktober 1994 med det formål å utvikle felles protokoller og standarder for World Wide Web, samt promotere utviklingen av World Wide Web. W3C er et internasjonalt industrikonsortium som ledes av MIT/LCS (Massachusetts Institute of Technology/Laboratory for Computer Science) i USA, Institut National de Recherche en Informatique et en Automatique (INRIA) i Europa og Keio University i Japan. Medlemsskap i W3C er åpent for alle organisasjoner/bedrifter som undertegner en medlemsskapsavtale.

Mer informasjon på http://www.w3.org/.

2.1.111 WAR – Web Application Archive

Man legger webkomponentene til en J2EE applikasjon i en pakke som kalles WAR. Et WAR inneholder som oftest annet i tillegg til webkomponenter:

- ?? Server side nytteklasser(database bønner)
- ?? Statiske webresurser(HTML, bilder, lyd)
- ?? Klinet side klasser(Appleter og nytteklasser)

2.1.112 Windows

Det mest utbredte operativsystemet for PCer. Utviklet av Microsoft Corporation. Windows har grafisk brukergrensesnitt, og finnes i flere versjoner der de mest brukte i dag er Windows 95/98/Me for eller Windows NT4/2000/Xp.

Mer informasjon på http://www.microsoft.com/windows/.

2.1.113 WWW – World Wide Web

Et system av nettverkstjenere tilgjengelig på Internett, og som støtter dokumenter formatert på en spesiell måte som ofte inkluderer grafikk, tekst, lyd og animasjon, samt linker til andre dokumenter. Dokumentene er formatert i et språk som kalles HTML. Ved

å klikke på linker på en webside, kan du komme videre til andre dokumenter på samme nettverkstjener, eller videre til en webside ett helt annet sted i verden. WWW ble opprinnelig utviklet ved CERN i Sveits. WWW er det mange tenker på når det snakkes om Internett, selv om Internett også er andre ting, som for eksempel elektronisk post, FTP, osv.

2.1.114 XMI - XML Metadata Interchange

Er en standard for representasjon av objektorienterte modeller i XML. Se XML.

2.1.115 XML - eXtensible Markup Language

Et språk utviklet for å gjøre det mulig å implementere SGML på World Wide Web. I likhet med HTML er språket konstruert spesielt for å lage Web-dokumenter, men språket er i motsetning til HTML ikke et fastlåst språk men utvidbart (derav ”extensible”). XML har mer avanserte muligheter enn HTML, blant annet mulighet for webutvikleren til å kunne lage sine egne ”tags”, og dermed kunne lage dokumenter med sofistikerte underliggende datastrukturer, som kan behandles av en hvilken som helst sluttbrukerapplikasjon, eller for eksempel en søkemotor. Mens man med HTML er fastlåst til en dokumenttype, kan man med XML selv definere sine egne dokumenttyper. XML ble utviklet av XML Working Group (opprinnelig kjent som SGML Editorial Review Board) i 1996.

Quick Project Organizer

Oppdragsgiver: EDB ASA

Prosjektgruppe 16:

Atle Nes

Magnus Kinn Solbjørg

Ole Kristian Hoel

Odd Christian Landmark

Martin Sleire Vatne

IDI,NTNU høsten 2001

SIF8080 Kundestyrt prosjekt

Målsetting med presentasjonen

- Gi en kjapp introduksjon til de oppgavene systemet løser
- Gi en introduksjon til de konsept som systemet er bygget etter
- Gi en ”flying start” til bruk av systemet

2

Dagens agenda

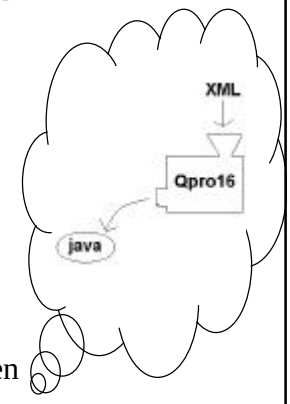
- Oppgaven, dens utvikling og løsning
- Presentasjon av løsningen
- Demonstrasjon av løsningen
- Oppsummering
- Spørsmål og svar

3

Den opprinnelige oppgaven

Et "light" verktøy for generering av EJB-baserte systemer.

Inndata til verktøyet er en beskrivelse av en datamodel eller UMLmodel og andre nødvendige parametre i XML.



Utdata fra verktøyet skal som minimum være:

- sql create-script for opprettelse databasen
- et ferdig generert EJB prosjekt

4

Jammen, dette er jo løst før…

En markedsundersøkelse fant 21 forskjellige eksisterende løsninger på oppgaven…..

- Rational Rose 2001
- Together Control Center 5
- StructureBuilder
- SoftModeler 3
- Select

+++

5

Nye kriterier fra kunden

Kodegeneratoren må være:

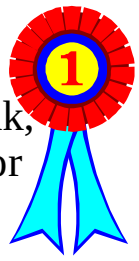
- generell og kunne generere for mange ulike språk
- mest mulig uavhengig fra leverandør
- enkel å oppdatere til å støtte nye språk

6

Vår løsning på de nye kriteriene

Vår **egen løsning** skal generere kode fra et sett av brukerdefinerte regler og maler.

- ✍Er generell, støtter alle språk som brukeren har laget regler og maler for
- ✍Er uavhengig av leverandør
- ✍Er enkel å oppdatere til å støtte nye språk, brukeren lager kun reglene og malene for språket

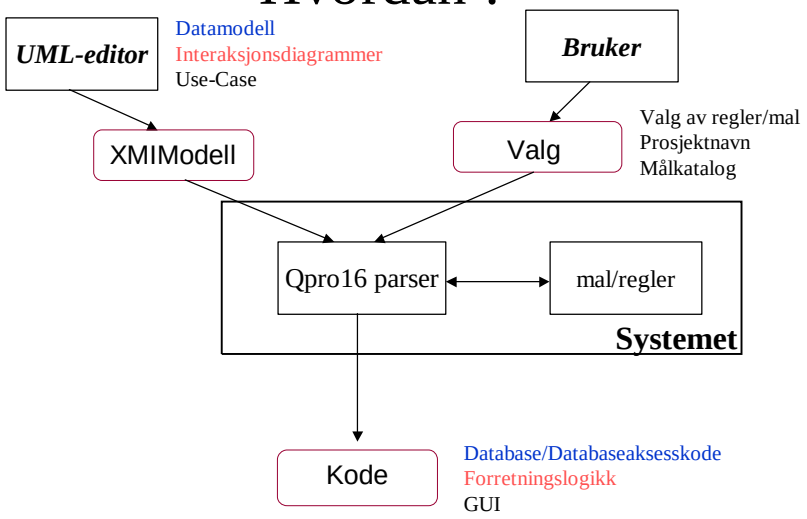


7

- ✍Oppgaven, dens utvikling og løsning
- ✍**Presentasjon av løsningen**
- Demonstrasjon av løsningen
- Oppsummering
- Spørsmål og svar

8

Hvordan ?



9

Maler, hva er det?

- **Maler:** Skjellett for den ferdige koden, inneholder variabler der informasjon fra modellen skal inn. Lagres som tekst filer.

```
public class %CLASSNAME%{
```

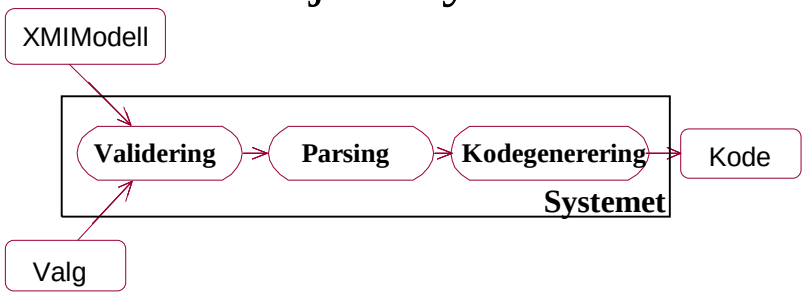
10

Regler hva er det?

- **Regler:** Forteller hva som er relevant informasjon i modellen, og hvordan denne skal behandles før de kombineres med malen. Lagres som XML filer.
- En regelfil kan bruke ***mange*** malfiler.

11

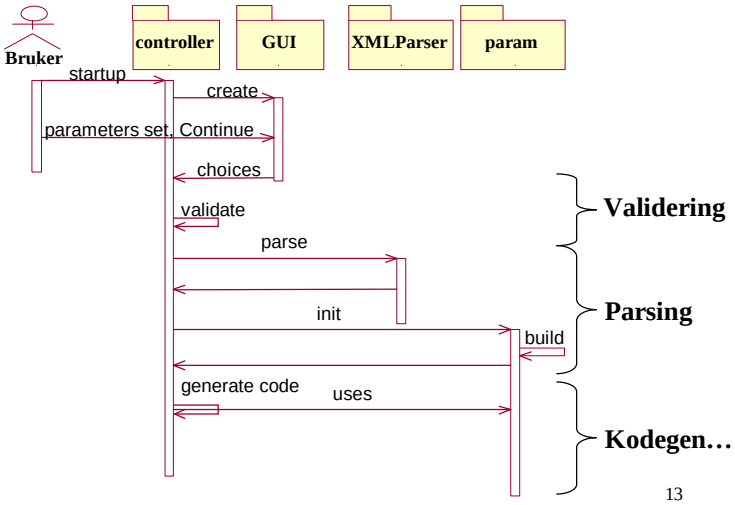
Hva skjer i systemet?



- Validering, finnes filene, er de av rett format o.l
- Parsing, inndata, regler og maler parses til trestrukturer og/eller objektrepresentasjoner.
- Kodegenerering, koden bygges

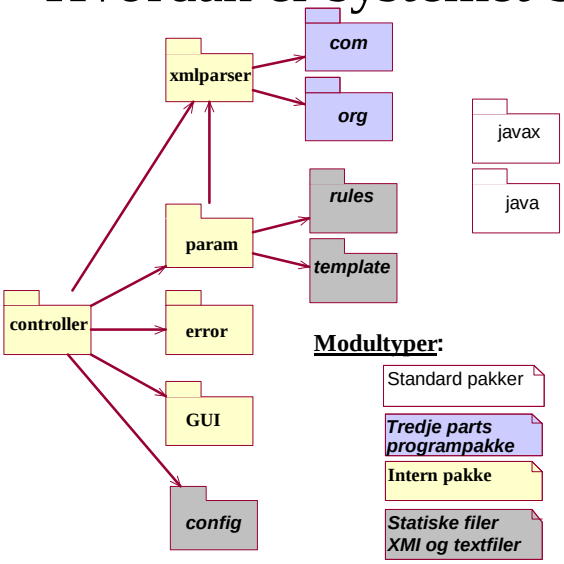
12

Hva skjer i systemet? (II)



13

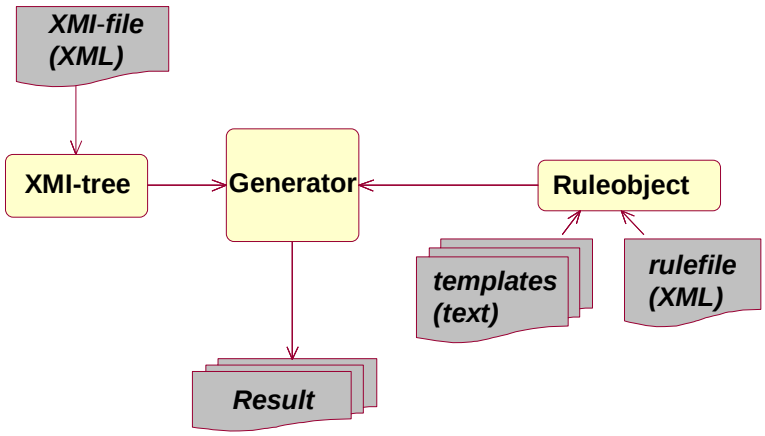
Hvordan er systemet bygd opp?



- Delt i 4 forskjellige typer moduler
- Består både av statiske filer og kode.

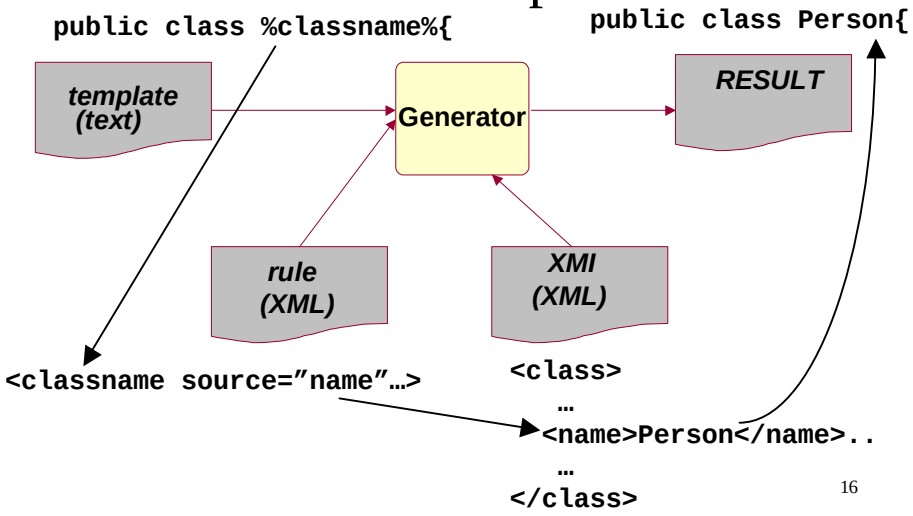
14

Hvordan brukes reglene og malene i kodegenereringen??



15

Regler, maler og inndata, et eksempel



16

Hvordan fungerer generatoren?

- Dybde først søk i modellen, der det sjekkes om det finnes en mal for nodene
- Når en node med tilhørende mal finnes hentes den (de) aktuelle malen(e) ut og det løpes igjennom malen
- Hver gang det finnes en variabel brukes tilhørende regel til å hente og behandle informasjonen som skal settes inn

17

Hvordan fungerer generatoren? (II)

```
generate(XMItre-rot rot)
  For alle barneNoder bnode til rot
      Hvis bnode er triggernode i regel-fil
            templateGenerate(bnode,templatefil)
      Ellers
            generate(bnode)

templateGenerate(subtre,templatefil)
  For alle Elementer tempEl (%...%) i templatefil
      Finn tilsvarende element e i rules-fil
      Finn node n i subtre hvor n.navn = e.source
      Hvis regelen sier bytt
            Legg n.value på plassen til tempEl
      Hvis regelen peker på en mal
            templateGenerate(n,mal)
      Hvis regelen ikke finnes
            gå videre
```

18

✍Oppgaven, dens utvikling og løsning

✍Presentasjon av løsningen

✍**Demonstrasjon av løsningen**

- Oppsummering
- Spørsmål og svar

19

Demo

•Testkjøring av programmet

•Hva skjedde egentlig?

•Nye inndata og nye parametre

20

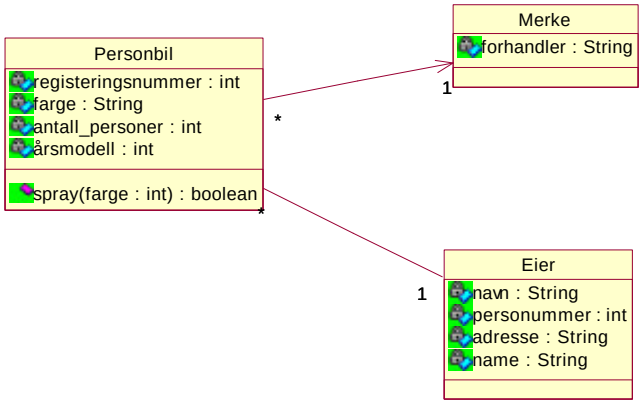
Testkjøring

21

Hva skjedde?

22

UML-modellen



23

XMI inndata

```
<XMI xmi.version = '1.0' timestamp = 'Tue Nov 06 16:38:32 2001' >
 <XMI.header>
  <XMI.documentation>
   <XMI.exporter>Unisys.JCR.2</XMI.exporter>
   <XMI.exporterVersion>1.3.2</XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name = 'UML' xmi.version = '1.3'/>
 </XMI.header>
<XMI.content>
<!-- ========== TESTCASE2    [Model] ========== -->
<Model_Management.Model xmi.id = 'G.0' >
  <Foundation.Core.ModelElement.name>TESTCASE2</Foundation.Core.ModelElement.name>
  <Foundation.Core.ModelElement.visibility xmi.value = "public"/>
```

24

Regelfilen

```
<TEMPLATES>
    <T1 trigger="Foundation.Core.Class" source="ejb.qpt" target="%CLASSNAME%EJB.java" />
    <T2 trigger="Foundation.Core.Class" source="pk.qpt" target="%CLASSNAME%PK.java" />
    <T3 trigger="Foundation.Core.Class" source="home.qpt" target="%CLASSNAME%Home.java" />
    <T4 trigger="Foundation.Core.Class" source="remote.qpt" target="%CLASSNAME%.java" />
</TEMPLATES>
```

trigger="**Foundation.Core.Class**"

25

XMI inndata

```
<Foundation.Core.GeneralizableElement.isLeaf xmi.value = "false"/>
<Foundation.Core.GeneralizableElement.isAbstract xmi.value = "false"/>
<Foundation.Core.Namespace.ownedElement>
  <!-- ==================== TESTCASE2::Personbil    [Class] ==================== -->
  <Foundation.Core.Class xmi.id = 'S.1' >
    <Foundation.Core.ModelElement.name>Personbil</Foundation.Core.ModelElement.name>
    <Foundation.Core.ModelElement.visibility xmi.value = "public"/>
    <Foundation.Core.ModelElement.isSpecification xmi.value = "false"/>
```

<Foundation.Core.Class xmi.id = 'S.1' >

26

Malfil

```
public abstract class %CLASSNAME%EJB implements javax.ejb.EntityBean {

   public javax.ejb.EntityContext EJB_Context;

            %ATTRIBUTE%

   /**
    * Constructor
    */
   public %CLASSNAME%EJB() {

   }
```

27

Malfil

public abstract class **%CLASSNAME%**EJB implements javax.ejb.EntityBean {

Regelfil

<**CLASSNAME** type="replace" link=""
source="**Foundation.Core.ModelElement.name**" mandatory="true" />

XMI inndata

<Foundation.Core.Class xmi.id = 'S.1'>
<**Foundation.Core.ModelElement.name**>Personbil</Foundation.Core.ModelElement.name>

Generert kode

public abstract class **Personbil**EJB implements javax.ejb.EntityBean {

28

Nye inndata og nye regler

29

Malfil

En liten historie:

%PERSON% dro til %BY% for å kjøpe %TING%,
men %TING%en ble brukt opp på turen hjem

30

Regelfilen

```
<RULEFILE>
   <VARIABLE_CHARACTER char="%" />
   <ID_ATTRIBUTE name="id" />
   <TEMPLATES>
      <t1 trigger="BYHISTORIE" source="byhistorie.qpt" target="%PERSON%_historie.txt" />
   </TEMPLATES>
   <INTERNAL_TEMPLATES />
   <RULES>
      <PERSON type="replace;filereplace" source="PERSON.navn" />
      <BY type="replace" source="BY.navn" />
      <TING type="replace" source="TING.navn" />
   </RULES>
</RULEFILE>
```

31

XML inndata

regelfil:
<t1 trigger="BYHISTORIE" source="byhistorie.qpt"
target="%PERSON%_historie.txt" />

```
<HISTORIER>
   <BYHISTORIE>
      <PERSON.navn>Kari</PERSON.navn>
      <BY.navn>Svinesund</BY.navn>
      <TING.navn>mat</TING.navn>
   </BYHISTORIE>
   <BYHISTORIE>
      <PERSON.navn>Per</PERSON.navn>
      <BY.navn>Orkanger</BY.navn>
      <TING.navn>HB</TING.navn>
   </BYHISTORIE>
</HISTORIER>
```

regelfil:
<BY type="replace" source="BY.navn" />

32

Ny testkjøring

33

✍Oppgaven, dens utvikling og løsning
✍Presentasjon av løsningen
✍Demonstrasjon av løsningen
✍**Oppsummering**
- Spørsmål og svar

34

Resultatet av prosjektet

- En generell regelbasert kodegenerator, der alle innstillinger kan forandres vha regler og maler
- Implementasjonen oppfyller alle krav med kritisk eller høy prioritet
- Konstruksjonen oppfyller alle kravene som er gitt

35

Mulige forbedringer med nåværende implementajon

- Forbedring av brukergrensesnitt
- Utvidelse av eksisterende regler og maler:
  - Deployment descriptorer i EJB reglene
  - Støtte for relasjoner i EJB reglene
  - Støtte for ulike typer XMI
- Støtte for flere variabler i filnavn.
- Støtte for statiske filnavn

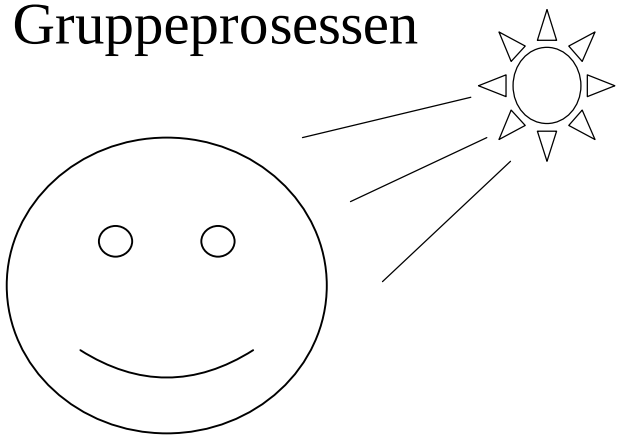
36

Videre utvikling av prosjektet

- Opprettelse av regler og maler for flere systemer:
  - COM /VB
  - C#
- Verktøy for å spesifisere regler og maler
- Bedre feilhåndtering
- XMI konverterer

37

Gruppeprosessen



38

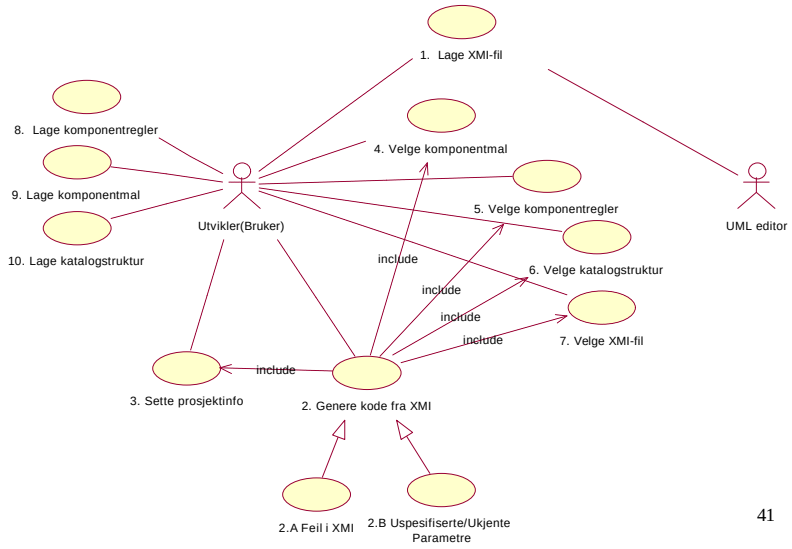
- ✍Oppgaven, dens utvikling og løsning
- ✍Presentasjon av løsningen
- ✍Demonstrasjon av løsningen
- ✍Oppsummering
- ✍**Spørsmål og svar**

39

Ekstra foiler

Taes i bruk dersom det er tid, eller
dersom det dukker relaterte spørsmål

Brukstilfeller



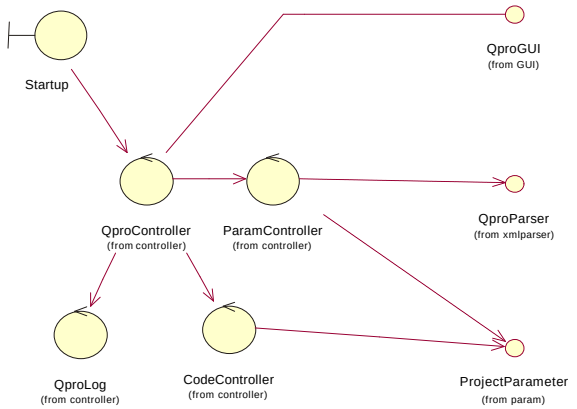
41

Brukstilfelle - estimering

- En metode for å estimere tidsforbruket på å konstruksjon og implementasjon av et system basert på brukstilfellene til systemet
- Særdeles enkel å bruke, krever bare at brukstilfellene spesifiseres i et spesielt format.
- Gav ett estimat på 488,48 timer i vårt tilfelle
- Det reelle tallet var 673,5 noe som gir et avvik på ca 38 %

42

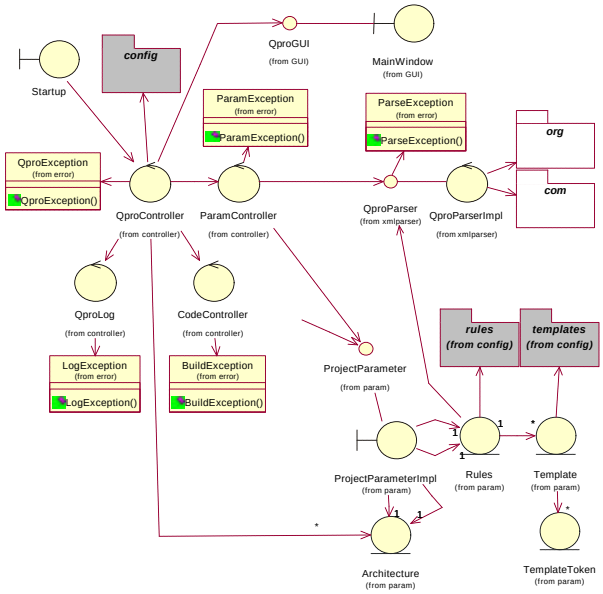
Sentrale objekter i Qpro16



- Kontrollklasser og grensesnitt
- Tjenestene til pakkene definert i grensesnitt
- QproController *kommunikasjon*
- ParamController *validering/parsing*
- CodeController *kodegenerering*

43

Hele systemet



44

Tjenestene fra pakken param

- getXMIDocument – returner XMI modellen
- getProjectName - prosjketnavnet
- getSrcDir – returnerer målkatalogen
- getArchitecture – returnerer valgte arkitektur
- isTrigger – trigger denne noden noen maler
- getTemplatesByTrigger – alle malene noden trigget
- getRuleElement – regelen og dets subtre
- getInternalTemplateByName – returnerer en intern mal
- getIdAttributeName – det attributet som id'en ligger i

45

Mulige attributter i reglene

•type: multi eller replace
•source: refererer til node i inndata
•link:
    trengs ikke bestandig
    referer til hvor sourcenoden finnes
•sourcerestriction:
    trengs sjelden
    betingelse som subtre til source må oppfylle
•template:
    brukes når type="multi"
    angir intern mal
•mandatory: angir om det må finnes inndata for variabel i mal
•map: peker på sett av verdier som skal byttes med andre
•separator: brukes sammen med multi

46

Avansert eksempel

```
public abstract class %CLASSNAME%EJB implements javax.ejb.EntityBean {

   public javax.ejb.EntityContext EJB_Context;

            %ATTRIBUTE%

   /**
    * Constructor
    */
   public %CLASSNAME%EJB() {

   }
```

47

Avansert eksempel

```
<ATTRIBUTE type="multi" source="Foundation.Core.Attribute"
            template="ATTRIBUTE_TEMPLATE" mandatory="false" separator="&#013;    ">
        <ATTRIBUTE_VISIBILITY type="replace" link="“
                source="Foundation.Core.ModelElement.visibility;xmi.value">
        </ATTRIBUTE_VISIBILITY>
        <ATTRIBUTE_TYPE type="replace" link="Foundation.Core.Classifier;xmi.idref“
                source="Foundation.Core.ModelElement.name">
        </ATTRIBUTE_TYPE>
        <ATTRIBUTE_NAME type="replace" link="“
                source="Foundation.Core.ModelElement.name">
        </ATTRIBUTE_NAME>
</ATTRIBUTE>
```

48

Avansert eksempel

```
<INTERNAL_TEMPLATES>
        <ATTRIBUTE_TEMPLATE>
            %ATTRIBUTE_VISIBILITY%
            %ATTRIBUTE_TYPE%
            %ATTRIBUTE_NAME%;
         </ATTRIBUTE_TEMPLATE>
                    .
                    .
                    .
</INTERNAL_TEMPLATES>
```

49