UNIVERSITETET I OSLO
Institutt for informatikk

Virksomhets-modellering som basis for spesifikasjon av IT-systemer

Et metodeforslag

Masteroppgave

Unni Løland

1. mai 2006

Forord

Denne oppgaven er skrevet som en del av min mastergrad ved Institutt for Informatikk, Universitetet i Oslo, profesjonsstudiet. Arbeidet er gjort ved SINTEF, avdeling for informasjons- og kommunikasjonsteknologi, samvirkende og tiltrodde systemer.

Først vil jeg takke min veileder, Dr. Arne-Jørgen Berre, for hans oppfølgning og inspirasjon gjennom hele prosessen. Det har vært interessant og lærerikt, og jeg er takknemlig for å ha fått muligheten til å være med på forskningsprosjektet INTEROP. En spesiell takk går til Ida Solheim ved SINTEF for uvurderlig hjelp i sluttfasen.

Jeg vil også rette en takk til mine medstudenter for hyggelig og faglig samvær gjennom hele studietiden. Takk for at dere har gjort tiden ved universitetet uforglemmelig! Takk også til alle i ProsIT.

Til slutt vil jeg takke familie og venner for all oppmuntring og støtte, og for å ha holdt ut med meg i alle faser av skriveprosessen.

Oslo, 1.mai 2006

Unni Løland

Innhold

Figurer

Tabeller

Kapittel 1

Introduksjon

1.1 Problemet

En utfordring med å lage IT-systemer er å lage dem slik at de blir best mulig tilpasset til hvordan virksomheten fungerer. Hvordan virksomheten fungerer beskrives gjennom dens forretningsprosesser, organisasjonsstruktur, ressurser og forretningsregler. Et nytt IT-system bør ikke være for vanskelig for forretningsfolkene å ta med i deres arbeidsprosesser, altså å ta i bruk. Dersom IT-systemet da er tilpasset virksomheten, vil det være enklere å bruke, fordi det er mer tilrettelagt for personene som skal bruke det. Problemet ligger i hvordan IT-systemet kan tilpasses.

En virksomhet styres av regler. Dette er regler som styrer forretningsprosesser, regler som sier noe om hvordan og når en beslutning skal tas, regler som forteller hvem som har lov til å gjøre hva i virksomheten, regler som definerer hvordan prisen på et produkt skal beregnes, og regler som definerer hva som til enhver tid skal være gjeldenede praksis i en bedrift. Slike regler kalles forretningsregler. Forretningsreglene kan være eksplisitt satt gjennom forretningspolicies eller implisitt ved ulike former for kunnskap i bedriften. Disse reglene kan bli endret ofte. I et IT-system vil de forretningsreglene som er av betydning for IT-systemet være implementert, det vil si at reglene er en del av programkoden som utgjør IT-systemet. Når en forretningsregel endres, kan ikke endringen håndteres effektivt, fordi en må gå gjennom all programkode og finne ut hvor regelen er gitt, og for hvert sted i koden den er gitt, må man endre den. Det å effektivt håndtere endring av regler er en annen utfordring med IT-systemer. Den henger sammen med den første utfordringen fordi regelhåndtering er en viktig del av et IT-system.

1.2 Virksomhetsmodellering – en mulig løsning

For å tilpasse et IT-system til en bedrift kan det være en god hjelp å lage modeller av virksomheten, i hvert fall av den delen av virksomheten hvor IT-systemet vil ha en innvirkning, eller de delene av virksomheten som påvirker IT-systemet. Det å lage slike modeller kan være til hjelp fordi de kan bidra til å skreddersy IT-systemet til virksomheten det lages for. Det å gjøre virksomhetsmodellering er noe som har eksistert lenge, og det finnes utallige metoder og modelleringsspråk for å lage en modell av virksomheten. Disse modellene har til nå ikke blitt brukt noe særlig videre for å spesifisere arkitekturen til et IT-system. Jeg tror at hvis vi bruker virksomhetsmodeller for å lage arkitekturmodeller og i neste runde programkode, så får vi IT-systemer som er bedre tilpasset virksomheten. For å finne ut om dette stemmer må følgende spørsmål besvares:

> **Spørsmål 1:** *Kan virksomhetsmodeller transformeres til arkitekturmodeller og videre til programkode og dermed sikre at IT-systemet tilfredsstiller forretningsfolkenes behov?*

Virksomhetsmodellering gjøres ved hjelp av forretningsfolk. Det er de som har kunnskap om hvordan virksomheten fungerer. Virksomhetsmodellering kan gjøres ved å rådføre seg med forretningsfolk, eller helst ved at forretningsfolk tar del i virksomhetsmodelleringen. Da er det større sannsynlighet for at virksomhetsmodellene blir korrekte.

Forretningsregler må håndteres i et IT-system, fordi dette er regler som endres ofte. Spørsmålet er bare hvordan denne håndteringen kan effektiviseres. Ved å separere reglene i en egen komponent vil regelhåndteringen effektiviseres fordi en komponent er enklere å vedlikeholde enn å finne igjen alle reglene innbakt i andre komponenter. Forretningsregler skal identifiseres på virksomhetsmodelleringsnivå, fordi kunnskapen om disse reglene er det forretningsfolk som sitter inne med. Dette gir oppgav til to spørsmål som må stilles for å kunne gi et svar på hvordan man effektivt kan håndtere forretningsregler i et IT-system:

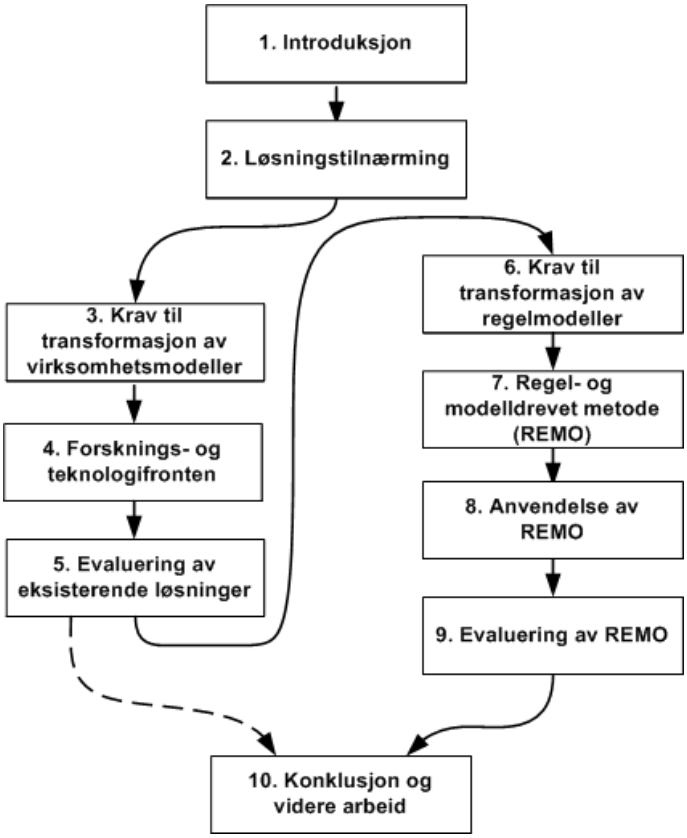
> **Spørsmål 2:** *Kan forretningsregler skilles ut som et eget element i virksomhetsmodellen og transformeres derfra til en arkitekturmodell og videre til programkode?*

> **Spørsmål 3:** *Kan separasjon av forretningsregler fra annen programkode forenkle vedlikeholdet av regler?*

Denne oppgaven søker å besvare disse spørsmålene. Dette vil bli gjort på to måter. Den første er å evaluere eksisterende løsninger for å se om det er mulig å bruke dem. Den andre er ved å introdusere en metode som kan møte de utfordringene som beskrevet innledningsvis, i den grad eksisterende løsninger ikke strekker til.

1.3 Leserveiledning

Denne oppgaven er delt inn i to løp. Den første delen av oppgaven undersøker eksisterende løsninger for virksomhetsmodellering, og ser på muligheten for å bruke disse for å spesifisere systemarkitektur. Den andre delen av oppgaven introduserer en metode for å formulere forretningsregler i en virksomhetsmodell og transformere disse reglene til input til en systemarkitektur. Totalt sett vil disse to delene gi svar på spørsmålene. Den første delen gir mesteparten av svaret for spørsmål 1, og den andre delen gir mest grunnlag for å besvare spørsmål 2 og 3, da den undersøker hvordan forretningsregler kan håndteres. Figur 1.1 viser strukturen i denne oppgaven. I kapittel 2 presenteres løsningstilnærmingen. Dette er en tilnærming som er grunnlaget både for evalueringen av eksisterende løsninger og for utviklingen av metoden som denne oppgaven presenterer. Her vil også en avgrensning av oppgaven gjøres.



Figur 1.1: Strukturen i oppgaven

Kapittel 3 presenterer krav som virksomhetsmodellen må oppfylle for å kunne brukes for å bidra til spesifikasjonen av systemarkitekturen, ved modelltransformasjon. Her stilles det krav til både virksomhetsmodellene og arkitekturmodellene, og til modelltransformasjonene. Kapittel 4 presenterer eksisterende løsninger for virksomhetsmodellering, arkitekturmodellering og modelltransformasjoner. Det hele evalueres i kapittel 5.

Løp nummer to presenterer, etterprøver og evaluerer min metode REMO. Kapittel 6 formulerer kravene til metoden. Det vil si at kravene fra kapittel 3 detaljeres for den biten som er metodens viktigste bidrag, nemlig forretningsregler. Selve metoden presenteres i kapittel 7. Etter at metoden er presentert, gjøres en anvendelse av den, og dette kommer i kapittel 8. Metoden evalueres i kapittel 9.

Kapittel 10 konkluderer og gir forslag til videre arbeid. Konklusjonen er basert både på evalueringen av eksisterende løsninger og evalueringen av min metode.

Kapittel 2

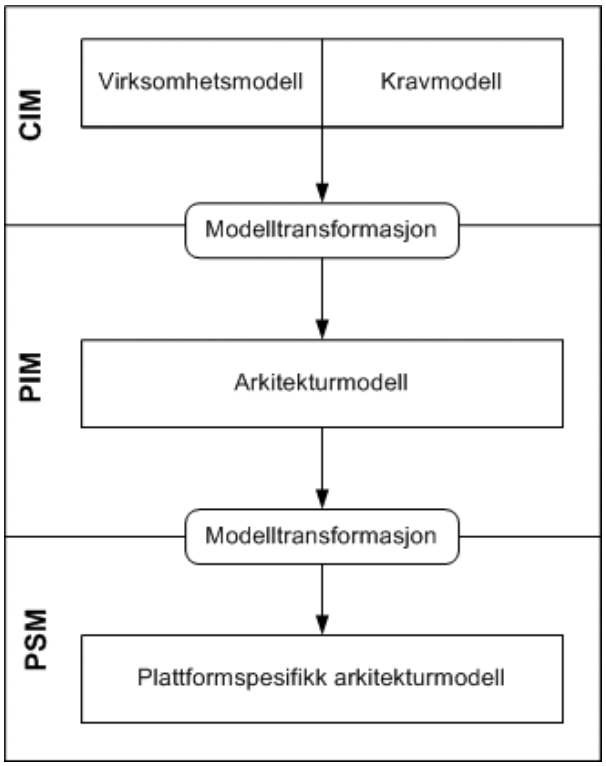
Løsningstilnærming

2.1 Modeller og modelltransformasjoner

Tilnærmingen er basert på konseptet Model Driven Architecture (MDA) [43] som er lansert av Object Management Group (OMG). MDA er utviklet fra tanken om at modellering er et bedre fundament for utvikling og vedlikehold av systemer enn kun kodeskriving. MDA er ikke et bestemt rammeverk, men gir oss ulike synspunkt som vi kan bruke når vi utvikler IT-systemer. MDA er beskrevet mer i detalj i kapittel 4. Denne oppgaven tar utgangspunkt i MDAs tre synspunkter, som består av modeller på tre abstraksjonsnivåer (figur 2.1 på neste side).

Det første nivået i tilnærmingen er Computationally Independent Model (CIM), som består av modeller som er med på å beskrive konteksten hvor IT-systemet skal implementeres og kravene til IT-systemet beskrevet fra virksomhetens synspunkt. På dette nivået vil virksomhetsmodellene og kravmodellen være. Nivå nummer to er Platform Independent Model (PIM) og her vil arkitekturmodellen være. Arkitekturmodellen er plattformuavhengig og skal ikke inneholde elementer som er spesifikke for en bestemt teknisk implementasjon. Ved å gjøre det slik er det enklere å implementere samme IT-system på flere ulike plattformer. Det siste nivået er Platform Specific Model (PSM). Her skal IT-systemet beskrives i henhold til den plattformen det skal implementeres på.

MDA gir veiledning om hvordan transformere PIM til PSM, men ikke hvordan CIM kan transformeres til PIM. Vi skal i denne oppgaven undersøke noe hvordan CIM kan transformeres til PIM. Alt i CIM vil nok ikke transformeres, men håpet er å kunne bruke noen elementer i virksomhetsmodellen og transformere dette til arkitekturmodell eller som input til arkitekturmodellen, altså PIM. Figur 2.1 på neste side viser hvordan modelltransformasjonene vil komme mellom de forskjellige nivåene. Det er viktig å få med at alt i CIM ikke vil transformeres, noe vil bare videreføres som det er og andre elementer vil kanskje ikke brukes videre i det hele tatt. Denne figuren danner grunnlaget for kravene som



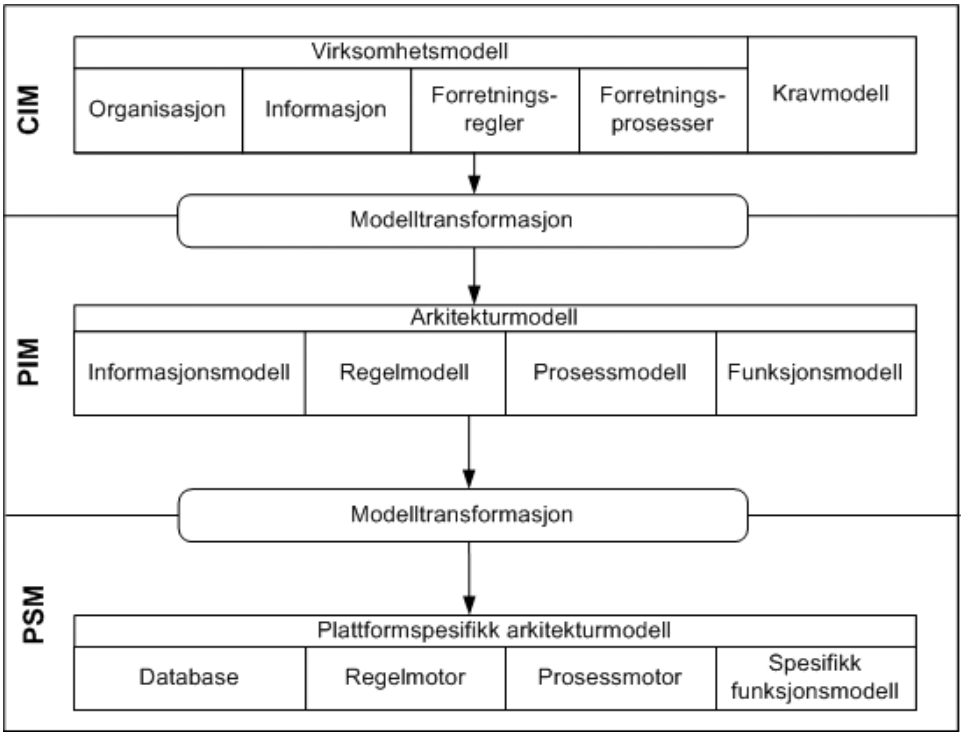
Figur 2.1: Utgangspunkt – modeller på tre abstraksjonsnivåer

skal identifiseres, evalueringen som skal gjøres av eksisterende løsninger og for metoden som denne oppgaven presenterer.

MDA er grunnlaget for tilnærmingen, men for hvert nivå (CIM, PIM og PSM) er det en inndeling. En CIM består av flere modeller, det samme gjør PIM og PSM. I dag finnes det mange forskjellige teknologier som støtter mer direkte opp under deler av virksomhetsmodellene. Det eksisterer f.eks. prosessmotorer som brukes for å implementere prosesser, og regelmotorer som kan håndtere regler. Ved å bruke teknologier som er knyttet mer direkte til hvordan virksomheten fungerer, så kan det være mulig å bruke virksomhetsmodellen til å spesifisere IT-systemene, og IT-systemene kan bli bedre fordi de er bedre tilpasset virksomheten. Med å direkte knytte opp til virksomheten tenker jeg på det å f.eks. kunne eksplisitt gjengi forretningsprosessene i en prosessmotor. Derfor ønsker jeg å ta i bruk slike teknologier, og ha dem med i PSM, i form av plattformspesifikke modeller av disse komponentene. Her vil da reglene skilles ut som en egen komponent, i form av en regelmotor.

Inndelingene av CIM, PIM og PSM har jeg kommet frem til på bakgrunn av litteraturstudier samt inspirasjon fra slike teknologier som nevnt ovenfor. En virksomhetsmodell inneholder i de aller fleste tilfeller tre elementer: informasjonsmodell (også kalt ressursmodell, forretningsvokabular e.l.), organisasjonsmodell og forretningsprosessmodell [69]. I tillegg bør forretningsreglene tas med i en virksomhetsmodell. Forretningsregler bør være

en del av virksomhetsmodellen fordi en forretningsregel er med på å heve strukturen i virksomheten eller å kontrollere eller påvirke oppførselen til virksomheten [1]. Foruten virksomhetsmodell har vi også kravmodell som en del av CIM. Dermed er det fem modeller CIM skal bestå av: organisasjonsmodell, informasjonsmodell, forretningsregelmodell, forretningsprosessmodell og kravmodell. PSM vil bestå av en modell av en bestemt database, en modell av en bestemt regelmotor og en modell av en bestemt prosessmotor, siden vi ønsker å benytte oss av teknologier som støtter mer direkte opp under virksomhetsmodellene. Foruten om det vil også et IT-system tilby noe funksjonalitet som kan vises gjennom en plattformspesifikk funksjonsmodell. Det kunne kanskje vært en mulighet å gå fra CIM direkte til PSM, men det er det i denne oppgaven ikke ønskelig å teste ut. Jeg baserer meg på MDA, og da vil en PIM komme før konkrete tekniske finesser beskrives. På bakgrunn av inndelingene i CIM og inndelingene i PSM får vi også en inndeling på PIM-nivå. En PIM inndeles i fire: informasjonsmodell, regelmodell, prosessmodell og funksjonsmodell. Disse inndelingene er å finne i COMET [10] og PIM4SOA [7], men regelmodellen er mitt bidrag i forhold til dem. Regelmodell er der for at forretningsreglene hele tiden skal være separert fra andre modellelementer, og til slutt separeres som en egen komponent.



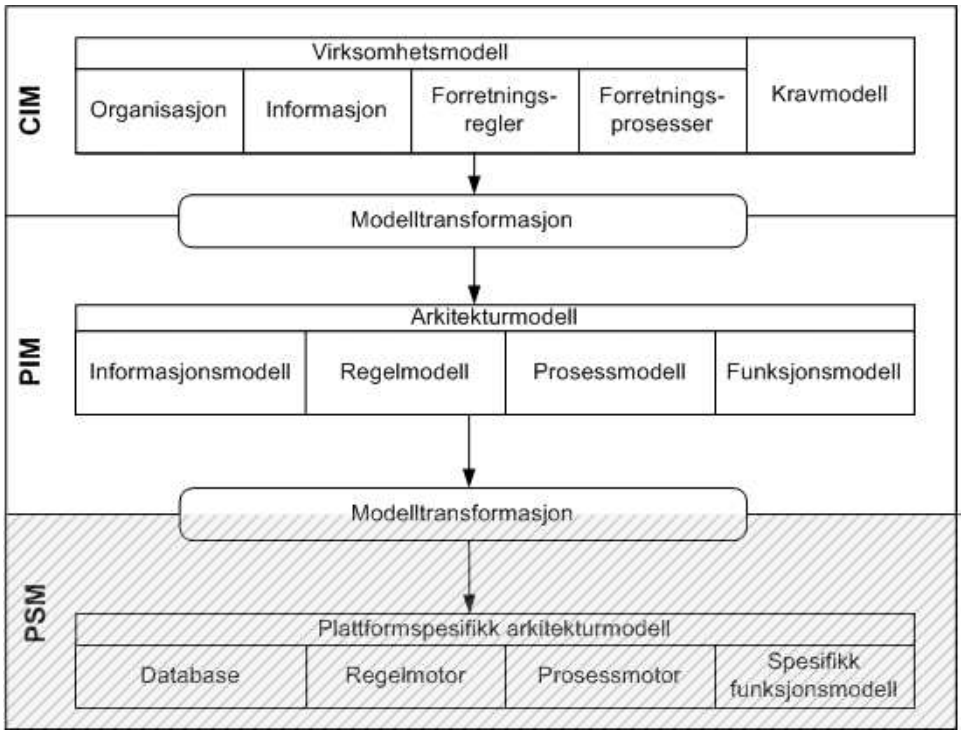
Figur 2.2: Detaljert versjon av modeller på tre abstraksjonsnivåer

2.2 Avgrensning av oppgaven

Tilnærmingen som ble presentert i forrige avsnitt, er meget omfattende. Den er laget med tanke på store, komplekse systemer, og det er mye teori og teknologier som omhandler de ulike elementene i den. Jeg har derfor vært nødt til å avgrense den på to måter. For det første har jeg måttet begrense antall eksisterende løsninger som jeg skulle evaluere. For det andre har jeg måttet avgrense metoden.

2.2.1 Eksisterende løsninger

I forrige kapittel ble tre spørsmål stilt. For evalueringen av eksisterende løsninger er målet hovedsakelig å besvare spørsmål 1, se på muligheten for å transformere virksomhetsmodellene. Dersom deler av virksomhetsmodellene kan transformeres til input til arkitekturmodellen som er i PIM, så kan vi konkludere med at det er mulig å bruke virksomhetsmodellene i spesifiseringen og senere implementasjonen av et IT-system. Overgangen fra PIM til PSM, og videre til programkode finnes det flere veiledninger på og derfor er PSM valgt bort, for regler er denne transformasjonen på vei. Denne avgrensningen er vist i figur 2.3 ved at PSM er skravert. Jeg har dermed begrenset evalueringen til å gjelde eksisterende løsninger for CIM, PIM og modelltransformasjoner. Dette er vist med hvit bakgrunn i figur 2.3

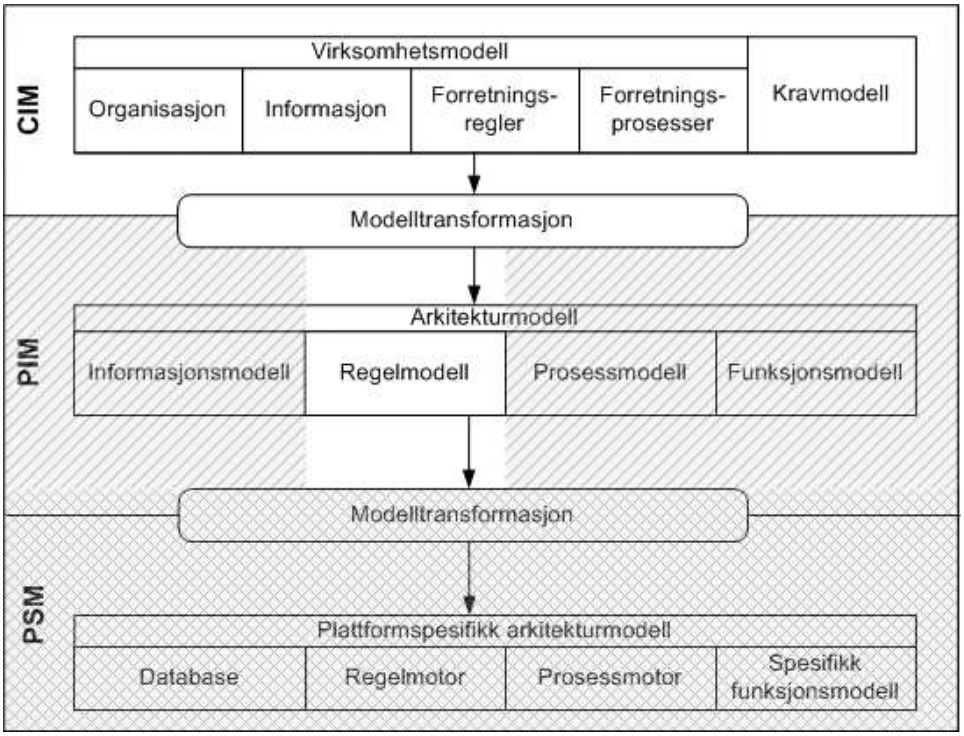


Figur 2.3: Avgrensning av evaluering av eksisterende løsninger

2.2.2 Metoden

Metoden REMO skal gi en veiledning på hvordan forretningsregler kan lages som en del av virksomhetsmodellen og transformeres til input til arkitekturmodellen. Metoden er også laget med den hensikt å bruke virksomhetsmodeller for å spesifisere IT-systemer, og for dette brukes en del av resultatene fra evalueringen av eksisterende løsninger.

REMO dekker det hvite feltet i figur 2.4. For PIM gir REMO en veiledning for alle elementene, men fokuset ligger på regelmodellen. Dette er vist ved at informasjonsmodell, prosessmodell og funksjonsmodell er enkeltskravert i figuren. PSM ser REMO bort fra, og vises ved at PSM er dobbeltskravert.



Figur 2.4: Avgrensning av metoden

Kapittel 3

Krav til transformasjon av virksomhetsmodeller

Dette kapittelet presenterer krav til modelleringen og spesifikasjonene som må gjøres for å komme frem til det IT-systemet som er ønskelig for en bedrift. Kravene er basert på tilnærmingen presentert i kapittel 2. Kravene vil brukes for å evaluere eksisterende løsninger, med tanke på om de kan benyttes for ulike deler av tilnærmingen. De vil også bli brukt for metoden som presenteres senere.

3.1 Krav til Computationally Independent Model (CIM)

Hva er det så som må til for å lage en modell av virksomheten? Dette er spørsmålet vi må stille oss for å komme frem til krav for CIM. I "Enterprise Modeling and Integration"[69] er en virksomhetsmodell definert som "one representation of a perception of an enterprise. It can be made of several submodels, including (but not limited to) process models, data models, resource models, and organization models. The content of an enterprise model is whatever the enterprise considers important for its operations." [69, side 23]

Kravene til virksomhetsmodellen skal gjøre det mulig å bruke virksomhetsmodellering for å spesifisere et IT-system. Jeg har identifisert syv krav til CIM. Kravene er vist i tabell 3.1 på neste side og denne tabellen vil brukes som grunnlag for evalueringen av CIM. Noen av kravene er krav som kommer i tillegg til elementene som er satt til å være en del av CIM i figuren som illustrerer tilnærmingen (figur 2.3). De elementene som ikke er gitt i figuren er brukervennlighet og verktøystøtte. Kravmodellen som vist i figuren omtales her som kravet om funksjonalitetsbeskrivelse.

Krav til CIM		Forklaring
CIM1:	Brukervennlighet	Virksomhetsmodeller skal være forståelige og brukervennlige for forretningsfolk
CIM2:	Verktøystøtte	Alle modellene skal kunne lages i et verktøy
CIM3:	Informasjonsmodell	CIM skal inneholde en informasjons-modell
CIM4:	Organisasjonsmodell	En virksomhetsmodell må vise organisasjonsstrukturen til virksomheten
CIM5:	Forretningsprosess-modell	Forretningsprosessene i virksomheten må modelleres
CIM6:	Funksjonalitets-beskrivelse	Den ønskelige funksjonaliteten til IT-systemet må spesifiseres
CIM7:	Forretningsregel-modell	Forretningsreglene i bedriften som kan være av betydning for IT-systemet må spesifiseres

Tabell 3.1: Krav til CIM

CIM1: Brukervennlighet

CIM skal gi oss en modell av virksomheten, konteksten hvor IT-systemet skal tas i bruk, og en beskrivelse av ønskelig funksjonalitet sett i konteksten av virksomheten. På dette nivået er det ønskelig med en modell som er forståelig for forretningspersoner, personer som arbeider i den virksomheten vi beskriver, fordi det er disse personene som skal hjelpe til med å lage virksomhetsmodellen og verifisere den. Det er derfor et krav til forståeligheten av modellene på dette nivået. Med forståelighet mener jeg at modellene skal være enkle å forstå, gjerne vist på en visuell og intuitiv måte. Akkøk [2] har vist at diagrammatiske språk er enklere å forstå enn naturlig språk, og dette gir grunnlag for at elementer bør modelleres og vises grafisk. Kravet kaller jeg for *brukervennlighet*. Brukervennlig er noe som er enkelt å forstå og også enkelt å bruke. Siden forretningsfolk skal hjelpe til med å lage modellene på CIM-nivå, må også de klare å bruke eller utvikle de modellene vi trenger for videre spesifisering av IT-systemet som skal lages.

CIM2: Verktøystøtte

Det er ikke noe særlig poeng i å lage alle modellene dersom det ikke er noe støtte for å bruke dem videre i systemmodeller. Dersom det er støtte i verktøy kan vi lettere bruke modellene videre i systemmodellene ved å benytte bestemte transformasjoner. Uten noe verktøystøtte ville disse modellene blitt tegnet på papir. Det er greit å tegne modeller på papir for å få et inntrykkk av bedriften, men da blir ikke virksomhetsmodellen brukt så direkte til systemmodellering som det er ønske om. Derfor er det et krav om verktøystøtte for alle modeller som er en del av CIM.

CIM3: Informasjonsmodell

Informasjonen som eksisterer i en virksomhet har mye å si for prosessene og reglene som er i virksomheten. Informasjonen kan vises i en informasjonsmodell. Denne vil da vise ord og uttrykkk som brukes i bedriften: virksomhetens vokabular. En annen grunn til at en informasjonsmodell er viktig er at den gjør det slik at forretningsfolk og IT-folk må snakke sammen og dermed unngå en del misforståelser rundt ord og uttrykk i bedriften. Det å opprette en god forståelse mellom de som skal utvikle IT-systemet og de som skal bruke det, er et steg på veien for å få et egnet system. Dermed er det et krav om at *informasjon* er modellert eller spesifisert i CIM.

CIM4: Organisasjonsmodell

Organisasjonskartet viser informasjon som eksisterer i virksomheten, men informasjonen i organisasjonskartet er mer spesialisert. Den sier noe om hvordan bedriften er organisert; hvem som er hvor, hvilke stillinger som finnes i bedriften og/eller hvilke grupper/avdelinger som er der. Organisasjonskartet er viktig input til andre elementer i en virksomhetsmodell, som f.eks. for å vise hvem eller hva som har ansvaret for en prosess. Det er derfor et krav om at en virksomhetsmodell skal vise organisasjonsstrukturen til virksomheten.

CIM5: Prosessmodell

Prosessene beskriver det som skjer i bedriften, og beskriver samtidig hvordan det skjer. I "Enterprise Modeling and Integration" er en forretningsprosess definert som "a sequence (or partially ordered set) of enterprise activities, execution of which is triggered by some event and will result in some observable or quantifiable end result" [69, side 23].

Av denne definisjonen ser vi at en forretningsprosess kan beskrive mye av hvordan aktiviteter forekommer i en bedrift. Ved å modellere prosessene i virksomheten er det mulig å danne et godt bilde av hvordan virksomheten fungerer. Gjennom prosessmodellering kan vi også tenke oss hvor og hvordan vi skal implementere IT-systemet. Uansett formål med IT-systemet, vil vi gjøre lurt i å analysere og modellere prosessene i en bedrift. Prosessmodellering er og en god teknikk for å involvere forretningsfolk i virksomhetsmodelleringen, ved at de kan tegne og forklare forretningsprosessene for IT-folk, og de kan bruke disse videre i sin systemutviklingsprosess.

CIM6: Funksjonalitetsbeskrivelse

Foruten en beskrivelse av virksomheten i CIM må vi også ha med en beskrivelse av kravene eller funksjonaliteten til systemet vi skal implemente-

re. Det som skal beskrives her er ønskelig funksjonalitet i informasjonssystemet, men beskrivelsen må være på et implementasjonsuavhengig nivå. Denne beskrivelsen må vise hvordan det ønskelige IT-systemet vil passe inn i bedriften; hvem som er aktører, og hva slags funksjonalitet systemet skal tilføre bedriften. Hvor detaljert denne beskrivelsen skal være må bestemmes ut fra hva som er nødvendig for IT-systemet det gjelder. Det er i hvert fall et krav om beskrivelse av funksjonalitet allerede på CIM-nivå.

CIM7 - Forretningsregelmodell

En forretningsregel er noe som styrer virksomheten i henhold til dens forretningspolicy [1]. Regler er viktige på virksomhetsmodellnivå, fordi kunnskapen om regler er det forretningsfolk som sitter inne med. Dette kan være både skrevne og uskrevne forretningsregler. Forretningsregler kan ha ulike detaljeringsnivå, men alle reglene er av stor betydning for virksomheten. Tar vi disse reglene med i virksomhetsmodellen, vil de med større sannsynlighet bli ivaretatt når systemet implementeres. Ved å ha reglene i en egen modell vil vi få god hjelp av forretningsfolk til å identifisere alle, og vi gjør det da eksplisitt, i stedet for at de indirekte kommer frem gjennom andre modeller. Det er også et ønske om å ha forretningsreglene i en egen komponent for å kunne håndtere endringer av dem enklere. Derfor har jeg satt opp krav om at forretningsreglene skal spesifiseres i en egen modell.

3.2 Krav til Platform Independent Model (PIM)

En PIM skal vise systemarkitekturen til et IT-system som skal implementeres, men uten å vise detaljene for hvordan systemet bruker en bestemt plattform [43]. En *arkitektur* kan defineres som "a finite set of interrelated components put together to form a consistent whole defined by its functionality" [69]. Arkitekturmodellen må inneholde alle komponenter vi ønsker i IT-systemet, samt funksjonaliteten som binder disse komponentene sammen. Det finnes ganske mange metoder og tilnærminger for å lage en arkitekturmodell, og målene for kravene til arkitekturmodellen er ikke å definere hvordan en arkitekturmodell skal lages. Målet til kravene er å spesifisere hva som bør være til stede i en plattformuavhengig modell, samt hva som må til for å vise sammenhengen mellom elementene.

I tilnærmingen er følgende presentert som en del av PIM: funksjonsmodell, informasjonsmodell, regelmodell og prosessmodell. I tillegg er det viktig å vise hvordan disse elementene henger sammen. Derfor er det også andre elementer det er krav om som skal være en del av utformingen av PIM. Kravene er vist i tabell 3.2 på neste side.

Krav til PIM		Forklaring
PIM1:	Komponentstruktur	En arkitekturmodell må vise komponent-strukturen
PIM2:	Komponentinteraksjon	Interaksjonen mellom komponentene må vises
PIM3:	Funksjonsmodell	Funksjonaliteten som binder komponentene sammen må spesifieres
PIM4:	Informasjonsmodell	Databasekomponenten må modelleres
PIM5:	Regelmodell	Regelmotorkomponenten må modelleres
PIM6:	Prosessmodell	Prosessmotorkomponenten må modelleres
PIM7:	Interfacespesifisering	Alle komponentinterface må detaljeres
PIM8:	Verktøystøtte	Alle modellene skal kunne lages i et verktøy

Tabell 3.2: Krav til PIM

PIM1: Komponentstruktur

I og med at grunnarkitekturen består av flere komponenter (database, regelmotor og prosessmotor) er det viktig at vi modellerer alle komponentene nøye. Vi må først og fremst vise strukturen. Hvilke komponenter som snakker med hvem og detaljere komponentene dersom de inneholder komponenter/subsystemer. Derfor er det et krav for PIM om en modell som viser *komponentstrukturen*.

PIM2: Komponentinteraksjon

Hver komponent må spesifiseres med innhold, interface og sammenheng med andre komponenter på et mer detaljert nivå enn i komponentstrukturmodellen. For hver funksjon bør en vise hvordan interaksjonen er mellom de ulike komponentene. Ut fra disse interaksjonene vil det være enkelt å identifisere hva som må ligge i et interface i hver komponent og også hva hver komponent bør gjøre.

PIM3: Funksjonsmodell

Det som informasjonssystemet skal gjøre må detaljeres og helst vises i en egen modell. I denne oppgaven vil denne delen kalles funksjonsmodell, og det er et krav om at en slik modell skal være til stede i PIM.

PIM4: Informasjonsmodell

Alt av informasjon i virksomheten som er relevant for IT-systemet skal vises i en informasjonsmodell. Den må være til stede fordi informasjonen som brukes i IT-systemet må tas vare på. En informasjonsmodell vil i implementasjonen være en database. Denne informasjonsmodellen bør

også innebære de delene av informasjonsmodellen i CIM som er av relevans for arkitekturen.

PIM5: Regelmodell

Fra CIM har vi et krav om en egen regelmodell. For å kunne bruke forretningsreglene i implementasjonen, må de også være med i en PIM. Derfor er det her et krav om en regelmodell. Siden reglene skal separeres fra resten av implementasjonen, må dette vises på arkitekturnivå.

PIM6: Prosessmodell

Det er et krav om å ha en egen prosessmodell også på PIM-nivå. Dette er på grunn av to ting: det ene er forretningsprosessene som er på CIM-nivå og det andre er prosessmotoren som er en del av implementasjonen. For å bruke virksomhetsmodellene for å spesifisere IT-systemer må noen av virksomhetsmodellene brukes i arkitekturbeskrivelsen. Forretningsprosesser er et av de viktigste elementene i en virksomhetsmodell, og derfor er det viktig med en prosessmodell også i PIM. Prosessmodellen i PIM kan være noe forskjellig fra prosessmodellen i CIM på den måten at prosessene på PIM-nivå vil være mer rettet mot IT-systemet.

PIM7: Interfacespesifisering

Alle komponenter må ha et interface klart definert slik at det er enkelt for komponentene å snakke sammen. Derfor er det krav om interfacespesifisering.

PIM8: Verktøystøtte

For å lage arkitekturmodellen er det nødvendig med verktøy for å lage modellene. Dersom det ikke er mulig å lage modeller i verktøy blir det vanskelig å transformere dem og bruke dem direkte i implementasjon av IT-systemer. Det er behov for maskinlesbare modeller for å bruke automatisk transformasjon.

3.3 Krav til transformasjoner

For å bruke virksomhetsmodellene til å spesifisere IT-systemer så må modellene transformeres på en eller annen måte. Enten må modellene manuelt tas med videre for å spesifisere arkitekturen til systemet, eller så kan modellene transformeres ved bruk av et modelltransformasjonsspråk. Jeg ønsker å bruke det siste alternativet, fordi jeg mener at vi da får

utnyttet virksomhetsmodellene mest mulig. I MDA er det gitt prinsipper for å transformere fra PIM til PSM, og de prinsippene skal vi benytte oss av. Det finnes lite litteratur om modelltransformasjoner mellom CIM og PIM, og derfor skal jeg senere foreslå noen transformasjoner. Uansett, så er det flere krav til modelltransformasjoner. Jeg har identifisert seks krav for modelltransformasjoner, og disse er vist i tabell 3.3

Krav til transformasjoner	Forklaring
T1: Metamodellbasert	Transformasjonene skal være metamodell-baserte
T2: Forståelighet	Transformasjonsspråket skal være enkelt å bruke og forstå
T3: Ivareta informasjon	Trasformasjonen må ivareta informasjon fra modellen den trasformeres fra
T4: Sporbarhet	Det skal være mulig å spore tilbake transfor-masjonene
T5: Verktøystøtte	Det må være god støtte i verktøy for transfor-masjonene
T6: Standardisering	Transformasjonene må være i henhold til OMGs foreslåtte standard for modell-transformeringsspråk

Tabell 3.3: Krav til modelltransformasjoner

T1: Metamodellbasert

For å transformere en modell til en annen modell må det spesifiseres mappinger. Mappinger er relasjonene mellom modellene, hvilke elementer i kildemodellen (modellen som det transformeres fra) som skal transformeres til hvilke elementer i målmodellen (modellen som det skal trasformeres til). Det finnes ulike typer mappinger. Modeller kan mappes ved å bruke modelltypemapping, metamodellmapping, modellinstansmapping og andre som er kombinasjoner av disse [43]. Ved å bruke metamodellbasert mapping, defineres mappingene mellom metamodellene. Dette krever at alle modeller som det skal transformeres fra og til er definerte med metamodeller. Da blir transformasjonskoden skrevet for hver mapping på et overordnet nivå, og er lett å gjenbruke for modeller laget i henhold til de metamodeller som er basert. Det betyr at dersom vi definerer mapping og transformasjonsregler mellom en del av CIM til en del av PIM, så kan samme modelltransformasjon brukes for alle prosjekter som ønsker å transformere en gitt modell i CIM til den i PIM. Derfor stilles det et krav om at transformasjonene skal være metamodellbaserte.

T2: Forståelighet

Transformasjonsspråket skal være lett å bruke og det skal være lett å forstå. Selv om det er metamodellbasert og mulig for gjenbruk vil det kunne

oppstå behov for å forandre på mappinger eller transformasjonsregler, og da er det greit at koden ikke er alt for kompleks, eller vanskelig å forstå. Derfor stiller jeg krav om forståelighet til transformasjonene. For at det skal være forståelig og lett å bruke må det være god dokumentasjon av språket.

T3: Ivareta informasjon

Når en modell transformeres til en annen modell er det en viss fare for at informasjon kan gå tapt. Det er viktig å få med seg det som det er behov for videre i modellene det transformeres til. Det er nødvendigvis ikke alt i en virksomhetsmodell som er nødvendig å ta med seg til en arkitekturmodell, men det som defineres som nødvendig må være med så fullstendig som mulig. Derfor er det et krav til at trasformasjonene ivaretar all informasjon som er definert som nødvendig. Det som defineres som nødvendig er det som det er behov for å ha med i arkitekturmodellen.

T4: Sporbarhet

Når CIM er transformert til PIM, så er det ikke absolutt at CIM er helt ferdig. Underveis i utvidelsen av PIM vil det være mulig å oppdage elementer som burde vært gjort på CIM-nivå, men som enda ikke er blitt modellert eller spesifisert. Da kan det være ønskelig å transformere tilbake igjen, det vil si, transformere fra PIM til CIM. Derfor er det et krav om at *sporbarhet* er mulig i transformasjonsspråket eller transformasjonsverktøyet. Dette gjør det også enklere å oppdatere IT-systemer.

T5: Verktøystøtte

Noe av det viktigste med transformasjonene er at det er mulig å bruke. Det vil si at det finnes gode verktøy for det som også gjør det mulig å bruke modellen vi lager i CIM, PIM og PSM som input og output modeller. Dersom vi ikke får transformert modellene våre, er det lite poeng i å ha et fint språk når det ikke kan brukes. Derfor er et av de viktigste kravene for transformasjonene *verktøystøtte*.

T6 - Standardisering

OMG har presentert en standard for modelltransformasjoner som heter Query View Transformation (QVT). Det er ønskelig at transformasjonsspråket er laget i henhold til denne standarden.

Kapittel 4

Forsknings- og teknologifronten

4.1 Valg av teknologier

I forrige kapittel presenterte jeg et utgangspunkt med modeller på tre abstraksjonsnivå, for hvordan spesifisering av et IT-system bør gjøres med fokus på virksomhetsmodellering. Vi skal i dette kapittelet se på forsknings- og teknologifronten som er av relevans for dette utgangspunktet. Utgangspunktet er basert på MDA, og derfor er det første som blir presentert i dette kapittelet en beskrivelse av MDA.

For hver del av modellen (CIM, PIM og PSM) eksisterer det ulike metoder og teknologier. I tillegg finnes det verktøy og språk for å gjøre modelltransformasjoner. Denne oppgaven begrenser seg til kun å se på CIM, PIM og transformasjon mellom CIM og PIM. For CIM er disse valgt:

- Architecture of Integrated Information Systems (ARIS)
- Extended Enterprise Modeling Language (EEML)
- Process Organisation Product * (POP*)
- Semantics of Business Vocabulary and Rules (SBVR)

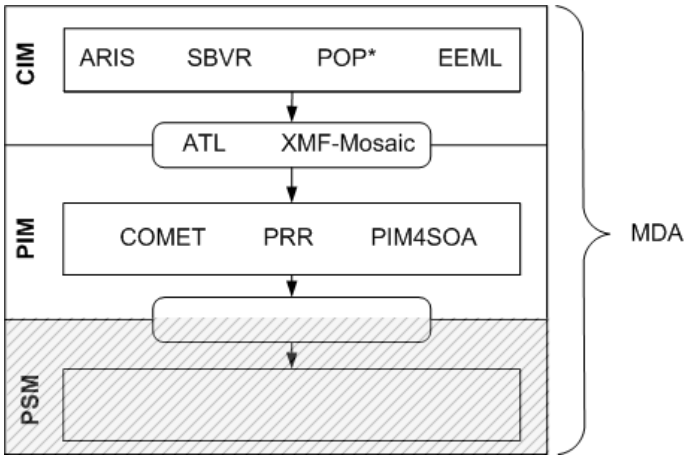
Det eksisterer mange rammeverk, metoder og språk for virksomhetsmodellering og i tillegg B er det beskrevet noen flere enn de som presenteres her.

For PIM er følgende valgt:

- Component and Model-based Development Methodology (COMET)
- Platform Independent Model for Service Oriented Architecture (PIM4SOA)
- Production Rule Representation (PRR)

PRR er valgt ut fordi denne er forventet å kunne bruke som regelmodell i PIM. Når det gjelder modelltransformasjoner skal vi se nærmere på XMF-Mosaic og Atlas Transformation Language (ATL). På grunn av avgrensingen skal vi ikke se på PSM. Jeg har gjort noe arbeid når det gjelder implementasjonsteknologier for regler, og dette arbeidet er lagt ved i tillegg F.

Figur 4.1 viser hvilke metoder og teknologier som dette kapittelet tar for seg, relatert til utgangspunktet som ble presentert i kapittel 2. MDA har å gjøre med alle tre abstraksjonsnivåene, og det ser vi illustrert i figuren. For CIM-nivå ser vi ARIS, SBVR, EEML og POP* er satt inn, men hvor disse er satt må ikke sees i forhold til inndelingen av CIM i figur 2.3 på side 8. De valgte teknologiene skal sees på for hele CIM, det samme gjelder for PIM.

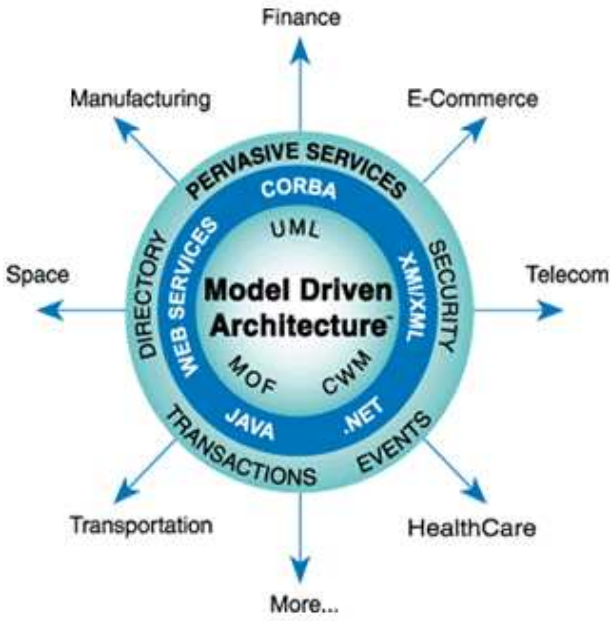


Figur 4.1: Teknologier som kan bidra til å realisere den valgte tilnærmingen

4.2 Modelldrevet arkitektur

Model Driven Architecture (MDA) [43] er et initiativ fra OMG for å utvikle standarder basert på den ideen at modellering er et bedre fundament, enn kun kodeskrivning, for utvikling og vedlikehold av systemer [40]. MDA er ikke et bestemt rammeverk, men gir ulike synspunkt og veiledning på hvordan vi skal gjøre utviklingen. De tre primære målene til MDA er portabilitet, interoperabilitet og gjenbrukbarhet ved å separere elementer på arkitekturnivå. MDA gir en tilnærming for å spesifisere et plattformuavhengig system, spesifisere plattformer, velge en plattform for et system og transformere alle systemspesifikasjonene til en, for en bestemt plattform [43]. Detaljene rundt modellene, som modelleringsspråk og implemtasjonsteknologi, må vi bestemme selv, alt etter hva som er mest hensiktsmessig.

MDA gir tre synspunkt å ta utganspunkt i: datauavhengig synspunkt (computation independent viewpoint), plattformuavhengig synspunkt (platform independent viewpoint) og plattformspesifikt synspunkt (platform specific viewpoint). Disse tre synspunktene gir oss tre modeller i



Figur 4.2: Modelldrevet arkitektur [82]

MDA: Computationally Independent Model (CIM), Platform Independent Model (PIM) og Platform Specific Model (PSM).

CIM er et syn på et system fra det datauavhengige synspunktet. CIM viser ikke detaljene til strukturen av systemet, men den viser kravene til systemet. CIM beskriver situasjonen som systemet skal brukes i [43]. Derfor kaller vi gjerne denne modellen for en domenemodell eller en forretningsmodell. En slik modell er helt uavhengig av hvordan systemet blir implementert. En CIM kan inkludere mange modeller, noen som gir mer detaljer enn andre, eller som fokuserer på bestemte problemområder.

En PIM er et syn på et system fra det plattformuavhengige synspunktet. En PIM viser en gitt grad av plattformuavhengighet, slik at den kan være passende for flere forskjellige plattformer av lignende type [43]. Denne modellen beskriver systemet, men viser ikke detaljer rundt bruk av en gitt plattform (noe som egentlig fremgår av navnet). En PIM kan bestå av spesifikasjoner fra ulike synspunkt, som virksomhet, informasjon og beregning. PIM gir flere fordeler: man står relativt fritt til å velge plattform, man får en høyere grad av gjenbrukbare systemer og man gjør det lettere med tanke på interoperabilitet mellom ulike plattformer. Hvis f.eks. et informasjonssystem skal implementeres i forskjellige bedrifter som bruker ulike plattformer, er dette enklere dersom systemet er modellert fra et plattformuavhengig synspunkt.

En PSM er et syn på et system sett fra et plattformspesifikt synspunkt [43]. PSM kombinerer spesifikasjonene fra PIM med detaljer som spesifiserer hvordan systemet bruker en bestemt plattform. Tidligere har en gjerne gått direkte til dette nivået, og da har ikke gjenbruk vært like enkelt, med tanke på ulike plattformer.

Modelltransformasjon er prosessen som går ut på å konvertere en modell til en annen modell av samme system [43]. MDA gir veiledning på hvordan en PIM transformere til en PSM. Når vi skal transformere en PIM til en PSM trenger vi en spesifikasjon på hvordan vi skal gjøre det, dette kalles for en mapping. PIM til PSM transformering er en teknikk som det er sett en del på. Ved å gjøre mappinger vil overgangen til PSM være enkel og robust. Transformering fra CIM til PIM er det dessverre ikke sett på i like stor grad i MDA.

4.3 Virksomhetsmodell

Virksomhetsmodellering kan defineres som "the process of building models of whole or part of an enterprise (e.g. process models, data models, resource models, new ontologies etc.) from knowledge about the enterprise, previous models, and/or reference models as well as domain ontologies and model representation languages" [69]. Denne definisjonen viser hvor bredt virksomhetsmodelleringsfeltet er; utviklingen av alle typer modeller som kan illustrere en del av virksomheten, betraktes som en del av virksomhetsmodelleringen. I dag finnes det mange ulike metoder, rammeverk og språk som er utviklet spesielt for virksomhetsmodellering.

Fremgangsmåter for virksomhetsmodellering kan deles inn i fire hovedkategorier [18]:

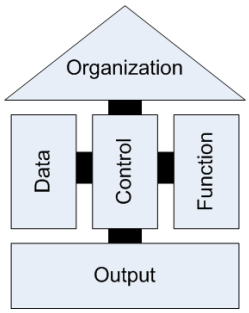
1. Menneskelig forståelse og kommunikasjon: Å finne meningen i aspekter ved en virksomhet og kommunisere dette til andre mennesker
2. Analyse ved hjelp av datamaskin: Å få kunnskap om virksomheten gjennom simulering eller deduksjon
3. Ta i bruk og aktivere modeller: Å integrere modellen i et informasjonssystem, og dermed tvinge det til å aktivt delta i arbeidet som utføres av organisasjonen.
4. Å gi konteksten for et tradisjonelt systemutviklingsprosjekt, uten at det er direkte implementert.

De fleste metoder, språk, rammeverk og verktøy som er utviklet, er vinklet mot kategori 2 og 3.

Det finnes flere organisasjoner som driver med standardisering av virksomhetsmodelleringsspråk og -metoder. De mest sentrale er: Business Process Management Initiative (BPMI), OMG, Organization for the Advancement of Structured Information Standards (OASIS), IInternational Standards Organization (ISO) og OpenGroup. I tillegg B er det beskrevet en del generelle rammeverk og språk for virksomhetsmodellering. I dette avsnittet har jeg valgt ut ARIS, EEML, POP* og SBVR som metoder og modelleringsspråk som jeg ønsker å beskrive nærmere. Disse vil bli evaluert i kapittel 5 i henhold til kravene stilt i kapittel 3.

4.3.1 Architecture of Integrated Information Systems (ARIS)

ARIS [62] er et rammeverk basert på konseptet om forskjellige synspunkt. Her brukes ikke synspunkt på samme måte på som i MDA. Synspunkt retter seg mer mot ulike deler av virksomheten. Målet er å redusere kompleksiteten ved å dele virksomheten inn i individelle synspunkt. De delene virksomheten deles inn i er: *function view*, *organization view*, *data view*, *output view* og *control view*. Dette kan illustreres ved ARIS-huset, se figur 4.3.



Figur 4.3: ARIS-huset [62]

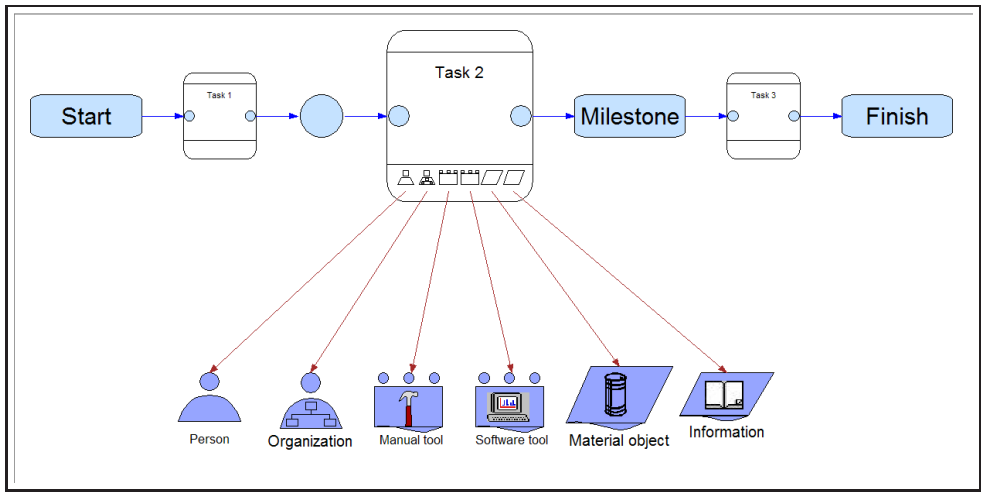
I funksjonsdelen modelleres alle funksjoner som gjelder for virksomheten vi modellerer. Funksjoner blir ofte beskrevet i forhold til andre komponenter. Funksjonsdelen er nært knyttet til datadelen, dette fordi funksjoner gjør om inndata til utdata. Organisasjonsdelen er ment for å modellere organisasjonen i virksomheten. Det vises med organisasjonskart. Datadelen inkluderer beskrivelsen av dataobjekter, manipulert av funksjoner. Dataobjekter er hendelser, meldinger, omgivelser eller informasjonstjenester. Outputdelen viser all output fra forretningsprosessene. Output er resultatet av en prosess. Den viktigste delen er kontrolldelen. Denne delen er koblingen mellom alle andre deler, og det er her selve forretningsprosessen modelleres. Her vises det hvordan dataene kobles opp mot funksjonene, hvilke hendelser som inntreffer, hva som er output fra prosessene og hvilke organisasjonsenheter som er inkludert i prosessene.

4.3.2 Extended Enterprise Modeling Language (EEML)

Extended Enterprise Modeling Language (EEML) er et modelleringsspråk som er laget for å støtte prosessmodellering på tvers av flere lag [37]. De lagene som er av interesse for denne oppgaven er: beskrive prosesslogikk, konstruere aktiviteter, håndtere arbeid og utføre arbeid. Ved bruk av EEML har man en mulighet for å modellere hele flyten i en virksomhet, med enkle modeller.

I EEML er det to sentrale domener man modellerer i forhold til: EEML Process Domain og EEML Resource Domain. I prosessdomenet beskriver man forretningsprosesser. I forretningsprosessen kan man vise f.eks. sam-

menhengen mellom oppgaver og beslutningene i virksomheten. I ressursdomenet beskriver man de ressursene man har, som f.eks. personer og organisasjoner. Sammenhengen mellom prosessdomenet og ressursdomenet beskrives med relasjoner. En oppgave har en rolle, og en rolle er bekledd av en ressurs. Slik viser man hva de ulike oppgavene i virksomheten er, samtidig som man viser hvem eller hva som har ansvaret for dem, og hvilken rolle de eller det spiller i denne oppgaven. Figur 4.4 gir en oversikt over de ulike objektene vi modellerer med i EEML. Her ser vi at *person*, *organization*, *software tool*, *manual tool*, *information* og *material object* hører til ressursdomenet, mens *task*, *milestone*, *start* og *finish* er elementer i prosessdomenet.



Figur 4.4: Oversikt over objekter i EEML

4.3.3 POP*

POP* [5] er en metodikk som er utviklet i ATHENA-prosjektet (se tillegg G for beskrivelse av ATHENA). Denne metodikken brukes for å takle interoperabilitetsproblemer mellom forskjellig virksomhetsmodeller i ulike bedrifter. POP* er laget for å være et utvekslingsformat, slik at bedrifter kan utveksle virksomhetsmodeller selv om de anvender forskjellige modelleringsspråk og verktøy. POP* baserer seg på mange ulike virksomhetsmodelleringsteknikker som f.eks ARIS, Graphs with Results and Activities Interrelated (GRAI) m.m.

POP* har to deler:

1. POP* metamodellen som definerer konseptene og forholdene deres for POP* modelleringskonstrukter.
2. Et sett av retningslinjer og scenario som beskriver hvordan metamodellen bør brukes og håndteres for å utveksle virksomhetsmodeller.

POP* metamodellen gir et relativt lite sett av elementære modelleringselementer basert på alle virksomhetsmodelleringsspråk fra partnere i ATHENA. Metamodellen for POP* er organisert i henhold til kunnskapsdi-

mensjoner. En slik dimensjon gir et perspektiv som kan anvendes på den virksomheten som modelleres. Dimensjonene som det deles inn i er: **p**rosessdimensjon, **o**rganisasjonsdimensjon, **p**roduktdimensjon, beslutningsdimensjon og infrastrukturdimensjon. De tre første dimensjonene (process, organisation og product) gir opphavet til navnet POP*.

POP* metamodellen kan brukes som grunnlag for å lage virksomhetsmodeller. Det er blitt laget en UML-profil for POP* som kan brukes for å modellere virksomheter, og det er denne som vil bli brukt for evaluering av POP*. Metamodellen presenteres i neste kapittel i forbindelse med evalueringen, og en utfyllende beskrivelse av metamodellen er å finne i tillegg C.

Retningslinjer for bruk av POP*

POP*-metamodellen er laget i den hensikt å støtte interoperabilitet mellom virksomheter. Denne metamodellen er derfor laget slik at det er mulig å uveksle virksomhetsmodeller mellom virksomheter som bruker ulike verktøy for virksomhetsmodellering. De retningslinjene som er definert for POP* er retningslinjer for hvordan vi kan bruke POP*-metamodellen som utvekslingsformat. Det er definer tre retningslinjer for dette:

1. **Mappingkonstrukter mellom et spesifikt virksomhetsmodelleringsspråk og POP*.** Den viktigste oppgaven er å definere en mapping som viser forholdene mellom elementene i POP* og elementene i virksomhetsmodelleringsspråket som skal oversettes. Det er allerede definert flere slike mappinger under utviklingen av POP*. Dette er mappinger fra kjente virksomhetsmodelleringsspråk som partnerne i ATHENA hadde erfaringer med.
2. **Oversett modeller til POP*.**
3. **Lag en fil for utveksling av metadata.** Til slutt må det genereres en eXtensible Markup Language (XML) fil for modellutveksling ved å bruke XML Interchange Format fra POP*.

4.3.4 Semantics of Business Vocabulary and Rules (SBVR)

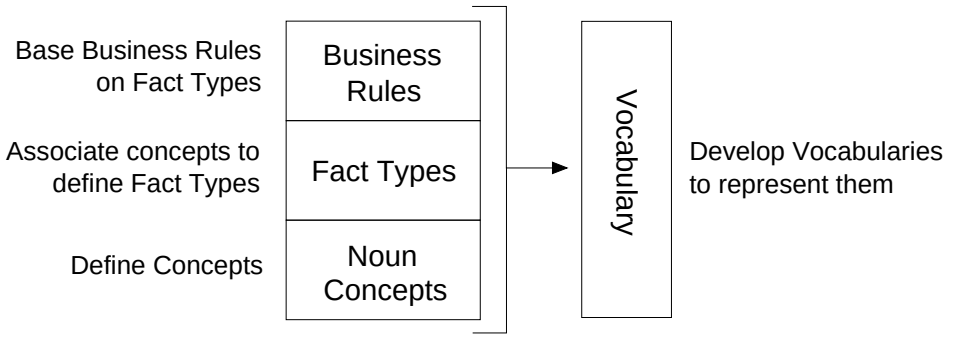
SBVR [1, 53] er en tilnærming for å spesifisere forretningsregler. Tanken bak SBVR er at dette skal være en måte som er forståelig for forretningsfolk, siden det er forretningsfolk som er med på å spesifisere forretningsreglene som eksisterer i en bedrift. Derfor er naturlig språk valgt som uttrykksform for SBVR.

SBVR er svaret på en Request For Proposal (RFP) fra OMG som Business Rules Team (BRT) har laget. BRT er navnet på en gruppe bestående av ulike aktører, bl.a. personer fra Business Enterprise Integration (BEI) [76] og Business Rule Group (BRG) [77]. RFP-en [25] som ble formulert

foreslo å lage følgende for å standardisere opprettelsen og bruken av forretningsregler:

- En metamodell som forretningsfolk kan bruke til å spesifisere forretningsregler.
- En metamodell for å uttrykke vokabularer og definisjoner av termene som brukes i forretningsregler.
- Representasjon med XML.

"Rules build on facts, and facts build on concepts as expressed by terms" [23]. Dette er tankegangen bak SBVR. Først skal forretningsvokabularet defineres, og deretter forretningsreglene, og alle forretningsreglene må være understøttet av forretningsvokabularet. Regler bygger på fakta, som igjen baserer seg på konsepter og for å uttrykke det trenger vi et vokabular. Dette gir SBVR oss og er illustrert i figur 4.5.



Figur 4.5: SBVR [64]

Forretningsvokabularet består av alle konsepter og uttrykk i bedriften, og også faktatyper. Et konsept er en enhet av kunnskap som er laget av en unik kombinasjon av karakteristikker [1]. Undergrupper av konsepter er *noun concept* og *fact type*. Man definerer alle konsepter ved å si hvilken concept type de er. De mest sentrale konsepttypene er *role*, *individual concept*, *characteristic* og *binary fact type*. Et fakta er en påstand som er antatt å være sann i virksomheten. Et enkelt eksempel på et fakta er "person har navn". Her er da faktaet basert på to konsepter, konseptet person og konseptet navn, og konseptene er bundet sammen med verbet *har*.

En forretningsregel kan kategoriseres som strukturell forretningsregel eller operativ forretningsregel. En strukturell forretningsregel er en regel som har den hensikt å opptre som et kriterium for definisjon. En operativ forretningsregel er en regel som har den hensikt å produsere en passende designeffekt. En forretningsregel er basert på en eller flere *fact types*.

SBVR har definert metamodeller for hvordan forretningsvokabular og forretningsregler skal uttrykkes. En del av metamodellene er beskrevet nærmere i tillegg E, sammen med nærmere forklaring av bestanddelene i SBVR og oppbygningen av det.

Sentralt i SBVR er muligheten for å representere vokabularet med XMI, slik at det er enkelt å utveksle vokabularer mellom virksomheter.

4.3.5 Verktøy for SBVR

Unisys Rules Modeler (URM) er et av de første verktøyene som baserer seg på SBVR. URM kjører på Microsoft Visual Studio 2005 og har også en template som kan brukes i Microsoft Word. Her, som i SBVR, er det naturlig språkmodellering som er fokuset, men URM har også støtte for å transformere til og fra UML-modeller. URM benytter XMI for å kunne utveksle vokabularer mellom virksomheter. Dermed er muligheten for interoperabilitet mellom virksomheter større.

URM har foreslått sin egen fremgangsmåte for å identifisere regler, men denne er tilnærmet lik det vi har fra SBVR. Termer, faktatyper og regler er tre kjente begreper som er å finne igjen i URM. En av forskjellene er inndelingen av regler; URM definerer tre kategorier, i motsetning til to kategorier som det er i SBVR. Kategoriene i URM er kravregler, defineringsregler og deklerasjonsregler [66]. Med URM er det enkelt å generere IT modeller, software komponenter og databaser, med mulighet for lett å spore tilbake til reglene og vokabularet som ligger til grunn for det.

Kapittel 5 inkluderer en evaluering av URM som en del av evalueringen av SBVR.

4.4 Plattformuavhengig modell

Den plattformuavhengige modellen skal vise arkitekturen til IT-systemet. Det finnes mange måter å gjøre det på, og her er det valgt ut tre som skal studeres: PRR, COMET og PIM4SOA. PRR er en standard foreslått for produksjonsregler, og produksjonsregler kan brukes for regelmodell i PIM. COMET og PIM4SOA er begge generelle metoder som bør kunne brukes for store deler av PIM, slik som gitt i tilnærmingen i kapittel 2.

4.4.1 Produksjonsregler

Forretningsregler er regler som hører hjemme på virksomhetsnivå og kan beskrive alt fra integritet til prosesser i virksomheten. For at regler skal implementeres som en uavhengig komponent i systemet må reglene også vises som en egen komponent i arkitekturmodellen (jamfør kapittel 3). Det som er foreslått for PIM-nivå er produksjonsregler. PRR [54] er foreslått som en ny OMG-standard for å forbedre modellering av produksjonsregler, spesielt med tanke på Unified Modeling Language (UML) og MDA. En *produksjonsregel* er definert som "a statement of programming logic that specifies the execution of one or more actions in the case that its conditions

are satisfied" [49]. En produksjonsregel skiller seg fra forretningsregler på den måten at den er mer rettet mot IT-systemet og ikke virksomheten, og at den definerer mer oppførsel i IT-systemet enn det forretningsregler gjør.

Flere leverandører av regelmotorer har blitt med for å utvikle denne standarden. PRR gir et forslag til en standard representasjon av en produksjonsregel. Den foreslåtte representasjonen skal være enkel å mappe til forretningsregler på CIM-nivå, og også til konkrete regelmotorer på PSM-nivå. PRR gir en plattformuavhengig modell med høy sannsynlighet for støtte på PSM-nivå fra leverandørene av regelmotorer som bidrar i utarbeidingen av PRR.

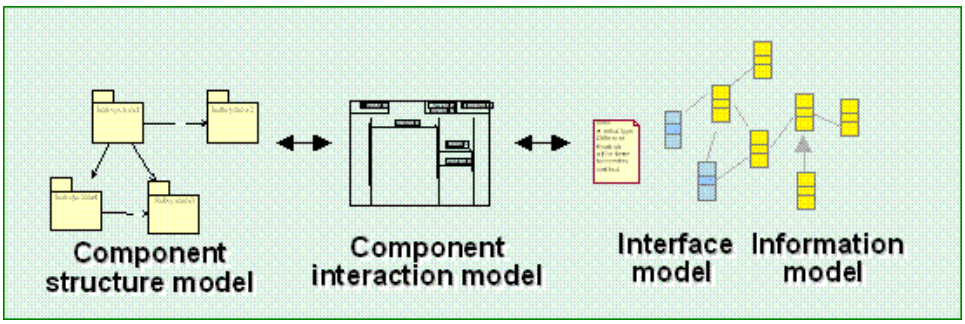
Hovedhensikten til metamodellen foreslått i PRR er å støtte språk som kan brukes med UML-modeller. Ved å gjøre det kan man eksplisitt representere produksjonsregler som synlige, separate og primære modellelementer i UML-modeller [54].

En produksjonsregel er på formen *if [condition] then [action-list]*. Forretningsregler som f.eks. angir integritet vil ikke være en produksjonsregel. Et eksempel kan være: *et produkt X kan kun ha en pris*. Dette er en regel som kan identifiseres på virksomhetsnivå som en forretningsregel, men den har ingenting å gjøre med oppførsel, og representeres derfor ikke som en produksjonsregel, men heller som en del av et klassediagram eller lignende. Det er forretningsregler som definerer oppførsel som er det som kalles produksjonsregler på PIM-nivå.

I PRR [54] står det at det er forventet at når standarder for virksomhetsmodellering, inkludert forretningsregler, blir publisert, at elementer i virksomhetsmodellen/CIM skal være mulige å mappe, via en transformasjonsstandard, til UML-modellelementer som PRR på PIM-nivå, i henhold til prinsippene i MDA.

4.4.2 COMET

I COMET er det definert fire modeller som skal lages for å utforme et IT-system. Først er det forretningsmodellen (Business Model) som inneholder forretningsprosesser, forretningsressurser og Work Analysis Refinement Model (WARM). Deretter er det det som kalles for kravmodell i COMET. Kravmodellen består av system boundary model, use case scenario model og referansearkitekturmodell [10]. Forretningsmodellen og kravmodellen brukes videre som input til arkitekturmodellen. I arkitekturmodellen er det en komponentstrukturmodell, en komponentinteraksjonsmodell, en interfacemodell og en informasjonsmodell. Arkitekturmodellen brukes deretter videre for å lage en plattformspesifikk modell. I COMET tilsvarer forretnings- og kravmodellen CIM fra MDA, arkitekturmodellen tilsvarer PIM og den plattformspesifikke modellen tilsvarer da selvsagt PSM. COMET kan brukes på alle tre nivåer, men jeg har valgt å kun evaluere arkitekturmodellen.



Figur 4.6: Utvikling av arkitekturmodell i COMET [10]

Arkitekturmodellen beskriver den overordnede arkitekturen til systemet og dets inndeling i komponenter. Dette gjøres ved å vise hvordan komponenter og subsystemer samarbeider, vise komponentstrukturer, komponentinteraksjoner og interface og protokoller til komponentene [10]. Arkitekturmodellen består av 4 modeller. Komponentstrukturmodellen beskriver komponentene på et høyt nivå, og viser også deres interne avhengigheter. Komponentinteraksjonsmodellen beskriver interaksjonene mellom komponentene med et eller flere sekvensdiagrammer. Interfacemodellen beskriver i detalj interfacene til komponentene. Det som da beskrives er komponentenes operasjoner og detaljert oppførsel. Informasjonsmodellen viser hvordan all informasjon som brukes er strukturert. Figur 4.6 viser hvordan en arkitekturmodell utvikles i COMET.

Poenget med *Component Structure Model* er å forstå og beskrive de komponentene som bygger opp produktet, avhengighetene mellom komponentene, interfacene som komponentene tilbyr, og komponentenes bruk av andre komponenter gjennom deres interface [10]. Modellen bør dokumentere softwarearkitekturen som er representert gjennom komponentene som produktet består av og modellen bør vise avhengighetene mellom komponentene. *Komponentinteraksjonsmodellen* fokuserer på samarbeidet/samhandlingen mellom komponenter med den hensikt å tilby tjenester. En interaksjonsmodell kan modelleres med et UML sekvensdiagram og/eller UML aktivitetsdiagram. *Component Interface Model* beskriver interfacet til hver komponent som er identifisert gjennom de andre komponentmodellene. Spesifikasjonen av interfacemodellen bør fange opp og beskrive alle systemtjenestene i detalj. *Component Information Model* presenterer et sett av UML klassediagrammer som beskriver informasjonsmodellen for de tilhørende komponentene. Informasjonen er det som kommer frem fra operasjonene/metodene som er dokumentert i interfacemodellen.

4.4.3 PIM4SOA

PIM4SOA er laget for å minske gapet mellom virksomhetsmodeller og tjenesteorienterte implementasjoner [7] og er utviklet i ATHENA-prosjektet. PIM4SOA har POP* metamodellen på CIM-nivå som utgangspunkt. Fi-

re viktige aspekter blir dekket av PIM4SOA: tjenester, prosesser, informasjon og Quality of Service (QoS). Informasjonsdelene holder all informasjon som er relevant for virksomheten. Dette er en av de viktigste bitene i PIM4SOA fordi de andre tre er avhengige av informasjonsdelen, og benytter seg til stadighet av den. Tjenestebiten beskriver den funksjonaliteten som skal tilbys i virksomheten, og er med på å skille funksjonaliteten til systemet fra den tekniske implementasjonen. Prosessdelen beskriver et sett av interaksjoner mellom tjenestene ved å vise meldingene de utveksler. Quality of Service (QoS) er en del av beskrivelsen og modelleringen av ikke-funksjonelle aspekter til tjenestene som er beskrevet i tjenestebiten. For hver av disse delene er det laget en metamodell i PIM4SOA. Metamodellene er vist og forklart i tillegg C.

4.5 Modelltransformasjoner

For å kunne automatisere bruken av modeller på CIM-nivå til modeller på PIM-nivå er modelltransformasjon et viktig hjelpemiddel. Å transformere en modell vil si å lage en ny modell på bakgrunn av en modell i henhold til gitte transformasjonsregler. Modelltransformasjoner kan gjøres på flere måter som f.eks. å transformere på bakgrunn av en metamodellmapping eller en modellinstans mapping [43]. Vi skal se på modelltransformasjoner som baserer seg på metamodeller. For å gjøre transformasjon basert på metamodellmapping, er det nødvendig med en metamodell for den modellen som det skal transformeres fra, og en metamodell for den modellen det skal transformeres til. Deretter må det defineres regler for hvordan elementer i kildemetamodellen skal transformeres til elementer i målmetamodellen, og det spesifiseres med et transformasjonsspråk. Jeg har valgt ut to språk å se på, ATL og transformasjonsspråket i XMF-Mosaic.

4.5.1 Atlas Transformation Language (ATL)

ATL er utviklet av ATLAS INRIA [79] og LINA [81] forskningsgruppe. Dette språket er laget som et svar på QVT RFP [24] laget av OMG.

Bakgrunnen for en transformasjon med ATL, er en eller flere kildemodeller, og en eller flere målmodeller. For å kunne gjøre transformasjonen må både kilde- og målmodellen være typet i henhold til sine respektive metamodeller [22]. Transformasjonene defineres på metamodellnivå.

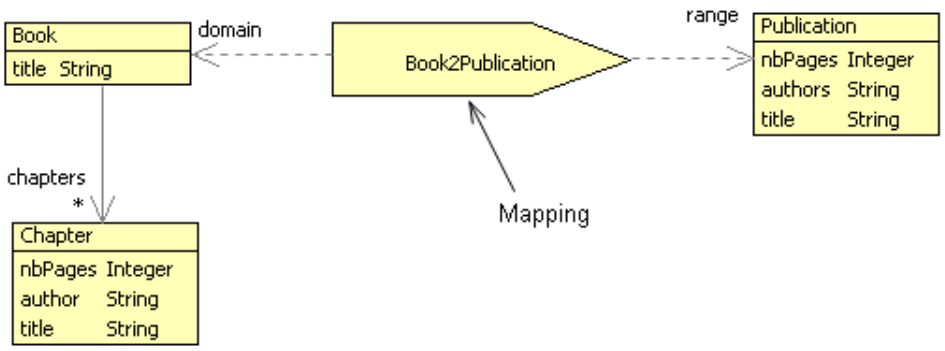
Det har blitt utviklet en Integrated Development Environment (IDE) for ATL som brukes på Eclipse: ATL Development Tools (ADT). ADT bruker Eclipse Modeling Framework (EMF) for å håndtere modeller, da spesielt metamodeller. I tillegg gir ADT et koderedigeringsprogram for å skrive transformasjonskode. ATL kan også brukes i Rational Software Modeler (RSM). Syntaksen i ATL vil bli vist i kapittel 5.

4.5.2 XMF-Mosaic

XMF-Mosaic er en modellbasert utviklingsplattform for å lage verktøy, og er basert på Eclipse. En del av XMF-Mosaic er muligheten for å definere mappinger og gjøre transformasjoner. XMF-Mosaic gir mekanismer for å kunne gjøre modell-til-modell-transformasjon og modell-til-kode-transformasjon. Disse mekanismene er XMap og eXtensible Object Command Language (XOCL).

XMap er et språk for å gjøre modell-til-modell-transformasjon som støtter mønsterbasert mapping av inputmodeller til outputmodeller. Noen XOCL uttrykk brukes som en del av XMap, og XOCL er et språk som ligner på Object Constraint Language (OCL) [14].

I XMF-Mosaic er det mulig å vise modellmappinger grafisk, slik at en kan vise visuelt hvordan elementer i en metamodell mapper til noe i en annen metamodell. I figur 4.7 er det vist et eksempel på en slik mapping. Figuren viser hvordan et bokobjekt, fra bokmetamodellen, vil transformeres til en publikasjon, i publikasjonsmetamodellen. Videre vil en da definere transformasjonsregler for hvordan det skal skje.



Figur 4.7: Book2Publication i XMF-Mosaic

4.6 Oppsummering

I dette kapittelet har vi sett på ulike teknologier for CIM, PIM og modelltransformasjoner. For CIM så vi på ARIS, SBVR, EEML og POP*, for PIM så vi på PRR, COMET og PIM4SOA og modelltransformasjonsspråkene vi så på var ATL og språket i XMF-Mosaic.

ARIS er et komplett virksomhetsmodelleringsrammeverk og kan gjenkjennes ved ARIS-huset, som viser inndelingene i ARIS: organisasjon, data, kontroll, funksjon og output. EEML er et modelleringsspråk som er rettet mot virksomhetsmodellering og det viktigste her er skillet mellom ressursdomenet og prosessdomenet. POP* gir en metamodell for virksomhetsmodellering og er innedelt i flere dimensjoner: prosess, organisasjon og produkt, m.m. SBVR er en standard adoptert i OMG for å utforme forretnings-

vokabular og forretningsregler. Sentralt her er bruken av strukturert naturlig språk, og det at alle begreper og fakta skal defineres. I tillegg så må alle forretningsregler som defineres være støttet opp av faktatyper i forretningsvokabularet.

PRR er en foreslått representasjon for produksjonsregler. Produksjonsregler er regler som er rettet mot oppførsel, men som mulig kan komme ut fra forretningsregler. COMET er en metodikk som gir en komplett veiledning for utvikling av arkitekturmodell. Fokuset i COMET ligger på komponenter. PIM4SOA gir metamodeller for arkitekturmodellen for en tjenestorientert arkitektur.

ATL er et transformasjonsspråk som baserer seg på metamodelltransformasjoner. Det vil si at modellene som transformeres skal være typet med deres metamodeller. XMF-Mosaic er et verktøy som blant annet tilbyr et transformasjonsspråk som kan brukes for å transformere modeller. I XMF-Mosaic er det mulighet for å illustrere modelltransformasjonene grafisk.

Kapittel 5

Evaluering av eksisterende løsninger

I dette kapittelet evalueres de eksisterende løsningene, som ble presentert i kapittel 4. For virksomhetsmodell (CIM) skal ARIS, EEML, POP* og SBVR evalueres, for plattformuavhengig modell (PIM) skal COMET, PRR og PIM4SOA evalueres og for modelltransformasjoner skal ATL og XMF-Mosaic evalueres. Løsningene evalueres i henhold til kravene presentert i kapittel 3. Først skal eksisterende teknikker, metoder og modelleringsspråk for å lage CIM evalueres, deretter PIM og til slutt modelltransformasjonsspråk.

For hvert modelleringsspråk eller metode som evalueres, settes resultatet av evalueringen i en tabell hvor kravene står i den ene kolonnen og resultatet av evalueringen i den andre. Evalueringen gis poengene 0, 1 eller 2. 0 betyr at kravet ikke er oppfylt, 1 betyr at kravet er delvis oppfylt, og 2 at kravet er oppfylt.

5.1 Virksomhetsmodell

Det er mange elementer i en virksomhetsmodell: organisasjonskart, forretningsprosesser, forretningsregler, tjenester, beslutningsstruktur og forretningsressurser. Ikke alle rammeverk og språk skiller mellom disse elementene. Vi har i kravene sett at det er ønskelig å gjøre dette skillet eksplisitt, slik at informasjon og fakta kan lagres som objekter eller data i en database, regler i en regelmotor, prosesser i prosessmotor og tjenestene eller funksjonene til systemet i en egen funksjonsdel som gjør bruk av de forskjellige komponentene.

Til CIM har jeg stilt syv krav. Dette er krav om brukervennlighet, verktøystøtte, informasjonsmodell, organisasjonsmodell, prosessmodell, funksjonalitetsbeskrivelse og forretningsregelmodell.

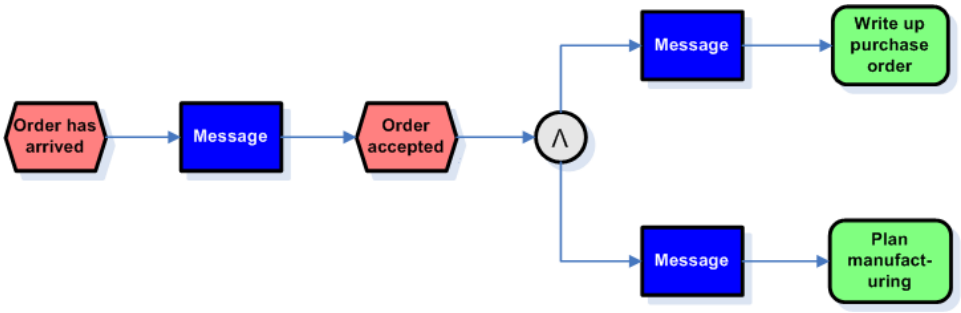
5.1.1 Evaluering av ARIS

ARIS er en metode for virksomhetsmodellering og kan gjenkjennes ved ARIS-huset (se figur 4.3 på side 23) og metodens inndeling i individuelle deler. Delene det deles inn i er funksjon, organisasjon, data, output og kontroll.

I organisasjonsdelen er det mulig å få hele organisasjonsmodellen laget. Dermed oppfyller ARIS krav om organisasjonsmodell. I denne delen er det enkelt å lage organisasjonskart etter det behovet en har. Det kan være detaljert, eller det kan lages på et overordnet nivå.

I output- og datadelen kommer informasjonen i virksomheten klart frem. Outputdelen viser dataobjekter og informasjonstjenester som kan være resultat av en prosess. Hvilken prosess outputen er et resultat av vises i kontrolldelen. Datadelen viser alle dataobjekter som eksisterer i virksomheten.

Prosessmodellen er det som vises i kontrolldelen. Kontrolldelen viser sammenhengen mellom alle de ulike delene av virksomheten. ARIS har god støtte for prosessmodellering og det er bl.a. mulig å bruke logiske operatorer for å få frem prosessene. Figur 5.1 viser et eksempel på en ordrehåndteringsprosess laget med ARIS. *Order has arrived* og *order accepted* er hendelser, *message* er en melding som sendes mellom hendelsene og til funksjonene. Etter at hendelsen *order accepted* har inntruffet vil to funksjoner forekomme, illustrert ved *and*-operatoren. Funkjsonene er *write up purchase order* og *plan manufacturing*.



Figur 5.1: ARIS-prosess [62]

Funksjonalitetsbeskrivelse vises delvis i funksjonsdelen. I funksjonsdelen vises den overordnede beskrivelse av funksjonliteten, men den går ikke i detalj. Det er ønskelig med et visst detaljnivå, slik at spesifiseringen av arkitekturmodellen blir enklere. Funksjoner blir i ARIS sett på som individulle synspunkt på forretningsprosesser [62].

Regler er derimot et element som ikke kommer så tydelig frem ved å modellere med ARIS. Dersom det finnes forretningsregler i bedriften, så tar man bare indirekte hensyn til dem. De blir ikke eksplisitt modellert eller nedtegnet på papir på noen som helst måte.

Totaltsett så liker jeg ARIS på den måten de deler inn modellene sine etter ulike deler fra ARIS-huset. Eksplisitt skille mellom ulike deler av virksomheten gjør det enkelt for oss når vi har tenkt å dele opp systemet i ulike komponenter. Dermed kan hver enkelt transformasjon bli enklere. Selve modelleringsspråket er Event-driven Process Chains (EPC) [60]. Dette er et språk som er enkelt å forstå, og enkelt å bruke. Siden ARIS-metoden er enkel og modelleringsspråket er enkelt, så skårer ARIS høyt på brukervennlighet.

Det finnes eget verktøy for å modellere med ARIS, men dette er ikke basert på standarder som f.eks. XMI o.l. I tillegg er det mulig å bruke Microsoft Office Visio med en EPC-stensil, men dette er bare et tegneverktøy og gir ikke maskinlesbare modeller. Det er ønskelig med en metode som kan anvendes i flere verktøy. Det er mulig å lage en UML-profil for ARIS, men da kan vi risikere et visst informasjonstap. Det kan bli litt vanskelig å bruke ARIS når vi skal gjøre modelltransformasjoner.

I tabell 5.1 er evalueringen av ARIS oppsummert. Bakgrunnen for denne evalueringen er et casestudie som jeg gjennomførte i forbindelse med et essay om virksomhetsmodellering.

Krav til CIM	**ARIS**
CIM1: Brukervennlighet	2
CIM2: Verktøystøtte	1
CIM3: Informasjonsmodell	2
CIM4: Organisasjonsmodell	2
CIM5: Prosessmodell	2
CIM6: Funksjonalitetsbeskrivelse	1
CIM7: Forretningsregelmodell	0

Tabell 5.1: Evaluering av ARIS som metode for å lage CIM

5.1.2 Evaluering av EEML

EEML er et enkelt språk med gode symboler for å illustrere ulike elementer i virksomheten. EEML har mange symboler, men allikevel enkle symboler. Dette gjør det enkelt for personer uten mye erfaring med språket, å lett kunne forstå modellene. En hammer for å illustrere verktøy er et symbol som er lett forståelig for allmennheten, det samme er symbolet som minner om en person, som da også symboliserer en person. EEML er med andre ord meget brukervennlig.

EEML er et språk som er laget for et verktøy som heter Metis. Dette er et verktøy utviklet av Computas (som nå er gått over til Troux). Verktøyet er et godt verktøy for virksomhetsmodellering, fordi det gjør det enkelt å få oversikt over tunge komplekse modeller. I Metis er det mulig å zoome inn og ut av prosesser, oppgaver o.l., slik at en lett kan få et overordnet perspektiv på modellene. Et meget godt verktøy når man har å gjøre med store komplekse virksomhetsmodeller. I og med at EEML er et språk

med kun et verktøy som støtter språket, er det vanskelig å bruke disse modellene videre med modelltransformasjoner. I tillegg er dette verktøyet dyrt. Derfor velger jeg å gi EEML 1 for verktøystøtte.

Informasjonsmodell, organisasjonsmodell og prosessmodell er de tre elementene som kommer best frem i EEML. Ressursdomenet gjør det enkelt å modellere alle tenkelige ressurser som eksisterer i bedriften, både informasjonsmateriale, softwarekomponenter og personer. Prosessmodellen kommer frem av prosessdomenet, og prosessene kan enkelt knyttes opp mot ressurser. EEML er dynamisk, ved at vi underveis kan forandre på tilstandene til oppgavene o.l., men samtidig så fokuserer EEML mer på organisering av virksomheten og hvor prosessene kommer inn, ikke like mye på prosessene i seg selv.

Funksjonalitetsbeskrivelse og forretningsregelmodell er to krav EEML ikke oppfyller. EEML er laget med den hensikt å modellere virksomheten, ikke å modellere virksomheten med tanke på informasjonssystemet som en ønsker å spesifisere. Forretningsregler er det ingen mulighet for å skille ut, her får man kun fram regler indirekte gjennom prosesser, men langt i fra alle ønskelige regler. I tabell 5.2 er evalueringen oppsummert, og her har jeg da satt 0 for CIM6 og CIM7.

Krav til CIM	**EEML**
CIM1: Brukervennlighet	2
CIM2: Verktøystøtte	1
CIM3: Informasjonsmodell	2
CIM4: Organisasjonsmodell	2
CIM5: Prosessmodell	2
CIM6: Funksjonalitetsbeskrivelse	0
CIM7: Forretningsregelmodell	0

Tabell 5.2: Evaluering av EEML som rammeverk for CIM

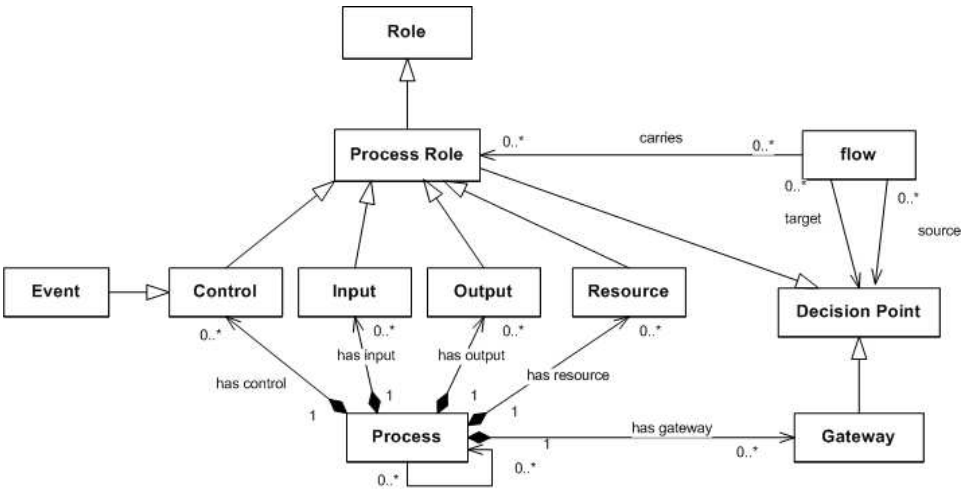
5.1.3 Evaluering av POP*

POP* er en metodikk laget for å kunne utveksle virksomhetsmodeller mellom bedrifter. Jeg ønsker å evaluere POP* metamodellen som metode for å lage en virksomhetsmodell. Evalueringen er gjort ved modellering av casen (casebeskrivelse er i tillegg D).

POP* metamodell

POP* har fem dimensjoner (prosess, organisasjon, produkt, beslutning og IT-infrastruktur). For hver dimensjon er det laget en metamodell. I tillegg er det laget en metamodell for generelle konsepter, som alle dimensjonene også bruker.

Prosessdimensjonen gir konseptene *Process Role*, *Control*, *Input*, *Output*, *Resource*, *Event*, *Process*, *Gateway*, *Decision Point* og *Flow*. Dette er elementer som kan relateres til aktiviteter, oppgaver og prosesser som pågår i virksomheten. Når ulike virksomhetsobjekter deltar i en prosess, beskrives dette med ulike typer prosessroller. Selve logikken i prosessen uttrykkes med flyt og beslutningspunkt. Prosessdimensjonen er veldig viktig i beskrivelsen av virksomheten fordi den gir en forståelse av hvordan virksomheten fungerer: f.eks. hva som skjer når en gitt hendelse inntreffer, eller hva slags beslutninger som tas i bestemte situasjoner. I denne dimensjonen kan vi dermed danne oss et helhetlig bilde av virksomheten. Prosessen kan modelleres ved å bruke UML 2.0 aktivitetsdiagram med stereotyper definert fra denne metamodellen. Figur 5.2 viser prosessdimensjonen.



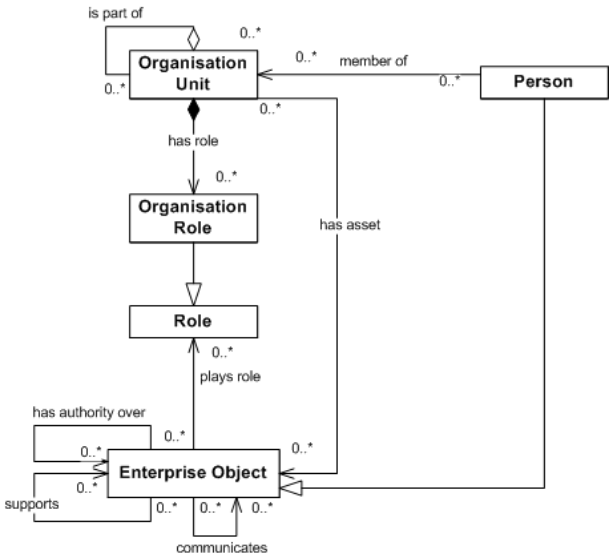
Figur 5.2: Prosessdimensjonen i POP* [5]

Figur 5.3 på neste side viser organisasjonsdimensjonen. Her er det meningen å beskrive organisasjonsstrukturen i virksomheten. Her kan både struktur, medlemmer i organisasjonen og hvilke posisjoner medlemmene har vises. Konseptene som definerers her er *Organisation Unit*, *Person* og *Organisation Role*. Denne dimensjonen lages med UML 2.0 klassediagram med tilhørende stereotyper.

Produktdimensjonen brukes for å modellere produktarkitekturer og produktstrukturer for å kunne vise design, utvikling, datahåndtering og andre elementer. Produktdimensjonen er beskrevet mer i detalj i tillegg C. De to siste dimensjonene er beslutningsdimensjonen og dimensjonen for IT-infrastruktur. Disse er ikke ansett som relevante og jeg refererer til beskrivelsen av metamodellen i [5] for detaljer om disse dimensjonene.

Evaluering

Det er laget en UML-profil for POP* som kan brukes for å lage virksomhetsmodeller. Denne profilen er laget i henhold til metamodellen beskrevet



Figur 5.3: Organisasjonsdimensjonen i POP* [5]

over og i tillegg C. Profilen er allsidig ettersom den dekker alle de elementene som finnes i de ledende metoder og modelleringsspråk for virksomhetsmodellering.

Etter å ha modellert store deler av Demo Telco casen [36] med POP* har jeg sett at POP* metodikken dekker mye. I prosessdimensjonen får man beskrevet prosessene i DemoTelco på en utfyllende måte. Ved å bruke stereotypene er det enkelt å detaljere en prosess med tanke på input, output og ressurser, som er viktige bestanddeler i en prosess. Bruken av UML 2.0 aktivitetsdiagram gjør det dessuten enkelt å vise hvordan prosessen f.eks. går på tvers av flere avdelinger eller forretningsinteressenter.

Organisasjonsdimensjonen viser organisasjonsstrukturen på en enkel og oversiktlig måte ved å bruke stereotypene og UML 2.0 klassediagram. Her har en mulighet for å modellere alle aspekter i en organsiasjon, eller bare deler av den, alt etter hva det er behov for.

Produktdimensjonen kan brukes for å vise litt av informasjonen i virksomheten og noe funksjonalitetsbeskrivelse. Funksjonalitetsbeskrivelsen er da i forhold til hva slags tjenester eller produkter bedriften tilbyr. Jeg synes denne dimensjonen er litt vag, i og med at den kan brukes til å vise både informasjon og funksjonalitet. Informasjon kan en få også dersom en tar med dimensjonen for IT-infrastruktur, så derfor gir jeg POP* 1 for støtte for informasjonsmodell, og også 1 for funksjonalitetsbeskrivelse. Det burde vært et bedre skille på disse to elementene. Funksjonalitetsbeskrivelsen er tilstede, men på en meget svak måte.

Verktøystøtten for POP* er bra, i og med at det er laget en UML-profil for den. UML-profilen jeg har er laget for RSM og Rational Software Architect (RSA), og den kan enkelt implementeres i andre verktøy som f.eks. Eclipse. Brukervennligheten er derimot ikke så bra. Forretningsfolk

vil ofte være med i prosessen for å lage virksomhetsmodeller, og det er viktig at modelleringsspråket er enkelt. UML-profilen er enkel å forstå, men det tar litt tid å sette seg inn i de ulike delene av POP*, som f.eks. hva som skal brukes når, og hvordan modellere ulike elementer i en virksomhetsmodell.

Tabell 5.3 oppsummerer evalueringen. Forretningsregler er det ikke støtte for i POP* og det ser vi igjen som 0 i oppsummeringen.

Krav til CIM	**POP***
CIM1: Brukervennlighet	1
CIM2: Verktøystøtte	2
CIM3: Informasjonsmodell	1
CIM4: Organisasjonsmodell	2
CIM5: Prosessmodell	2
CIM6: Funksjonalitetsbeskrivelse	1
CIM7: Forretningsregelmodell	0

Tabell 5.3: Evaluering av POP* som rammeverk for CIM

5.1.4 Evaluering av SBVR

For å evaluere SBVR har jeg anvendt det på DemoTelco casen [36]. I første omgang ble det gjort på papir, deretter anvendte jeg URM.

SBVR gir en meget detaljert fremgangsmåte for å identifisere forretningsregler. For å komme frem til ulike forretningsregler, må konteksten til virksomheten først identifiseres. Her skal forretningsvokabularet beskrives og også en organisasjonsstruktur i bedriften komme frem. Forretningsvokabularet blir veldig detaljert, med en definisjon for hvert ord og forklaring for hvert uttrykk.

Det mest positive med SBVR er nettopp denne detaljeringen. Det er mulig å få identifisert smådetaljer som er av betydning for reglene som eksisterer i virksomheten. Dette gir en god start når man skal starte en systemutviklingsprosess fordi alle begreper blir definert på et tidlig stadium. På den måten er det lettere å unngå misforståelser som ofte kan oppstå senere i en utviklingsprosess. Når man modellerer, så har disse begrepene mye å si for hvordan modellene skal bli. Dersom det f.eks. er en gruppe av mennesker som sammen skal utvikle modellene, så har kanskje personene ulike oppfattelser av begreper. Med en tidlig definisjon av begrepene er det mulig å unngå høylytte diskusjoner rundt modellene.

Detaljeringen i SBVR har også noen negative sider. For mye detaljer fører gjerne til at det blir for mye å holde oversikt over. Når alle fakta defineres med naturlig språk, gjelder det å holde tunga rett i munnen. Det at *person har navn* er lettere å uttrykke i en modell, f.eks. et klassediagram som viser oversikten over alle begrep og fakta i bedriften. SBVR foreslår en UML representasjon for forretningsvokabularet, men etter hva jeg forstår så er

det ønskelig at man gjør alt i tekst. URM har en del elementære ting som en standard del av hvert prosjekt, og det gjør det litt enklere å definere vokabularet, men samtidig så har URM en så streng syntaks, at det er vanskelig å lage hele vokabularet og reglene slik at alt er verifiserbart og i henhold til den bestemte syntaksen.

Brukervennligheten er ikke den beste i SBVR. Det er lett å forstå, siden man anvender naturlig språk, men det å gjøre alt i henhold til metamodellen er vanskelig. Det er mye som skal defineres før en faktisk kan skrive en forretningsregel. Jeg gir derfor kravet om brukervennlighet 1. Det er verktøystøtte for SBVR (Unisys Rules Modeler), men verktøyet er fremdeles på et tidlig stadium, og dermed ikke særlig bra. Det gir dårlige tilbakemeldinger når en får syntaks- eller verifiseringsfeil. En får tilbakemelding, men de er vanskelige å forstå. Det positive med URM er dets støtte for å eksportere og importere vokabular og regler med XMI. Dette kan gjøre det enkelt for f.eks. å eksportere inn vokabular til andre verktøy i form av UML-modeller. Denne eksporterings- og importeringsfunksjonaliteten i verktøyet er per i dag dårlig utviklet, og genererer ikke XMI-filer i henhold til metamodellen spesifisert i SBVR, men er forventet å gjøre det i neste versjon.

Organisasjonsmodellen kommer delvis frem i SBVR. Det er foreslått å vise organisasjonen med et UML-lignende klassediagram, men da med definisjoner i tillegg. Organisasjonen defineres nok litt mer avansert i SBVR enn nødvendig. For hver del skal det defineres hvilket språksamfunn det er, om det er et *community* eller *subcommunity*. Dette blir litt for detaljert, og organisasjonsmodellen mister noe av sin hensikt.

Informasjonsmodellen kommer tydelig frem i SBVR. Dette er i SBVR betegnet som forretningsvokabular. Som nevnt, så er det litt tungvindt å lage hele forretningsvokabularet med vanlig tekstlig beskrivelse, men dersom UML-klassediagram som foreslått i SBVR brukes i tillegg, blir forretningsvokabularet veldig bra. Kravet om informasjonsmodell gir jeg derfor 2.

Funksjonalitetsbeskrivelse er helt fraværende fra SBVR, fordi SBVR kun fokuserer på virksomheten. SBVR er laget med målet om å beskrive virksomheten, den fokuserer ikke på IT-systemet i det hele tatt. SBVR kan brukes som input til en systemutviklingsprosess.

Forretningsregler er SBVR gode på, men jeg vil ikke gi mer enn 1 på dette kravet. Dette er fordi det er vanskelig å lage en regel som er riktig støttet opp av fakta og begreper, og regelbegrepet i SBVR er litt for bredt. Verifisering i URM er vanskelig, det er for høyt krav til samsvar mellom fakta og regler. Forretningsreglene er foreløpig en del av virksomhetsmodellen, og da bør det ikke være et så stort fokus på teknisk korrekthet. Da blir det for komplisert til at forretningsfolk kan være med på denne utviklingen.

Flere av reglene som defineres i SBVR er prosesser. Det vil si at det vi modellerer i en prosessmodell også kan uttrykkes som regler i SBVR. Et

eksempel på en slik regel er "Når en ordre er mottatt, må det være tilfellet at den sendes videre til salgsavdelingen". En slik regel er bedre å uttrykke med f.eks. et UML-aktivitetsdiagram. I og med at det er flere etablerte modelleringsteknikker for å modellere prosesser, vil jeg si at den delen av SBVR er unødvendig. Det som bør analyseres er hvilke deler av SBVR vi ikke får uttrykket med de andre modelleringsteknikkene. De delene av SBVR vil være verdt å ta med seg videre. Hvilke deler av SBVR som bør tas med blir diskutert nærmere i kapittel 6 og 7.

Evalueringen er oppsummert i tabell 5.4.

Krav til CIM	**SBVR**
CIM1: Brukervennlighet	1
CIM2: Verktøystøtte	1
CIM3: Informasjonsmodell	2
CIM4: Organisasjonsmodell	1
CIM5: Prosessmodell	0
CIM6: Funksjonalitetsbeskrivelse	0
CIM7: Forretningsregelmodell	1

Tabell 5.4: Evaluering av SBVR som rammeverk for CIM

5.2 Plattformuavhengig modell

For PIM stilte jeg i kapittel 3 åtte krav. Dette er krav om komponentstruktur, komponentinteraksjon, funksjonsmodell, informasjonsmodell, regelmodell, prosessmodell, interfacespesifisering og verktøystøtte. PRR, PIM4SOA og COMET skal her evalueres i henhold til disse kravene.

5.2.1 Evaluering av PRR

PRR er ikke en tilnærming eller metode for å lage hele den plattformuavhengige modellen, men forventes kun å brukes for å spesifisere regelkomponenten i en PIM. PRR er foreløpig den eneste standarden for regelspesifisering i en PIM som også har tenkt på å kunne støtte opp mot andre modellelementer i en PIM. PRR har også sett på mappingmuligheter mot CIM og PSM. Denne får derfor helt klart 2 for kravet om regelmodell (PIM5).

Er det verktøystøtte for PRR? Svaret er både ja og nei. Det er enkelt å lage en UML-profil for PRR i henhold til den metamodellen som PRR gir, dersom en selv angir på hvilke UML-elementer stereotyper skal settes. En UML-profil kan benyttes i mange ulike programmer. I og med at det foreløpig ikke er laget noen UML-profil for PRR, er det noe uklart om hvordan UML-profilen bør lages for best mulig å modellere det som PRR er tenkt for. Derfor gir jeg PRR 1 for verktøystøtte. Tabell 5.5 oppsummerer evalueringen.

Krav til PIM	PRR
PIM1: Komponentstruktur	0
PIM2: Komponentinteraksjon	0
PIM3: Funksjonsmodell	0
PIM4: Informasjonsmodell	0
PIM5: Regelmodell	2
PIM6: Prosessmodell	0
PIM7: Interfacespesifisering	0
PIM8: Verktøystøtte	1

Tabell 5.5: Evaluering av PRR

5.2.2 Evaluering av PIM4SOA

PIM4SOA gir ingen veiledning på hvordan modellene skal tegnes, den gir kun en metamodell for hvordan de ulike delene skal representeres. Komponentstruktur, komponentinteraksjonsmodell og interfacespesifisering er derfor alle elementer som jeg gir 0.

PIM4SOA er en av få metodikker for PIM som tar for seg prosesser som en egen del. Dermed støtter det bedre opp for å implementere forretningsprosesser i en prosessmotor som f.eks. Business Process Execution Language (BPEL). ATHENA ser på hvordan en kan transformere prosessmodellen fra POP* til prosessmodellen i PIM4SOA, slik at en kan utnytte virksomhetsmodellen i informasjonssystemet man implementerer. På grunn av støtten for spesifisering av prosesser i PIM4SOA gir jeg den 2 for kravet om prosessmodell.

Tjenesteorientert arkitektur er det PIM4SOA er laget for, og derfor er det naturlig at tjenester er skilt ut som en egen del her. Det er ikke gitt at alle IT-systemer vil benytte seg av dette, men det vil være greit å ha det som en inspirasjon, i og med at det er mange komponenter som skal samspille. Service Oriented Architecture (SOA) er en løs integrasjonsteknologi, men også tette integrasjonsteknologier kan utnytte tankemåten i SOA. Derfor har jeg gitt PIM4SOA 2 i kravet for en funksjonsmodell.

Informasjon er også noe som er godt støttet i PIM4SOA gjennom *information model*. Noen av konseptene her kan nok utvides med vanlige konsepter fra UML 2.0, men ellers er det bra. Kravet om informasjonsmodell er tydelig oppfylt.

Regler er ikke skilt ut som en egen del i PIM4SOA, og er også den største mangelen. Selv om den får 0 for noen krav, så er de kravene ting som lett oppfylles av andre metoder som gir en tilnærming til utvikling av en arkitekturmodell. Det er et stort forbedringspotensiale for PIM4SOA, og å utvide med en regelkomponent er en av dem.

PIM4SOA er også noe som lett kan støttes i verktøy ved å lage en UML-profil i henhold til metamodellen definert for PIM4SOA. Dette vil nok bli gjort i ATHENA ganske snart. Det betyr at PIM4SOA har delvis

verktøystøtte i og med at det er lett å lage modeller med stereotyper som illustrer elementene. Tabell 5.6 oppsummerer evalueringen.

Krav til PIM	PIM4SOA
PIM1: Komponentstruktur	0
PIM2: Komponentinteraksjon	0
PIM3: Funksjonsmodell	2
PIM4: Informasjonsmodell	2
PIM5: Regelmodell	0
PIM6: Prosessmodell	2
PIM7: Interfacespesifisering	0
PIM8: Verktøystøtte	1

Tabell 5.6: Evaluering av PIM4SOA

5.2.3 Evaluering av COMET for arkitekturmodell

COMET er en omfattende metodikk som gir forslag til løsning og tilnærming både for CIM, PIM og PSM, men i COMET er det delt opp i *business model*, *requirements model*, *architecture model* og *plattform specific model*. Vi skal kun vurdere denne for PIM-nivå, det vil si *architecture model*.

COMET gir en generell tilnærming for PIM. Det er derfor ikke angitt spesifikke komponenter som skal være med i en arkitekturmodell, men gjennom tilnærmingen får vi vist noen av de komponentmodellene som jeg har stilt krav om.

Komponentstruktur vises tydelig. Hele COMET er en komponentbasert metodikk, og har hele tiden fokus på å definere komponenter i et system. Her kommer også komponentinterasjonsmodellen klart frem. Denne må være med for å definere hvordan komponentene bruker hverandre.

I interfacespesifiseringen skal alle interface for komponentene vises. Klassediagram brukes for å lage interfacemodell. Etter å ha laget interfacemodellen, er det enkelt å se hva som trengs av informasjon i systemet, og ut fra det lage en informasjonsmodell. COMET får dermed 2 både for interfacespesifisering og for informasjonsmodell.

Funksjoner og prosesser er ikke skilt ut som en egen bit i COMET. Noe funksjonalitet i systemet fremgår gjennom andre modeller her, som f.eks. komponentstruktur og komponentinteraksjon. Krav om funksjonsmodell får derfor 1. Prosessmodell er det ingen direkte støtte for. En lager modeller som kan minne om prosessmodeller, men disse er ikke linket opp mot forretningsprosessene, som er et ønske. Kravet om prosessmodell får derfor 0. Kravet om regler får også 0, fordi regler er det ikke fokusert på i det hele tatt i COMET. Evalueringen er oppsummert i tabell 5.7.

Krav til PIM	COMET
PIM1: Komponentstruktur	2
PIM2: Komponentinteraksjon	2
PIM3: Funksjonsmodell	1
PIM4: Informasjonsmodell	2
PIM5: Regelmodell	0
PIM6: Prosessmodell	0
PIM7: Interfacespesifisering	2
PIM8: Verktøystøtte	2

Tabell 5.7: Evaluering av COMET

5.3 Modelltransformasjon

Utgangspunktet for evalueringen er to elementer: ATL og XMF-Mosaic. Det finnes også andre transformasjonsspråk som kunne vært evaluert, men disse er valgt ut på grunn av deres forhold til standarder fra OMG og fordi det samtidig er en del forskjeller på dem. Evaluering av ATL er gjort på bakgrunn av bruk av ATL i RSM.

5.3.1 Evaluering av ATL

En ATL-modul definerer modell-til-modell-transformasjoner. En slik modul består av 4 elementer. Det første er en header. Headeren definerer de attributtene som er relevante for transformasjonen. Etter headeren er det en valgfri import-seksjon. Her kan det spesifiseres hvilke ATL-bibliotek vi ønsker å benytte oss av i ATL-modulen. Deretter kommer det et sett av hjelpere. Hjelpere er noe tilsvarende som metoder i Java. Til slutt i ATL-modulen kommer reglene som definerer hvordan målmodellen blir generert fra kilden [22]. I figur 5.4 på neste side ser vi et eksempel på en ATL-modul. Der er det markert hva som er *header*, *helper* og *rule*. Eksempelet i figur 5.4 baserer seg på kilde- og målmetamodellen som er vist i figur 5.5 på neste side.

Syntaksen som brukes for å lage hjelpere og regler er basert på OCL [14]. Dette er en ganske komprimert syntaks og ganske forskjellig fra objektorienterte språk som f.eks. Java. Fordelen med denne syntaksen er at en ikke trenger å skrive så mye for å få transformasjonsjobben gjort. Nærmere detaljer om syntaks er beskrevet i brukermanualen [22].

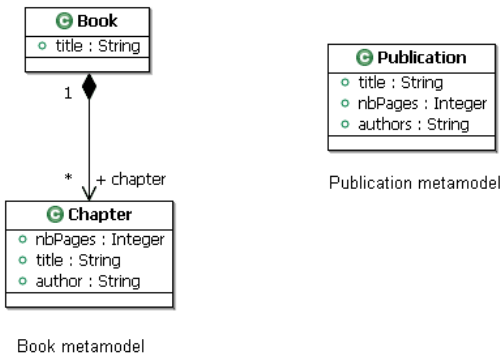
ATL er et kraftfullt språk. Det er mulighet for å uttrykke mye på få linjer. Jeg har anvendt ATL i RSM, og erfaringene derfra er at det er mye detaljer som må på plass før transformasjonen faktisk kan kjøres. Metamodellene lages grafisk, deretter må den eksporteres som .ecore, og etterpå importeres i RSM for å kunne brukes som kilde- eller målmodell. Selve modellen som skal transformeres må derimot i RSM defineres tekstlig, noe som gjør det veldig komplekst å bruke for større prosjekter. Det tar altfor lang tid å

```
module Book2Publication;
create OUT : Publication from IN : Book;

helper context Book!Book def : getAuthors() : String =
       self.chapters->collect(e | e.author)->
                      asSet()->
                      iterate(authorName; acc : String = '' |
                              acc +
                              if acc = ''
                                  then authorName
                                  else ' and ' + authorName
                              endif)
;
helper context Book!Book def : getNbPages() : Integer =
        self.chapters->collect(f|f.nbPages)->
                      iterate(pages; acc : Integer = 0 |
                      acc + pages)
;
helper context Book!Book def : getSumPages() : Integer =
        self.chapters->collect(f|f.nbPages).sum()
;

rule Book2Publication {
    from
        b : Book!Book (
            b.getSumPages() > 2
        )
    to
        out : Publication!Publication (
            title <- b.title,
            authors <- b.getAuthors(),
            nbPages <- b.getSumPages()
        )
}
```

Figur 5.4: ATL kode for Book2Publication



Figur 5.5: Metamodeller for Book (kilde) og Publication (mål)

oversette alle modellene til tekst, for så å transformere teksten. Da mister modelltransformasjonen litt av poenget, nemlig å gjøre det enklere å bruke modeller mer aktivt i spesifikasjonen av IT-systemer. På grunn av dette gir jeg ATL 1 for *forståelighet*. ATL er godt dokumentert.

Det som er veldig bra med ATL er at det er metamodellbasert, det er godt støttet opp i verktøy og det forholder seg til QVT-standarden foreslått av OMG. ATL ivaretar også informasjon, men pga at forståeligheten ikke er den beste, er det ikke så enkelt å spesifisere reglene og transformere på korrekt måte, og da er det heller ikke gitt at informasjonen ivaretas på ønskelig måte. Det samme gjelder også sporbarhet. I tabell 5.8 er evalueringen oppsummert.

Krav til transformasjonene	ATL
T1: Metamodellbasert	2
T2: Forståelighet	1
T3: Ivareta informasjon	1
T4: Sporbarhet	1
T5: Verktøystøtte	2
T6: Standardisert	2

Tabell 5.8: Evaluering av ATL for modelltransformasjoner

5.3.2 Evaluering av XMF-Mosaic

XMF-Mosaic er et mer visuelt verktøy enn det ATL er. Her er det mulighet for å vise i modeller hvilke elementer i en metamodell som skal mappes til hvilke elementer i en annen metamodell. Alle transformasjoner er metamodellbaserte. Ved å gjøre det på en slik visuell måte er det enkelt å forstå. Riktignok er syntaksen lignende ATL-språket og kan være litt komplisert å lære seg, men det visuelle veier opp for kompleks syntaks.

Det å ivareta informasjon er litt usikkert. I XMF-Mosaic er det ikke noen mulighet for å vise aggregering eller komposisjon, og da taper en litt informasjon dersom en tegner inn en metamodell som inneholder noen av de nevnte relasjonene. Informasjonen man transformerer ivaretas så lenge man definerer gode transformasjonsregler. Jeg gir XMF-Mosaic 1 for kravet om å ivareta informasjon. Det samme gjelder for sporbarhet.

XMF-Mosaic er et verktøy, og dermed er selvsagt verktøystøtten god. XOCL og XMap som er det XMF-Mosaic bruker som betegnelse for språket er noe i henhold til standarden QVT, men ikke i like stor grad som det er ønskelig.

Totaltsett er XMF-Mosaic et godt egnet program for modelltransformasjoner, men det er noe vanskelig å bruke det sammen med f.eks RSM. Det har støtte for å importere XMI, men det har ikke jeg fått til å fungere. Dersom det hadde gått an å importere og eksportere modellene i henhold til UML-standarder, ville XMF-Mosaic vært enda bedre. Det må også nevnes

at dette programmet er nyutviklet og mangler derfor en del nyttig funksjonalitet. Dette er ting som angrefunksjon, hurtigtaster for lagring, automatisk lagring av prosjektene når man avslutter og at prosjektene blir husket fra gang til gang, istedet for å måtte åpne dem hver gang en starter programmet. Evalueringen er oppsummert i tabell 5.9.

Krav til transformasjonene	XMF-Mosaic
T1: Metamodellbasert	2
T2: Forståelighet	2
T3: Ivareta informasjon	1
T4: Sporbarhet	1
T5: Verktøystøtte	2
T6: Standardisert	1

Tabell 5.9: Evaluering av XMF-Mosaic for modelltransformasjoner

5.4 Oppsummering av evaluering

5.4.1 Virksomhetsmodell

Metodene som er blitt evaluerte har alle vist seg å ha positive elementer. I tabell 5.10 er metodene som ble evaluerte for CIM oppsummert.

Krav til CIM	ARIS	EEML	POP*	SBVR
CIM1: Brukervennlighet	2	2	1	1
CIM2: Verktøystøtte	1	1	2	1
CIM3: Informasjonsmodell	2	2	1	2
CIM4: Organisasjonsmodell	2	2	2	1
CIM5: Prosessmodell	2	2	2	0
CIM6: Funksjonalitetsbeskrivelse	1	0	1	0
CIM7: Forretningsregelmodell	0	0	0	1
SUM	10	9	9	6

Tabell 5.10: Evaluering av CIM

ARIS er den metoden som kommer best ut i henhold til kriteriene som ble stilt i kapittel 3. Den gir et klart skille mellom ulike elementer som hører hjemme i en virksomhetsmodell, og samtidig viser godt sammenhengen mellom disse elementene i prosessmodellen. Den største mangelen her er representasjon av forretningsregler.

EEML og POP* har begge fått 9 poeng (av 14 mulige). EEML scorer høyt på brukervennlighet, informasjonsmodell, organisasjonsmodell og prosessmodell. POP* scorer høyt på verktøystøtte, organisasjonsmodell og prosessmodell. Likt for begge er mangelen på en egen forretningsregelmodell.

SBVR er den som kommer dårligst ut av evalueringen, men samtidig den eneste som støtter forretningsregler. SBVR er laget for å gi en klar

semantikk for forretningsregler, men den har blitt litt vel komplisert. Derfor har den bare fått 1 for forretningsregelmodell. URM er det eneste verktøyet for SBVR, og her er det for stort krav til samsvar mellom vakabularet og reglene. Det krever for mye tid å få så strengt samsvar på plass. SBVR er dessuten bare tekstlig, og det hadde vært fint om noen regler kunne vises grafisk.

En generell observasjon for alle er mangelen på god verktøystøtte og mangel på støtte for forretningsregler, til og med SBVR. Det er et behov for en metode som oppfyller alle krav. Den største utfordringen vil være å representere forretningsregler, i og med at det er færrest metoder som har dette oppfylt fra før av.

5.4.2 Plattformuavhengig modell

Metoder for å lage arkitekturmodeller er det mange av, men her er det kun valgt ut tre: PRR, PIM4SOA og COMET. I tabell 5.11 er evalueringen av dem oppsummert.

Krav til PIM	**PRR**	**PIM4SOA**	**COMET**
PIM1: Komponentstruktur	0	0	2
PIM2: Komponentinteraksjon	0	0	2
PIM3: Funksjonsmodell	0	2	1
PIM4: Informasjonsmodell	0	2	2
PIM5: Regelmodell	2	0	0
PIM6: Prosessmodell	0	2	0
PIM7: Interfacespesifisering	0	0	2
PIM8: Verktøystøtte	1	1	2
SUM	3	7	11

Tabell 5.11: Evaluering av PIM

Naturlig nok kommer PRR dårligst ut av evaluering, i og med at dette er en metode eller rettere sagt et forslag for hvordan man kan representere produksjonsregler. Det den får poeng for er for PIM5 og PIM8.

Metoden som kommer best ut av evalueringen er COMET. COMET er en komplett metodikk for utvikling av både CIM, PIM og PSM, men det er kun PIM-delen av COMET som er blitt evaluert. COMET gir en bra tilnærming til hvordan en skal spesifisere komponentene og deres sammenheng, men mangler elementer som prosessmodell og regelemodell.

PIM4SOA består hovedsakelig av en metamodell som sier noe om hvilke deler som skal være med i en plattformuavhengig modell. Metamodellen inneholder god støtte for prosessmodell, informasjonsmodell og for å representere tjenester.

Disse tre som her er evaluert utfyller hverandre. Det som en metode ikke er god på, er en annen god på. Det vil derfor være naturlig å plukke ut de positive elementene fra hver metode og sette dem sammen.

5.4.3 Modelltransformasjon

Grunnlaget for evaluering av modelltransformasjoner har vært ATL og XMF-Mosaic. Begge har vist seg å være gode verktøy. Tabell 5.12 oppsummerer evalueringen gjort av ATL og XMF-Mosaic.

Krav til transformasjonene	**XMF-Mosaic**	**ATL**
T1: Metamodellbasert	2	2
T2: Forståelighet	2	1
T3: Ivareta informasjon	1	1
T4: Sporbarhet	1	1
T5: Verktøystøtte	2	2
T6: Standardisert	1	2
SUM	9	9

Tabell 5.12: Evaluering av modelltransformasjoner

XMF-Mosaic er litt mer forståelig enn ATL. I dette ligger det at XMF-Mosaic er mer grafisk, det er her gode muligheter for å visualisere transformasjonene. Det gjør det igjen enkelt å vise hvordan en transformasjon skal foregå, og vil være godt egnet for å forklare tankene bak ulike modelltransformasjoner. ATL forholder seg mer til standarder utstedet av OMG enn det XMF-Mosaic gjør.

Kapittel 6

Krav til transformasjon av regelmodeller

Dette kapittelet vil presentere kravene for metoden. Etter å ha evaluert eksisterende løsninger, kan vi se at det er en mangel når det gjelder forretningsregler. Vi har sett at PRR er mulig å bruke for PIM, men dette må analyseres nærmere. SBVR er laget for å formulere forretningsregler, men den er ikke egnet for modelltransformasjon slik som er ønskelig. Derfor skal vi i dette kapittelet se nærmere på forretningsregler og detaljere spesifikasjonen av dem. Dette kapittelet introduserer nye krav som kommer i tillegg til kravene i kapittel 3, og alle kravene danner grunnlaget for metoden. Selve metoden presenteres i neste kapittel.

6.1 Nye krav til virksomhetsmodeller

Kravene definert for evalueringen av eksisterende løsninger (kapittel 3) danner også grunnlaget for metoden. Metoden skal kunne brukes for å anvende virksomhetsmodeller for å spesifisere IT-systemer, og da må vi ta hensyn til andre ting enn bare forretningsregler. Forretningsregler er, som vi så i evalueringen, det elementet som ble dårligst behandlet. Det vi skal finne ut er om forretningsregler kan skilles ut som et element i virksomhetsmodellen, transformere dette til input til PIM, og derfra videre til en spesifikk regelemotor. Dette krever at vi stiller krav til forretningsregelmodellen, og derfor får vi nye krav til CIM.

I kapittel 4 og 5 så vi at det finnes noe som heter PRR. PRR er laget for å representere forretningsregler på PIM-nivå. Reglene kalles for produksjonsregler. I metoden skal vi transformere forretningsregler til produksjonsregler, siden produksjonsregler ser ut til å være en fornuftig måte å representere regler i PIM på. Produksjonsreglene er utviklet med den hensikt at de kan transformeres til regelmotorer, og dette kan vi forhåpentligvis utnytte.

Før kravene som stilles for forretningsregler i metoden presenteres, skal vi detaljere spesifikasjonen av forretningsregler. Denne spesifikasjonen gir igjen grunnlag for å komme frem til kravene for forretningsreglene. Vi skal se hvordan forretningsregler kan kategoriseres, og hvilke kategorier som bør være med i forretningsregelmodellen i CIM. Etter det skal vi se på muligheter for å visualisere forretningsregler, og til slutt vil kravene presenteres.

6.2 Kategorisering av forretningsregler

Når en hendelse i virksomheten finner sted vil en forretningsperson gjerne legge til regler for å kontrollere eksekveringen av hendelsen. Det er da fire mulige måter for en regel å veilede en hendelse i virksomheten [70]:

- Regelen presenterer informasjon om forretningshendelsen
- Regelen styrer informasjon som lages av hendelsen
- Regelen initierer en hendelse utenfor grensene til målsystemet eller forretningshendelsen
- Regelen lager ny informasjon fra eksisterende informasjon

Disse fire punktene indikerer hva som er regelens intensjon sett fra en forretningspersons synspunkt. I "Business Rules and Information Systems" [44] vises det at en forretningsregel har å gjøre med fire forskjellige aspekter av bedriften: konsistens av informasjon, relasjoner mellom entiteter, identifisering av situasjoner og dataintegritet. Her vektlegges skillet mellom regler og prosesser. En prosess beskriver når regelen eksekveres, hvor regelen eksekveres og hvordan regelen skal implementeres i endelig design. Forretningsregelen i seg selv definerer hva som bør være tilfellet.

Jeg har sett på mange ulike kategoriseringer av forretningsregler, og deres definisjoner, og på bakgrunn av det kommet frem til fire kategorier som jeg vil dele forretningsregler inn i. De fire kategoriene er: integritetsregler, prosessorienterte forretningsregler, strukturelle forretningsregler og slutningsregler. Det er ikke gitt at alle typer regler vil være nødvendig å ha med i en forretningsregelmodell. Forretningsregler som ikke endres ofte, som kan vises i andre modeller og som ikke har noen klare fordeler av å implementeres separat fra annen programkode, er regler som ikke behøver å være med i en forretningsregelmodell. De fire kategoriene er grunnlaget for å finne ut hva som bør skilles ut i en egen forretningsregelmodell på virksomhetsmodelleringsnivå. For hver kategori vil jeg gi en definisjon, samt analysere kategorien i henhold til kriteriene under. Dersom de fleste kriteriene oppfylles, så taler det for å ha regelkategorien i forretningsregelmodellen.

Kriterium 1: *Kategorien kan ikke vises i andre modeller på CIM-nivå.* Dersom kategorien kan vises i andre modeller på CIM-nivå (informasjonsmodell, organisasjonsmodell eller prosessmodell), og reglene kommer

tydelig frem der, vil det ikke være nødvendig å lage en ekstra modell for å vise de forretningsreglene.

Kriterium 2: *Det er likheter mellom kategorien og produksjonsregler.* Hvis det er likheter mellom kategorien og produksjonsregler betyr det at det er enkelt å implementere reglene i en regelmotor, fordi regelmotorer støtter opp om produksjonsregler.

Kriterium 3: *Kategorien består av regler som kan bli endret ofte.* Ved å implementere regler i en regelmotor er det enkelt å håndtere endringer. Dersom regelen skal endres kan man endre den direkte i regelmotoren, istedet for å gå gjennom tusenvis av linjer med kode, og forandre regelen der hvor den brukes.

Kriterium 4: *Reglene i kategorien er av stor betydning for virksomheten.* Dersom reglene er særdeles viktige for en virksomhet, så bør det tas vare på i en eksplisitt del i implementasjonen.

Kriterium 5: *Kategorien vil ha fordeler med å bli implementert i en regelmotor.* Det kan være at en regel vil ha fordeler med å implementeres i en regelmotor, selv om de ovennevnte kriteriene ikke er oppfylt. Denne fordelen kan veie opp for de andre kriteriene.

Integritetsregler

Integritetsregler er begrensninger og regler som definerer hvordan ting skal være, eventuelt begrenser forekomsten av elementer eller lignende. En begrensning kan være en obligatorisk restriksjon eller en foreslått restriksjon på oppførselen til en forretningshendelse [70]. Et eksempel på en begrensning kan være "en kunde kan plassere maksimalt 10 ordre samtidig". En slik begrensning kan enkelt vises i en informasjonsmodell med klasser, assosiasjoner, kardinaliteter og roller (figur 6.1).

Figur 6.1: Begrensning illustrert i klassediagram

Definisjonen av integritetsregel er inspirert av definisjonene av *integrity rules* [71], *guideline* i [65] og [70] og *constraint* [70], samt Webster's Dicitionary [59].

Definisjon 6.2.1. *En integritetsregel er en forretningsregel som brukes for å håndheve strategier eller policies innen et forretningsdomene og som setter begrensninger på oppførselen til en forretningshendelse.*

- **Kriterium 1:** Som illustrert i figur 6.1, så kan noen integritetsregler vises ved kardinaliteter og roller i et klassediagram. Integritetsregler vil stort sett berøre informasjon og fakta i virksomheten, og det er derfor viktig at disse reglene blir definert sammen med informasjonsmodellen til virksomheten. OCL er et språk som kan benyttes

til dette dersom ikke alle integritetsreglene kan uttrykkes ved kardinaliteter og roller i klassediagrammet. Ved å legge inn OCL-uttrykk i klassediagram får en satt på ønsket integritetsregel i henhold til de termer eller fakta som reglene berører i virksomheten.

- **Kriterium 2:** Det er ingen direkte likheter mellom integritetsregler og produksjonsregler.
- **Kriterium 3:** Integritetsregler er ikke regler som vil endre seg så hyppig, fordi de setter begrensninger på elementer i virksomheten som er fastlagte i henhold til forretningspolicies.
- **Kriterium 4:** Integritetsregler er av betydning for virksomheten.
- **Kriterium 5:** Integritetsregler vil ikke ha noen stor fordel av å implementeres i en regelmotor fordi koblingen mellom informasjonsmodellen og regelmotoren vil bli mer komplisert. For hver gang et element legges til i informasjonsmodellen, må en gå til regelmotoren, slik at den kan bestemme om elementet er lovlig eller om det bryter en integritetsregel. Slike informasjonsmodeller vil som regel implementeres i en form for database, og databaser tilbyr støtte for håndtering av integritetsregler. Dermed er det enkleste å la databasen håndtere disse reglene.

Konklusjonen er at integritetsregler ikke vil være i en egen regelmodell fordi de best vises i et klassediagram, og enkelt kan implementeres i en database.

Prosessorienterte forretningsregler

Navnet på denne kategorien er valgt fordi regler her er forretningsregler som igangsetter eller påvirker prosesser i virksomheten. En prosessorientert forretningsregel har en eller flere betingelser som sjekkes, og dersom den evalueres til sann, initieres en forretningshendelse. Definisjonen av en prosessorientert forretningsregel, som vist under, er inspirert av definisjonen av *reaction* [71], *operative business rule* [1], *action enabler* [70] og *action assertion* [70].

Definisjon 6.2.2. *En prosessorientert forretningsregel er en forretningsregel som evaluerer en eller flere betingelser, og dersom de evalueres til sanne, igangsetter en prosess eller annen type forretningshendelse.*

- **Kriterium 1:** I en prosessmodell er det mulig å se igjen prosessorienterte forretningsregler ved at navnet på regelen står som input til prosessen. Regelen i seg selv vises ikke eksplisitt i en prosessmodell. Dette taler for å ha en egen modell for prosessorienterte forretningsregler.
- **Kriterium 2:** Det er en viss likhet mellom prosessorienterte forretningsregler og produksjonsregler. En produksjonsregel er på formen *if* <*condition*> *then* <*action list*> og en prosessorientert forretningsregel er på formen *if* <*condition list*> *then* <*process start*>. En slik regel

kan f.eks. være "hvis en ordre fra kunde er gyldig, så skal prosessen PlasserOrdre igangsettes"

- **Kriterium 3:** Om prosessorienterte forretningsregler er regler som vil bli endret ofte, er vanskelig å si. Det vil være avhengig av hvordan virksomheten fungerer. Forandres prosessene ofte? Forandres regler for beslutninger ofte? Disse spørsmålene er det vanskelig å gi svar på. Noen prosesser vil nok forandres ofte, mens andre vil ikke.

- **Kriterium 4:** Prosessorienterte forretningsregler er av høy prioritet, i og med at de har å gjøre med noe av det mest sentrale i en virksomhetsmodell, prosessene. Disse reglene bør vises som input til prosessmodeller eller modeller av beslutningsstrukturen.

- **Kriterium 5:** Ingen andre fordeler som må nevnes.

Mange av kriteriene gitt for at en regel skal være med i en forretningsregelmodell er oppfylte, og derfor skal prosessorienterte forretningsregler være med i en forretningsregelmodell. En prosessorientert forretningsregel vises ikke direkte i andre modeller på CIM-nivå, og prosessorienterte forretningsregler har store likheter med produksjonsregler. Prosessorienterte forretningsregler kan enkelt implementeres i en regelmotor, men det må passes på at det er en kobling mellom regelmotoren og prosessmotoren fordi prosessreglene har stor innvirkning på prosessene som er implementerte i prosessmotoren. Koblingen kan være i form av at den går via andre komponenter, eller at den går direkte mellom regelmotoren og prosessmotoren.

Strukturelle forretningsregler

Strukturelle forretningsregler er regler som tvinger en viss struktur inn i virksomheten. Denne kategorien faller ganske tett inn til integritetsregler, men skiller seg samtidig fra den. Strukturelle forretningsregler sier mer om strukturen i virksomheten, og angir også autoriseringer og definisjoner for virksomheten. Definisjonen av en strukturell forretningsregel er inspirert av definisjonen av *definitional* [71], *authorization* [71], *structural business rule* [1] og *structural assertion* [70].

Definisjon 6.2.3. *En strukturell forretningsregel er en forretningsregel som påtvinger at gitte fakta forblir sanne og som uttrykker deler av strukturen til virksomheten.*

- **Kriterium 1:** Flere regler i denne kategorien vil omhandle strukturen i virksomheten. Strukturen i virksomheten vises blant annet ved organisasjonskart, men kommer også frem gjennom prosessmodeller og modeller over beslutningsstrukturen. Dersom en strukturell forretningsregel påtvinger at gitte fakta forblir sanne, vil vi i likhet med integritetsregler, angi regelen i en informasjonsmodell i form av OCL-uttrykk.

- **Kriterium 2:** Det er ingen direkte likheter mellom strukturelle forretningsregler og produksjonsregler.
- **Kriterium 3:** Strukturelle forretningsregler vil ikke forandre seg så ofte. Dette er fordi det har å gjøre med struktur i virksomheten, og slik struktur er ofte fastlagt gjennom forretningspolicies. Organisasjonsstrukturen i en virksomhet er f.eks. noe som ikke vil forandre seg særlig ofte, men er mer fastlagt.
- **Kriterium 4:** Strukturelle forretningsregler er viktige for virksomheten. Alle forretningsregler er på en eller annen måte viktige for virksomheten, men ikke aller like betydningsfulle i den daglige drift. Strukturelle regler er som nevnt noe som er fastlagt i virksomheten, og det er derfor ikke behov for å aktivt benytte disse reglene i like stor grad som f.eks. prosessorienterte forretningsregler.
- **Kriterium 5:** Det er ikke behov for å implementere strukturelle forretningsregler i en regelmotor, fordi slike regler vises igjen i f.eks. organisasjonskart og informasjonsmodell. En informasjonsmodell bør heller implementeres i en database.

Når en strukturell forretningsregel uttrykker deler ved strukturen til virksomheten, kan vi se dette igjen i andre modeller som direkte omhandler strukturen i virksomheten, som f.eks. en organisasjonsmodell eller i informasjonsmodellen. Derfor er det ikke nødvendig å ha strukturelle forretningsregler i en egen forretningsregelmodell. De vises like så godt i andre modeller, og informasjonsmodellen vil til slutt implementeres i en database, og dermed kan også de strukturelle forretningsreglene implementeres der.

Slutningsregler

Slutningsregler er den siste kategorien av regler som jeg har definert. Denne kategorien omfatter beregningsregler og avledningsregler. Beregningsregler er f.eks. regler som brukes for å beregne pris, skatt eller lignende på et produkt. Definisjonen under er inspirert av definisjonene av *computation, inference og derivation* i [71] og [70].

Definisjon 6.2.4. *En slutningsregel er en regel som består av en betingelse og en eller flere konklusjoner, hvor konklusjonene er i form av en nytt faktum elle ny informasjon.*

- **Kriterium 1:** Slutningsregler kan vises i informasjonsmodeller. Dette er mulig ved å bruke OCL-uttrykk i klassediagrammet. For en del slutningsregler vil OCL-uttrykkene bli veldig kompliserte.
- **Kriterium 2:** Avledningsregler er utsagn som tester en betingelse, og dersom den er sann fastslår sannheten om et nytt faktum [70]. Eksempel: "Dersom en kunde ikke har noen utestående faktura, så er kunden av foretrukken status". Resultatet av en slutningsregel er

å lage ny informasjon. Disse reglene er på formen *hvis x så y*, og har likhet med produksjonsregler.

- **Kriterium 3:** Slutningsregler innebærer regler som helt klart vil forandre seg ofte. Hvis f.eks. momsreglene forandres vil det ha innvirkning på beregningsregler.
- **Kriterium 4:** Slutningsregler vil ha en påvirkning på mange ulike modeller i virksomheten. Dersom en prosess f.eks. har med beregning av pris på et element å gjøre, vil en beregningsregel være input, eller støtteregel underveis i prosessen. Dette gjør at slutningsregler er meget viktige for virksomheten, særlig i den daglige drift.
- **Kriterium 5:** Ved å ha slutningsregler i en regelmotor, vil det være enkelt å finne når gitte regler vil gjelde. I og med at prosesser vil gjøre bruk av regler, kan en enkel kobling mellom prosessene og reglene opprettes i regelmotoren.

Slutningsregler skal være med i forretningsregelmodellen, selv om det er mulig å vise dem i andre modeller. Dette fordi det er store likheter mellom slutningsregler og produksjonsregler, og fordi andre modeller, som prosessmodellen, bruker slutningsregler, og bruken er enkel ved å ha disse reglene i en regelmotor.

6.3 Visualisering av forretningsregler

Av de løsningene som ble evaluert i kapittel 5 var det bare SBVR, som oppfylte kravet, eller delvis oppfylte kravet, om en forretningsregelmodell. SBVR definerer alle forretningsregler med naturlig språk, og reglene brukes for å beskrive omtrent alle aspekter ved bedriften. Det er lettere å oppfatte prosessene i bedriften, organiseringen av bedriften m.m. dersom vi modellerer dette på en grafisk måte. Det menneskelige øyet oppfatter lettere symboler enn en mengde av tekst [2]. Vi må med andre ord se nærmere på hvordan vi kan visualisere regler.

Forretningsregler kan ifølge Vanthienen [67] visualiseres på fem ulike måter: naturlig språk, logikk, strukturert naturlig språk, beslutningstabeller (også trær, grafer og diagrammer) og med OCL. Naturlig språk kan være uklart og tvetydig, men er samtidig noe alle forstår. Ved å bruke logikk for å uttrykke regler får man en kraftig uttrykksmåte som hjelper til med å avgrense tolkningsmengden til regler, men logikk er ikke et språk for forretningsfolk. Strukturert naturlig språk er et subset av naturlig språk. Det er dette SBVR har valgt å bruke fordi det kan være uttrykksfullt og enkelt å forstå samtidig som det kan være en kraftig uttrykksmåte. Vanthienen mener at denne balansen mellom kraftig og enkelt å forstå, er vanskelig å finne, og derfor er ikke strukturert naturlig språk den beste måten å uttrykke regler på. OCL er en del av UML og er egnet å bruke til pre- og postbetingelser som jo regler ofte brukes til, men Vanthienen argumenterer for at OCL ikke dekker alt av forretningsregler. Beslutningstabellen er det

Vanthienen mener at er den beste måten å representere forretningsregler på. En *beslutningstabell* er definert som "a table representing the complete set of mutually exclusive conditional expressions in a predefined area" [67].

Beslutningstabeller bør brukes på grunn av deres kraftfulle visualisering. Slike tabeller vil gi en kompakt og strukturert presentasjon av alle reglene. I tabell 6.1 er det vist et eksempel på en beslutningstabell. Ved å bruke slike tabeller kan man lettere unngå feil fordi man setter opp alle alternativ på en strukturert måte. Fra en beslutningstabell er det mulig å få en rask og god eksekvering av reglene gjennom å sette tabellen opp som et tre.

1. Wholesaler	Yes					No
2. Quantity Ordered (Q)	Q<10	10<=Q<15			Q>=15	-
3. Travel Distance (D)	-	D<50	50<=D<100	D>=100	-	-
1. Discount 0%	-	-	-	-	-	x
2. Discount 2%	-	-	-	x	-	-
3. Discount 5%	-	-	x	x	-	-
4. Discount 10%	-	x	-	-	x	-
5. Railway Transport	-	-	-	-	x	x
6. Road Transport	x	x	x	x	-	-
7. Bill Type A	x	x	x	x	-	x
8. Bill Type B	-	-	-	-	x	-

Tabell 6.1: Eksempel på beslutningstabell

6.4 Krav til en forretningsregelmodell

Forretningsregler kan deles inn i fire kategorier, som definert i tidligere avsnitt: integritetsregler, prosessorienterte forretningsregler, strukturelle forretningsregler og slutningsregler. Av disse kategoriene er det kun to som skal være med i forretningsregelmodellen, og det er de prosessorienterte forretningsreglene og slutningsreglene. På bakgrunn av denne kategoriseringen har jeg kommet fram til fem krav til forretningsregelmodell (krav CIM7) som vist i tabell 6.2. Disse kravene er grunnlaget for metoden REMO, i tillegg til kravene fra kapittel 3. Kravene presentert her vil også brukes som grunnlag for evaluering av metoden.

Krav til CIM7 - Forretningsregelmodell
CIM7FR1: Prosessorienterte forretningsregler
CIM7FR2: Slutningsregler
CIM7FR3: Visualisering
CIM7FR4: Transformasjonsmulighet
CIM7FR5: Verktøystøtte

Tabell 6.2: Krav til CIM7 - Forretningsregelmodell

CIM7FR1 - Prosessorienterte forretningsregler

Dette kravet er satt fordi denne type regler skal være en del av forretningsregelmodellen, i og med at de ikke vises gjennom andre modeller på virksomhetsnivå. Disse vil ha en nøye sammenheng med prosessmodellen, og det er derfor viktig å definere eksplisitt og separert fra prosessmodellen, men samtidig vise hvordan elementene her kan knyttes opp mot prosessmodellen.

CIM7FR2 - Slutningsregler

Slutningsregler skal være en del av forretningsregelmodellen. Slutningsregler kommer best frem dersom vi modellerer eller spesifiserer dem separat fra hvor de spiller inn. Noen av reglene kan defineres med OCL i et UML klassediagram, men i og med at slutningsregler er regler som kan forandre seg ofte så er det ønskelig å ha dem med i en regelmotor.

CIM7FR3 - Visualisering

Dette kravet går ut på noe av det samme som vi finner i krav CIM1 (brukervennlighet). Forretningsreglene defineres ved at forretningspersoner hjelper oss å identifisere dem. Vi må derfor lage reglene på en måte som er lett forståelig for forretningsfolk. SBVR har valgt naturlig språk som sin visualisering av forretningsregler nettopp for at forretningsfolk skal forstå dem. Jeg mener at ren naturlig tekst fort kan bli litt mye, og at det er lettere å holde oversikt over figurer. Derfor er det et krav om god visualisering av forretningsregler i en forretningsregelmodell. Med visualisering mener jeg å presentere regler på en oversiktlig og helst grafisk måte.

CIM7FR4 - Transformasjonsmulighet

Forretningsreglene vi lager ønsker vi å transformere til en regelmodell på plattformuavhengig nivå. Dette kan gjøres i henhold til metamodellen definert for produksjonsregler [57] fra OMG. Derfor er det et krav om at forretingsregelmodellen skal være mulig å transformere til produksjonsregler ved hjelp av modelltransformasjoner.

CIM7FR5 - Verktøystøtte

Vi stiller krav om verktøystøtte spesifikt for forretningsregelmodellen. Forretningsregler er et såpass uavklart område i virksomhetsmodelleringen og derfor trengs det verktøystøtte som gjør det mulig å klart definere reglene som en del av virksomhetsmodellen, slik at transformasjon til produksjonsregler kan gjøres på en teknisk enkel måte.

Kapittel 7

Regel- og modelldrevet metode (REMO)

Den metoden som beskrives i dette kapittelet er laget for å tilfredsstille alle krav som beskrevet i kapittel 3 og i kapittel 6.

7.1 Metoden

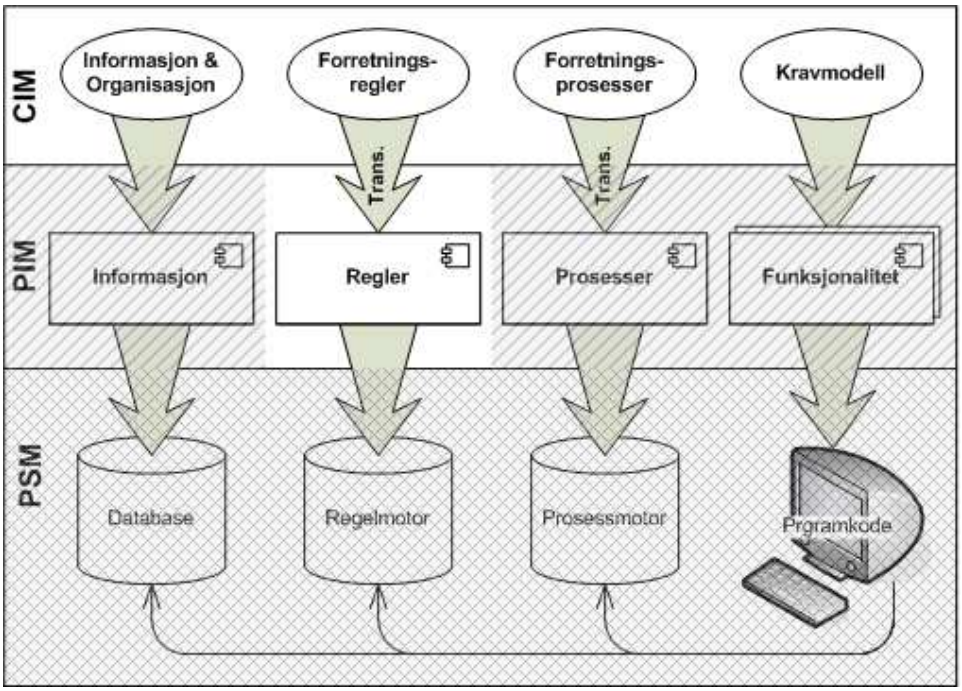
Metoden har fått navnet Regel- og modelldrevet metode (REMO). Dette er fordi metoden baserer seg på modelldrevet systemutvikling, med spesielt fokus på håndtering av forretningsregler på et tidlig tidspunkt.

Metoden er foreløpig kun utviklet for CIM og PIM, på grunn av avgrensningen som nevnt i kapittel 2. Metoden skal være relativt grei å utvide til å gjelde for PSM også, men da kreves det flere undersøkelser enn det som er gjort i denne omgang. REMO omfatter hvordan CIM skal utformes, hvordan transformere enkelte deler av CIM, og hvordan PIM bør utformes. Dette gjøres for å skille forretningsregler ut tidlig og bruke virksomhetsmodellene for å spesifisere IT-systemet. For PIM-nivå har regelmodellen vært hovedfokus.

I figur 7.1 på neste side er metoden illustrert og figuren er basert på utgangspunktet med modeller på tre abstraksjonsnivåer som vi så i figur 2.1 på side 6. CIM, PIM og PSM er alle delt inn i fire elementer: informasjon, regler, prosesser og et funksjonalitetselement. Hvert av disse elementene vil bli forklart etter hvert. I figuren er overgangene mellom abstraksjonsnivåene illustrert ved piler. Det er ikke gitt at alle pilene betyr modelltransformasjoner. Overgangene mellom PIM og PSM vil nok være modelltransformasjoner, men denne biten har jeg sett bort fra på grunn av avgrensningen som tidligere presentert. Overgangene mellom CIM og PIM har jeg spesifisert i REMO. Mellom forretingsregler i CIM og regler i PIM vil det være en modelltransformasjon, og det vil det også være mellom forretningsprosessene i CIM og prosessene i PIM.

Transformasjonen av regler vil bli presentert senere i dette kapittelet. Transformasjon av forretningsprosesser til prosessmodeller i PIM arbeider for tiden ATHENA med, og planen er å anvende deres resultater når de foreligger.

REMO bruker noen av de eksisterende løsninger som er presentert i kapittel 4. De som skal brukes er valgt ut på bakgrunn av evalueringen som er gjort tidligere. De neste avsnittene vil gi en detaljert veiledning for CIM, PIM og modelltransformasjoner.



Figur 7.1: Illustrasjon av REMO

7.2 Virksomhetsmodell og kravmodell

I REMO har jeg definert fem deler som skal være med i en CIM:

- **Forretningsregelmodellen** skal vise alle forretningsreglene i bedriften og inneholder prosessorienterte forretningsregler og slutningsregler.
- **Organisasjonsmodellen** definerer strukturen til virksomheten.
- **Informasjonsmodellen** viser forretningsvokabularet til virksomheten, avgrenset til den delen som er relevant for IT-systemet som skal spesifiseres eller tilpasses.
- **Forretningsprosessmodellen** definerer alle prosessene i bedriften som er relevante for systemet som skal implementeres. Her er det viktig å skille mellom "as is"- og "to be"- prosesser.

- **Kravmodell** foreskriver funksjonaliteten til systemet som skal implementeres. Her skal kravene sett i konteksten av virksomheten som er rettet til systemet modelleres. Kravmodellen ligger både som en del av CIM og som en del av PIM, og kan brukes for å få til overgangen fra CIM til PIM.

UML er valgt som hovedmodelleringsspråk, fordi dette er en standard som er meget utbredt i bruk. UML 2.0 er den nyeste versjonen av UML og vi skal benytte denne versjonen. I tillegg skal vi bruke UML-profiler basert på UML 2.0.

7.2.1 Forretningsregelmodell

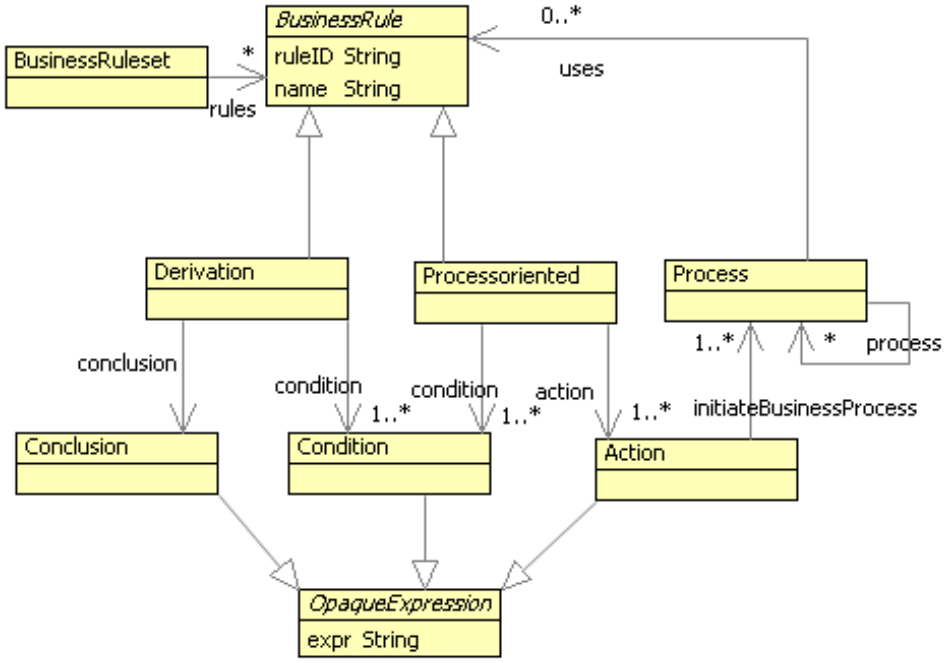
Forretningsregelmodellen skal inneholde alle forretningsregler som er viktige for virksomheten og som er av betydning for IT-systemet som skal spesifiseres. Forretningsregelmodellen skal inkludere to typer regler: prosessorienterte forretningsregler og slutningsregler. Det er viktig å visualisere reglene grafisk så langt det er mulig, og de må vises på en slik måte at forretningsfolk forstår. Vanthienens [67] meninger om å visualisere forretningsregler ved å bruke beslutningstabeller synes jeg er en god idé, men jeg har samtidig et ønske om å visualisere forretningsregler ved å bruke UML. Dessuten er det ikke mulig å visualisere alle regler med beslutningstabeller. Derfor vil utviklingen av forretningsregelmodellen foregå i flere steg. Det første en gjør er å formulere reglene ved naturlig språk (som brukes i SBVR) eller ved beslutningstabeller hvor det er mulig. Deretter oversetter man dette til selve forretningsregelmodellen.

Som vi så på i kapittel 4, gir SBVR oss strukturert naturlig språk som middel for å representere regler, og det skal vi også bruke. Reglene som finnes i virksomheten skal skrives basert på termer og fakta i informasjonsmodellen (avsnitt 7.2.2). Reglene kan formuleres på bakgrunn av forretningspolicy, kunnskap som forretningsfolkene har, eller som resultat fra modelleringen av forretningsprosessene og informasjonsmodellen. Ofte vil en komme på flere regler når en utarbeider andre modeller.

Beregningsregler og resten av det som hører hjemme i kategorien slutningsregler kan enkelt å visualiseres i en beslutningstabell. Ved å bruke beslutningstabeller for å visualisere regler er det lettere å få alle alternativ klart og tydelig frem som regler [67]. Tabell 6.1 på side 58 viser et eksempel på en beslutningstabell.

På bakgrunn av reglene vi definerer med naturlig språk og beslutningstabellene, kan vi utarbeide forretningsregelmodellen. Det er viktig å huske på at identifisering av forretningsregler er en inkrementell prosess, det vil si at det vil stadig oppdages nye regler som må tas med i regelmodellen. De reglene som identifiseres først, er ikke nødvendigvis alle reglene som en ender opp med til slutt i forretningsregelmodellen, det blir som regel flere. Reglene vil dessuten bearbeides med og reformuleres helt til man får de reglene som beskriver det som er ønskelig.

Jeg har utarbeidet en metamodell for forretningsregler. Denne har jeg laget en UML-profil for i RSM. Metamodellen er laget på bakgrunn av definisjonene for prosessorienterte forretningsregler og slutningsregler fra forrige kapittel. Figur 7.2 viser denne metamodellen. Den sentrale klassen i modellen er den abstrakte klassen *BusinessRule*. Denne definerer hva slags egenskaper en regel skal ha tilknyttet seg. Den skal ha en id og et navn. Forretningsregler kan deles i fire kategorier, hvorav to skal være med i forretningsregelmodellen. Disse to kategoriene er slutningsregler og prosessorienterte forretningsregler. De to, vist som klassen *Derivation* og *Processoriented*, er subklasser av *BusinessRule*. En slutningsregel kan ha en eller flere betingelser (*Condition*) tilknyttet seg, og en *conclusion*. Betingelser og konklusjoner består av et utsagn (*expr*) som de arver fra den abstrakte klassen *OpaqueExpression*. En prosessorientert forretningsregel kan også ha en eller flere betingelser tilknyttet seg, og kan ha en eller flere hendelser (*action*) som resultat. En hendelse kan igangsette en prosess. Klassen *process* har vi fra prosessmetamodellen (avsnitt 7.2.4). En prosess kan bruke forretningsregler underveis i prosessen, og disse kan være både slutningsregler og prosessorienterte regler.



Figur 7.2: Metamodell for forretningsregler

7.2.2 Informasjonsmodell

Alle regler bygger på fakta, og fakta bygger på termer [23]. Dette er bakgrunnen for å lage regler. Derfor må vi alltid definere termer og fakta for å lage regler. En term er et substantiv eller et uttrykk med flere substantiv, med en definisjon [70]. Eksempler på termer er kunde, ordre, bilutleie,

mobiltlf. osv. Slike begreper kan enkelt vises i en informasjonsmodell ved å bruke et klassediagram. Informasjonsmodellen underbygger forretningsregelmodellen, og derfor skal vi lage informasjonsmodellen slik at den beskriver hele forretningsvokabularet.

Som vi har sett i kapittel 6 skal en kategori av forretningsregler modelleres i informasjonsmodellen, og det er integritetsregler. De integritetsreglene vi ikke klarer å uttrykke med roller, kardinaliteter og assosiasjoner kan vi vise ved å bruke OCL [14]. Klassediagram er valgt for å modellere informasjonsmodellen.

7.2.3 Organisasjonsmodell

Jeg har valgt å bruke organisasjonsdimensjonen fra POP* [5] for å lage organisasjonsmodellen. I avsnitt 5.1.3 på side 36 er metamodellen for organisasjonsdimensjonen i POP* vist og forklart (se også tillegg C for forklaring av metamodellene i POP*).

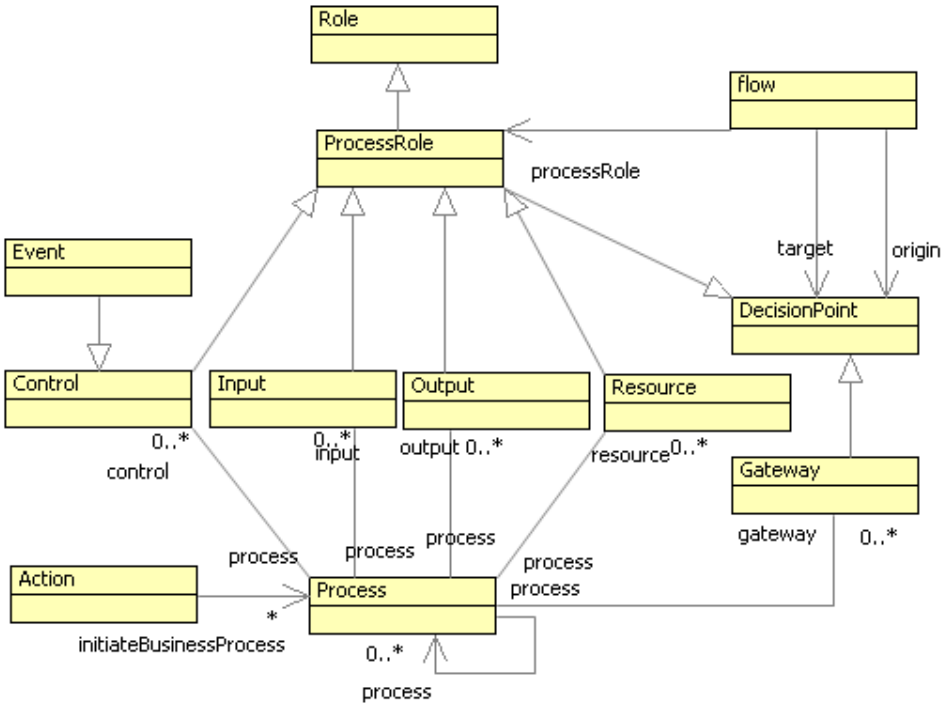
7.2.4 Prosessmodell

Metoder og språk for forretningsprosesser er presentert i kapittel 4. Jeg har valgt å bruke prosessdimensjonen fra POP* [5], men har utvidet den for å eksplisitt få frem hvordan prosessene henger sammen med forretnigsreglene.

I figur 7.3 på neste side er denne metamodellen vist, modellert i XMF-Mosaic. XMF-Mosaic har en UML-lignende syntaks, men tilbyr ikke elementene som aggregering og komposisjon, og derfor skiller denne seg noe fra prosessmetamodellen slik den er definert i POP*. Metamodellen for POP* er forklart i kapittel 5 og i tillegg C. Klassen *Action* er min utvidelse av denne metamodellen. Denne klassen er hentet fra metamodellen for forretningsregler (avsnitt 7.2.1), og den representerer hendelser som kan sette i gang prosesser. Ikke alle prosesser settes i gang av en prosessorientert forretningsregel, men det er viktig å få frem *når* den gjør det.

Forretningsprosesser kan beskrive to ting: forretningsprosesser slik de er i dag (“as is”) og forretningsprosesser slik de skal bli ved implementasjon av IT-systemet (“to be”). Ved å beskrive “as is”-prosessene får en et bilde av hvordan virksomheten er og fungerer per dags dato, før en begynner å forandre på prosessene. “To be”-prosesser skal lages med tanke på IT-systemet som er under utvikling.

For å modellere prosessene kan aktivitetsdiagram brukes, med profilen fra POP* tilknyttet.



Figur 7.3: Prosessmetamodell

7.2.5 Kravmodell

Denne modellen er ment som et hjelpemiddel for å komme fra CIM til PIM, men en del er viktig å ha med på dette nivået. Dette er modeller som brukes for å identifisere funksjonalitet til systemet. Det viktigste her er et Business Use Case-diagram. Det er et vanlig use case diagram hvor man viser hvilke aktører som bruker hvilken funksjonalitet, men det er mer fokusert mot virksomheten. Her vil man typisk trekke inn elementer fra organisasjonsmodellen. En person eller en organisasjonsrolle vil opptre som aktør i et use case-diagram. Dette use case-diagrammet er det samme som *system boundary model* fra COMET [10].

Denne modellen vil videre detaljeres ved at man går inn i hvert use case og detaljerer. Detaljeringen vises med nye use case diagrammer eller interaksjonsdiagrammer. Slike interaksjonsdiagrammer er også en del av prosessmodellen, men det kan være behov for å lage et interaksjonsdiagram som fokuserer enda mer på systemet og funksjonaliteten, og da kan man ta med det her. Dette kan også brukes som en overgang til PIM.

7.3 Plattformuavhengig modell

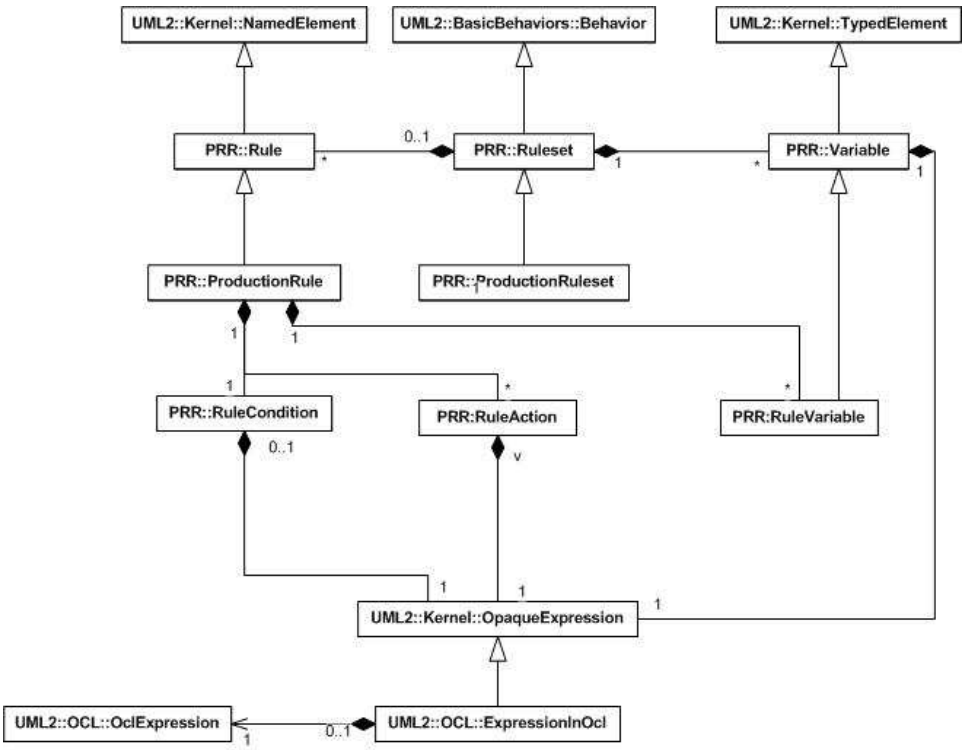
En plattformuavhengig modell skal beskrive arkitekturen til IT-systemet. I REMO er målet for denne å bruke så mye som mulig av CIM slik at systemet kan bli tilpasset best mulig til den virksomheten som skal bruke det.

På dette nivået er det litt opp til systemarkitekten hvilke modeller som skal være til stede, med tanke på det som ikke angår regelmodellen. REMO gir derfor bare en veiledning på hva som skal være med (jamfør avgrensningene i kapittel 2). De viktigste faktorene for å støtte opp mot virksomhetsmodellen er de tre tekniske komponentene som ble presentert i kapittel 3: en database, en regelmotor og en prosessmotor. Hver av disse skal vises med en modell på PIM-nivå. En informasjonsmodell skal illustrere databasekomponenten, en modell av produksjonsreglene skal illustrere regelmotoren, og en prosessmodell skal illustrere prosessmotoren.

7.3.1 Regelmodell

I evalueringen av eksisterende løsninger i kapittel 5, så vi at PRR oppfylte kravet om regelmodell for PIM-nivå. Derfor er PRR valgt som grunnlag for regelmodellene på PIM-nivå i REMO.

Regelkomponenten vil hovedsakelig bestå av forretningsregler som er transformert til produksjonsregler. Denne komponenten utarbeides i henhold til metamodellen definert i PRR [55]. Figur 7.4 viser metamodellen for produksjonsregler. Metamodellen tar utgangspunkt i konsepter fra UML 2.0, og utvider disse med konsepter for å beskrive regler.



Figur 7.4: Metamodellen for produksjonsregler [54]

PRR::Rule er et generelt konsept for regler. I fremtiden kan det være at man standardiserer flere typer regler, og da vil de komme som en kategori av

PRR::Rule. En regel representerer en betinget del med programmeringslogikk, inkludert da produksjonsregler. Et regelsett fungerer som en beholder for regler, og gir en kontekst for eksekvering av regler. Regelsett kan ha variabler tilknyttet seg. I figur 7.4 ser vi dette som PRR:Ruleset.

PRR::ProductionRuleset representerer et regelsett for produksjonsregler. Produksjonsregelsett har et attributt som beskriver den operasjonelle semantikken til regelsettet. PRR::ProductionRule er et utsagn av programmeringslogikk som spesifiserer eksekveringen av en eller flere hendelser i det tilfellet hvor betingelsene er oppfylte [54]. En produksjonsregel er på formen: *if [condition] then [action-list]*. En produksjonsregel må ha en betingelse og en eller flere hendelser tilknyttet seg.

PRR::Variable representerer en programmeringskonstrukt som holder en eller flere verdier for å brukes til eksekvering av regler. PRR::RuleCondition er et boolsk uttrykk som avgjør om gitt hendelse skal utføres eller ikke. I PRR kan det kun være en betingelse knyttet til en regel. PRR::RuleAction gir en liste av ordnede hendelser som skal inntreffe dersom en gitt betingelse evaluerer til sann. PRR::RuleVariable angir domenet som skal brukes i eksekveringen av regler.

Underveis i detaljeringen av regelkomponenten, eller andre komponenter på PIM-nivå, kan det være at en oppdager nye regler som må være med. Disse reglene kan da legges til i regelkomponenten. Det eneste en må passe på er at informasjonsmodellen på PIM-nivå også oppdateres, for når nye regler tas med, kan det være at regelen introduserer nye ord og uttrykk som ikke er tatt med i informasjonsmodellen fra før, men som er nødvendige for at reglene skal kunne eksekveres til slutt.

7.3.2 Resten av PIM

PIM er fra kapittel 2 avgrenset, men nedenfor er det en veiledning på hva som bør være med i PIM i REMO, i tillegg til regelkomponenten. Her er det både konkrete modellelement, og også forslag til hvordan sammenhengen mellom disse elementene kan defineres. Med konkrete modellelement tenker jeg på de som er vist i figur 7.1 presentert tidligere i dette kapittelet. Her er det gitt fire modellelement: informasjon (informasjonsmodell), regler (regelmodell som presentert i avsnittet over), prosesser (prosessmodell) og funksjonalitet (funksjonsmodell). For å vises sammenhengen mellom disse skal det i tillegg lages komponentstruktur, komponentinteraksjonsmodeller og interfacemodell. Dette har vi fra kravene i kapittel 3.

Informasjonsmodellen vil være en direkte transformasjon fra informasjonsmodellen i CIM, men med mulige tilføyninger dersom det underveis oppdages elementer som må lagres i en database.

Prosessmodellen er resultatet av transformasjonen av prosessmodellen i CIM. Her henviser jeg til bruk av prosessdimensjonen i PIM4SOA [7]. Forklaring av denne er lagt ved i tillegg C. Denne skal være et resultat

av transformasjon av prosessmodellen i CIM, men med utvidelser for å detaljere prosessene mot det rent tekniske. I ATHENA arbeides det med denne overgangen.

Funksjonaliteten i systemet er det som bestemmer hvordan alt henger sammen, og denne må dermed detaljeres. Dersom det er ønskelig kan hver funksjonalitet i systemet representeres som en egen tjeneste eller komponent, alt etter formålet med IT-systemet som lages. Her kan PIM4SOA anvendes, med tjenestebiten av metamodellen, dersom det skulle være hensiktsmessig.

For å definere hvordan de ulike komponentene henger sammen må man lage komponentstrukturen. Dette lages enkelt med et komponentstruktur-diagram. Denne vil vise strukturen av komponenentene på et overordnet nivå.

Interaksjonsmodellen skal definere den detaljerte sammenhengen mellom komponentene. Interaksjonsmodellen lages ved et aktivitetsdiagram eller sekvensdiagram, for å vise hvordan komponentene henger sammen for hver funksjonalitet definert i IT-systemet.

Det bør lages en interfacemodell slik at interfacene for hver komponent spesifiseres. Dette er for å lette implementasjonen av informasjonssystemet og dermed kunne transformere til en plattformspesifikk modell uten særlige utvidelser.

7.4 Modelltransformasjon

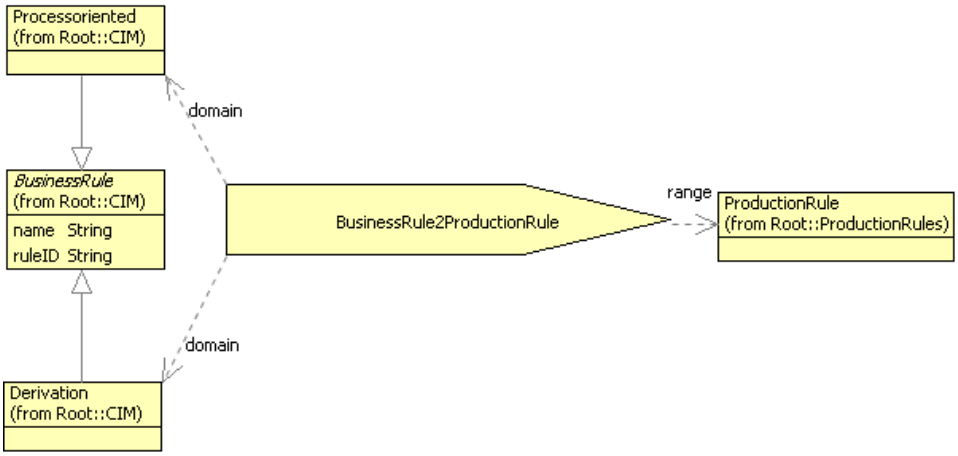
Modelltransformasjoner er en viktig del i MDA, og også i REMO. I REMO vil det være modelltransformasjon fra CIM til PIM, og også fra PIM til PSM når REMO utvides for PSM. Her gis et forslag for transformasjon fra CIM til PIM, og transformasjonene defineres mellom metamodellene til de modellene som skal transformeres.

7.4.1 Transformasjon fra CIM til PIM

For å transformere CIM til PIM trenger vi metamodeller for begge. CIM i REMO inkluderer nå ulike metamodeller for hvert element i CIM. Det vil si at metamodellene vi må forholde oss til er UML 2.0, POP* og metamodellen for forretningsregler definert i denne oppgaven. Nå vil det være en transformasjon av hver enkelt del på CIM-nivå til hver enkelt del på PIM-nivå. Det legges opp til noe bruk av manuelle transformasjoner, det vil si å bruke modellene på CIM og tegne inn det som er nødvendig i PIM. I tillegg vil også noen elementer bare videreføres slik de er, eller ikke videreføres i det hele tatt.

Forretningsregler

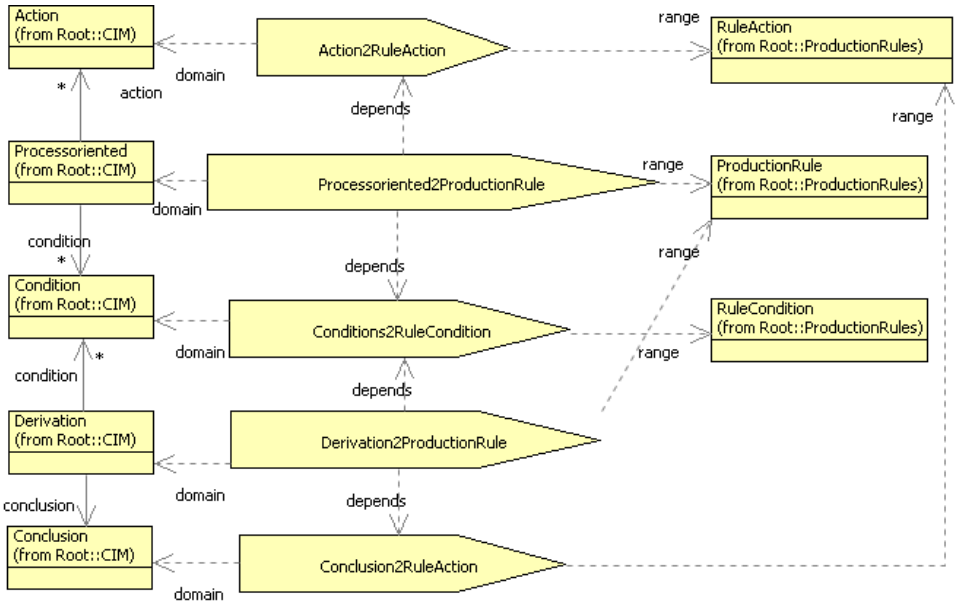
Regelmodellen i PIM skal være resultat av transformasjon av forretningsregelmodellen til produksjonsregler og eventuelle utvidelser som kommer på PIM-nivå. Metamodellen for produksjonsregler fra PRR [54] er valgt som grunnlag for regelmodellen i PIM. Grunnlaget for transformasjonen er metamodellen for forretningsregler og metamodellen for produksjonsregler. I figur 7.5 er den overordnede mappingen mellom forretningsregler og produksjonsregler vist. Figuren viser at det er prosessorienterte forretningsregler og slutningsregler som mappes til produksjonsregler. Mappingen er det elementet med navn "BusinessRule2ProductionRule". Denne figuren er ment som en oversikt over hvilke forretningsregler som mappes til produksjonsregler.



Figur 7.5: Mapping mellom forretningsregler og produksjonsregler

I figur 7.6 på neste side er en mer detaljert mapping av forretningsregler vist. Her er det definert fem mappinger. For å mappe prosessorienterte forretningsregler er "Processoriented2ProductionRule" definert. Denne går gjennom alle tilhørende betingelser og hendelser som er tilknyttet den prosessorienterte forretningsregelen, noe som vi ser i figuren med pilen *depends* som er satt mot "Condition2RuleCondition" og "Action2RuleAction". Den første mapper betingelsen for den prosessorienterte forretningsregelen til en betingelse for en produksjonsregel. Den andre mapper den hendelsen som er resultatet av en prosessorientert forretningsregel til en hendelse som er forbundet med en produksjonsregel. For å mappe slutningsregler om til produksjonsregler er "Derivation2ProductionRule" definert. Denne er laget på tilsvarende måte som for prosessorienterte forretningsregler, ved at den går gjennom betingelsene og konklusjonene som er knyttet til en slutningsregel og mapper disse over til tilsvarende elementer for produksjonsregler med mappingene "Condition2RuleCondition" og "Conclusion2RuleAction".

Metamodellen for forretningsregler (figur 7.2 på side 64) og produksjonsregler (figur 7.4 på side 67) er henholdsvis kilde- og målmodell. På



Figur 7.6: Detaljert mapping mellom forretningsregler og produksjonsregler

bakgrunn av disse har jeg spesifisert transformasjonsregler. Mesteparten av transformasjonsreglene er enkle med bare enkel navngivning og oppretting av objekter. Det som har vært utfordringen er å definere hvordan betingelsene for forretningsregler skal bli til betingelser for produksjonsregler. En forretningsregel kan ha mange betingelser, mens en produksjonsregel kan kun ha en betingelse ifølge PRR [54]. Måten dette er løst på, er ved å konkatenere betingelsene til forretningsregelen med den logiske operatoren *and* som skilleord.

For hver regel skal en gå gjennom alle dens betingelser, transformere disse, og dens konklusjoner eller hendelser, og transformere disse. Det vil være noe ulik transformasjon alt etter hva slags forretningsregel det er. Jeg vil her vise det som har med slutningsregler å gjøre. Alle transformasjonsregler er beskrevet i tillegg H.

For hver slutningsregel blir regelen "Derivation2ProductionRule" kalt. Denne regelen matcher regler av typen *derivation*, og navnet til regelen blir tatt vare på i variabelen *N*. De tilhørende betingelsene (*C*) og koklusjonen (*CON*) blir også tatt vare på i headeren. Etter *do* lages det en ny instans av en produksjonsregel. Produksjonsregelen får samme navn som forretningsregelen. Pekerne *ruleActions* og *ruleCondtition* blir satt ved å kalle på to andre mappinger.

Derivation2ProductionRule

```
@Clause Derivation2ProductionRule
  CIM::Derivation[name=N, condition = C,
                          conclusion = CON]
do
  ProductionRules::ProductionRule[name=N,
```

```
                                    ruleActions = RA,
                                    ruleCondition = RC]
where
  RA = Conclusion2RuleAction()(CON);
  RC = Conditions2RuleCondition()(C)
end
```

Transformasjonsregelen *Conclusion2RuleAction* er en veldig enkel regel. Det den gjør er å lage en *OpaqueExpression* med samme innhold som attributtet *expr* i konklusjonen til forretningsregelen. Dette er vist under.

Conclusion2RuleAction

```
@Clause Conclusion2RuleAction
  CIM::Conclusion[expr=EX]
do
  ProductionRules::RuleAction[opaqueExpression=OE]
where
  OE = ProductionRules::OpaqueExpression[body=EX]
end
```

Conditions2RuleAction er den transformasjonsregelen som er mest komplisert. Her skal alle betingelsene settes sammen og gjøres om til en betingelse. Regelen er løst ved en *clause* og en *operation (makeString)*.

Conditions2RuleCondition

```
@Clause Conditions2RuleCondition
  C->including(CIM::Condition[expr=E])
do
  ProductionRules::RuleCondition[opaqueExpression=OE]
where
  OE=ProductionRules::OpaqueExpression[
                              body=self.makeString(C)]
end

@Operation makeString(C : Set(CIM::Condition))
                                        :XCore::Element
      let expression = ""
      in @For c in C do
           expression := expression + c.expr;
           if not isLast
           then
             expression := expression + " and "
           else
             false
           end
         end;
         self.con_expr := expression;
         expression
      end
    end
```

De andre reglene som er definert er tatt med i tillegg H sammen med

anvendelsen av transformasjonene.

De andre elementene i CIM

Informasjonsmodellen i CIM og informasjonsmodellen i PIM har samme metamodell; UML 2.0 klassediagram, og derfor kan informasjonsmodellen fra CIM brukes direkte i PIM. Den eneste forandringen som vil være nødvendig på PIM nivå vil være eventuelle tilføyninger som kommer fra funksjonalitetsbeskrivelsene eller lignende. Når arkitekturen er ferdig, er det stor sannsynlighet for at informasjonsmodellen vil være utvidet.

Prosessmodellen i CIM skal transformeres til prosessmodellen i PIM. Dette er en transformasjon fra prosessdimensjonen i POP* til prosessdimensjonen i PIM4SOA, og de har noe forskjellige metamodeller. Dette jobber for tiden ATHENA med, og jeg henviser til å bruke deres resultater for dette.

Organisasjonsmodellen vil ikke transformeres til et gitt element i arkitekturen. Noe av informasjonen som fremgår i organisasjonsmodellen tas med i informasjonsmodellen i PIM. Hva som skal være med må vurderes alt etter hva som er nødvendig for implementasjon av systemet.

For å komme frem til ønsket funksjonlitetsbeskrivelse i PIM er det flere detaljeringsteknikker som kan anvendes. COMET gir blant annet noe veiledning på dette (se [10]). Jeg vil ikke gå nærmere inn på dette, jamfør avgrensningen i kapittel 2.

7.5 REMO oppsummert

Under følger en oppsummering av REMO. Oppsummeringen viser alle de ulike elementene som er med i CIM og PIM og tar med hvordan disse kan utformes med tanke på hva slags UML-diagram som anvendes og hvilke UML-profiler som legges til diagrammene.

1. CIM
   (a) Forretningsregelmodell –> Utform forretningsreglene med naturlig språk og beslutningstabeller. Oversett disse til modeller i henhold til metamodellen spesifisert i REMO. Bruk klassediagram med UML-profilen for forretningsregler.
   (b) Organisasjonsmodell –> viser strukturen i bedriften. Bruk UML klassediagram med POP*-profilen for organisasjonsdimensjonen.
   (c) Informasjonsmodell –> Her kommer all informasjon som eksisterer i virksomheten. Denne er også grunnlaget for forretningsreglene, alle ord og uttrykk som er med i forretningsregler må være med i informasjonsmodellen. Bruk vanlig UML klassediagram.

(d) Forretningsprosessmodell –> Forretningsprosessene i bedriften modelleres. Kan godt tenke gjennom hvor forretningsreglene kommer inn her, eller om noen forretningsregler setter i gang en forretningsprosess. Bruk UML aktivitetsdiagram med POP*-profilen for prosessdimensjonen.

(e) Kravmodell –> modellering av ønskelig funksjonalitet sett i virksomhetens sammenheng. Use case diagram.

2. Fra CIM til PIM

   (a) Transformere forretningsregler –> transformere i henhold til transformasjonsreglene som spesifisert i REMO.

   (b) Transformere forretningsprosesser –> transformere i henhold til transformasjonsreglene som ATHENA etterhvert skal spesifisere.

   (c) Videreføre informasjonsmodellen –> Denne tas med til PIM som den er.

   (d) Videreføre kravmodellen –> Bruk kravmodellen som du selv ønsker.

3. PIM. Består av modeller av gitte komponenter (database, regelmotor og prosessmotor), og en funksjonsmodell. I tillegg trenger man modeller for å beskrive sammenhengen mellom komponentene.

   (a) Lag en komponentstruktur. Har tre komponenter: database (informasjonsmodell),prosessmotor (prosessmodell) og regelmotor (regelmodell). Deretter har vi en eller flere funksjonskomponenter innholder systemets funksjonalitet. Hver komponent må detaljeres.

      i. Regelmotor –> Detaljeres ved en regelmodell. Denne er resultat av modelltransformasjoner. Forretningsregler fra CIM transformeres til produksjonsregler. Bruker klassediagram med en eventuell UML-profil for produksjonsregler.

      ii. Database –> Detaljeres ved en informasjonsmodell. Informasjonsmodellen får en ved å videreføre informasjonsmodellen fra CIM, og detaljere denne eller gjøre om på den, slik at det er mer spesifikt mot IT-systemet som spesifiseres.

      iii. Prosessmotor –> Detaljeres ved en prosessmodell. Nært koblet opp til prosessmodellen fra CIM. Vil være resultat av transformasjon av forretningsprosesser i CIM til prosesser i PIM. Vises ved PIM4SOAs prosesselement.

      iv. Funksjonskomponent –> Denne komponenten kan bestå av mange funksjonalitetsbaserte komponenter som hver og en benytter seg av de ulike tekniske komponentene. Detaljeres ved klassediagram, sekvensdiagram eller lignende.

(b) Lag komponentinteraksjon –> Vise hvordan komponentene samhandler i henhold til de ulike funksjonalitetene. Bruk sekvensdiagram.

(c) Lag interfacemodell –> Definerer interfacene for de ulike komponentene. Kan kombineres med komponentstrukturen. Vanlig klassediagram kan brukes.

Kapittel 8

Anvendelse av REMO

I dette kapittelet skal jeg vise anvendelsen av metoden som ble presentert i forrige kapittel. Metoden er anvendt på den delen av DemoTelco-casen som har å gjøre med håndtering av ordre (se tillegg D for detaljert casebeskrivelse).

8.1 Casen

DemoTelco-casen [36] har jeg fått fra Singular gjennom INTEROP-prosjektet [80] (beskrivelse av INTEROP er i tillegg G).

DemoTelco er en del av den greske gruppen Telco. Telco er spesialiserte innen telekommunikasjon. I DemoTelco er det forretningsprosesser for håndtering av innkjøp, ordre, varehus, finans og faste ressurser. Vi skal konsentrere oss om håndteringen av ordre.

En ordre kan komme fra tre ulike kunder: butikk, franchise eller forhandler. Butikkunder er butikker eid av DemoTelco, franchisekundene er franchise-butikker i DemoTelco og forhandlere er butikker eller personer som selger DemoTelcos produkter (gjerne i tillegg til andre produkter). Når en kunde (butikk, franchise eller forhandler) begynner å gå tom for f.eks. mobiltelefoner, så sender kunden en ordre inn til DemoTelcos salgsavdeling. Salgsavdelingen sjekker tilgjengeligheten for produktet og fullfører ordren. Hvordan ordren oppfylles er litt forskjellig alt etter hva slags kunde det er. Når ordren er fullført går den til logistikkavdelingen, og logistikkavdelingen leverer produktene som er bestilt til kunden.

I DemoTelco er det seks hovedavdelinger, men det er kun tre av dem som er involvert i håndtering av ordre. Salgsavdelingen har hovedansvaret for ordren. I denne avdelingen sjekkes det om det er nok på varelager, og hvis det er nok sendes ordren videre til der den skal. Logistikkavdelingen har ansvaret for å levere elementene i ordren til kunden som har bestilt dem. Den siste avdelingen som er av relevans for ordrehåndtering er finansavdelingen. For noen kunder er kredittkontroll en del av prosessen

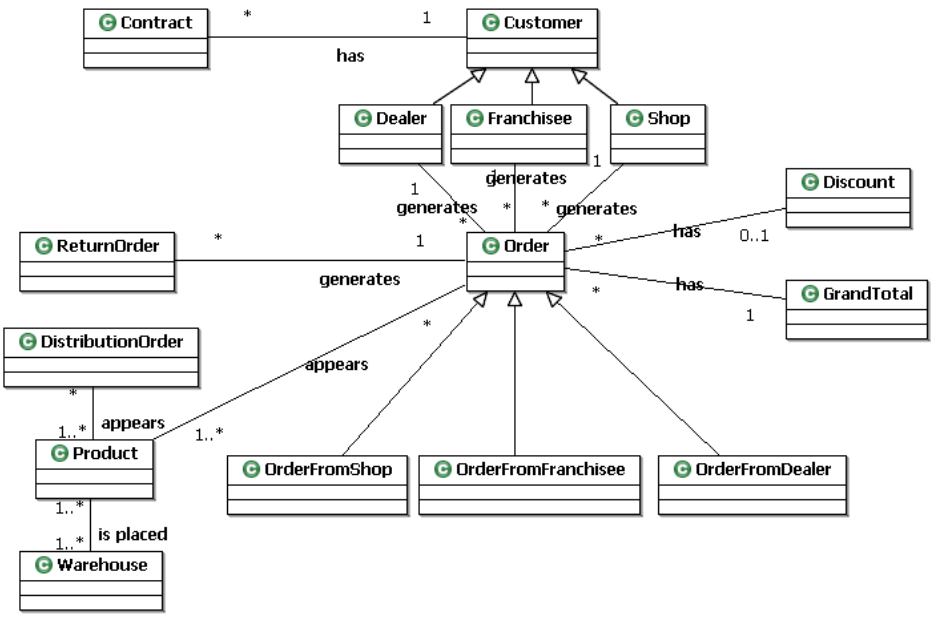
for håndtering av ordre, og kredittkontroll er det finansavdelingen som utfører.

8.2 Virksomhetsmodell og kravmodell

For å lage CIM har jeg laget en informasjonsmodell (se figur 8.1 og 8.2), et organisasjonskart (figur 8.3 på side 80), forretningsprosessmodell (se figur 8.6 til figur 8.9), forretningsregelmodell (se tabell 8.1 til figur 8.5) og en kravmodell (figur 8.12 på side 87).

8.2.1 Informasjonsmodell

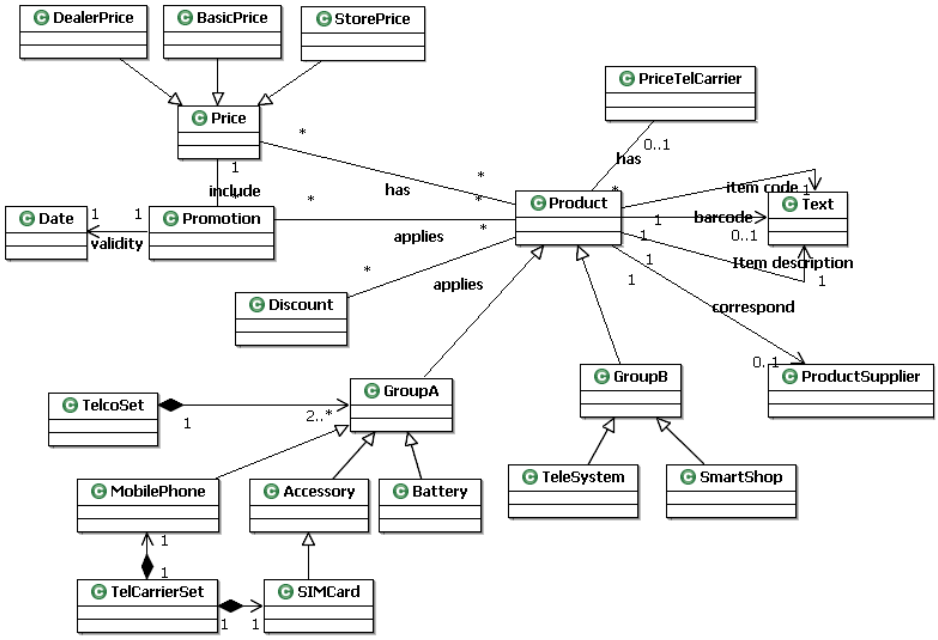
Informasjonsmodellen består her av to klassediagrammer. Det ene fokuserer på ordren (figur 8.1), og det andre tar for seg det som har å gjøre med produkt (figur 8.2 på neste side). I og med at dette er på virksomhetsnivå, er ikke informasjonsmodellene så detaljerte. Her er det tatt med de elementene som er relevante for å kunne bruke i forretningsregler og forretningsprosesser. De skiller seg noe fra hva en er vant til å tenke på som klassediagram, med f.eks. at noen av klassene ville vært attributter. Dette er fordi jeg kun ønsker å vise begrepene som er av relevans av bedriften.



Figur 8.1: Informasjonsmodell for ordre

Figur 8.1 viser informasjonsmodellen for ordre. Her ser vi at en kunde kan være av tre typer, og at hver kunde genererer mange ordre. Ordren kategoriseres videre etter hvilken kunde den kommer fra. Dette er gjort

fordi det er ulike regler som gjelder alt etter hvem ordren kommer fra. I en ordre er det mange elementer som må spesisfiseres, men jeg har valgt å kun ha med de elementene som er av relevans for forretningsreglene og prosessene. *GrandTotal* er den totale summen ordren kommer på, og *discount* er eventuell rabatt som beregnes fra *GrandTotal*.

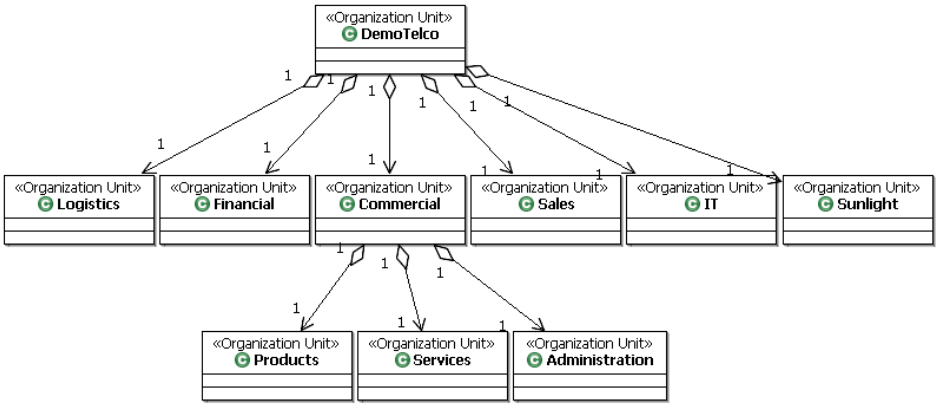


Figur 8.2: Informasjonsmodell for produkt

I figur 8.2 er informasjonsmodellen for produkt vist. I denne modellen er det vist hvilke produkter som DemoTelco har. Produktene kan deles i to grupper, gruppe A og gruppe B, hvor gruppe A er konkrete produkter, mens gruppe B er systemer. Et produkt kan ha flere priser. Det kan være *DealerPrice*, *BasicPrice* eller *StorePrice*, og prisene settes i henhold til gitte forretningsregler (forretningsreglene vises senere). Noen detaljer for produktet er vist, og det kan diskuteres om noen av detaljene hører hjemme under en plattformuavhengig modell. Dette er elementer som *item code*, *barcode* og *item description*. Jeg har valgt å ta dem med her for å vise hvordan integritetsregler illustreres i informasjonsmodeller. "A product has exactly one item code" er en integritetsregel som er vist i denne informasjonsmodellen. Den vises ved bruk av kardinalitet og roller.

8.2.2 Organisasjonsmodell

Telco er en stor organisasjon, og det er denne DemoTelco er en del av, men i denne CIM har jeg kun vist hvordan DemoTelco er organisert (se figur 8.3 på neste side). Jeg anså det ikke som relevant å ta med hele den store organisasjonsmodellen, i og med at det kun er ordreprosessen



Figur 8.3: Organisasjonsmodell

i DemoTelco som er i fokus for modelleringen.

8.2.3 Forretningsregelmodell

Forretningsreglene i DemoTelco har vært vanskelige å identifisere. Etter samtaler med personer i Singular har jeg kommet frem til noen som er relevante for håndtering av ordre. I figur 8.1 er en beslutningstabell for rabatt vist. Kunder kan få rabatter på ordren sin. Rabatten blir satt etter bestemte regler i DemoTelco. Dersom f.eks. den totale summen er større enn 5000, og mindre enn 10000 og kunden er butikk, så vil det være en 10% rabatt. Den neste beslutningstabellen er for å bestemme leveransedato. Dersom kunden er en butikk, og butikken er i Sofia, så skal leveransedatoen for ordren settes til 3 dager etter at ordren ble sendt inn til salgsavdelingen. Begge disse beslutningstabellene vil være en del av ordrehåndteringsprosessene, og sammenhengen mellom prosesser og regler er vist i figur 8.11 på side 86 og figur 8.10 på side 86.

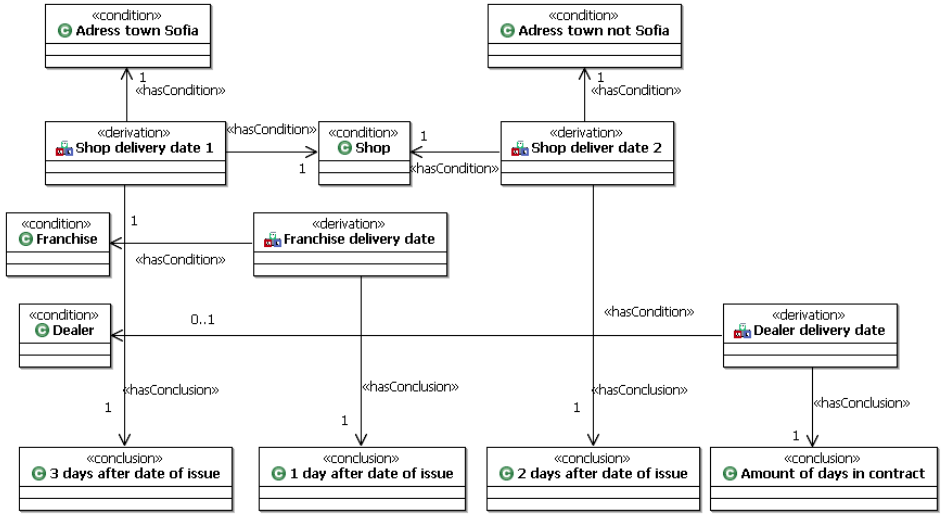
1. Grand Total	GT<=5000			5000<GT<=10000			GT>10000		
2. Customer	Shop	Franch.	Dealer	Shop	Franch.	Dealer	Shop	Franch.	Dealer
1. Discount 5%	x	-	-	-	-	-	-	-	-
2. Discount 8%	-	-	x	-	-	-	-	-	-
3. Discount 10%	-	-	-	x	x	-	-	x	-
4. Discount 14%	-	-	-	-	-	x	-	-	-
5. Discount 15%	-	-	-	-	-	-	x	-	-
6. Discount 18%	-	-	-	-	-	-	-	-	x

Tabell 8.1: CIM Beslutningstabell for rabatt

1. Customer Type	Shop		Franchiser	Dealer
2. Address town is Sofia	Yes	No	-	-
1. Delivery date is 3 days after date of issue	x	-	-	-
2. Delivery date is 2 days after date of issue	-	x	-	-
3. Delivery date is 1 days after date of issue	-	-	x	-
4. Delivery date is amount of days in contract	-	-	-	x

Tabell 8.2: CIM Beslutningstabell for leveransedato

Når det gjelder rabatt for distriubsjon (som vi senere vil se er indirekte vist i forretningsprosessmodellen), så fant jeg aldri ut av disse, selv etter samtaler med personer i Singular. Jeg fikk kun eksempler på hvordan det skulle gjøres. Rabatt for distribusjon er en helt klart en forretningsregel som kunne vært illustrert i en beslutningstabell.



Figur 8.4: Forretningsregler for å bestemme leveransedato

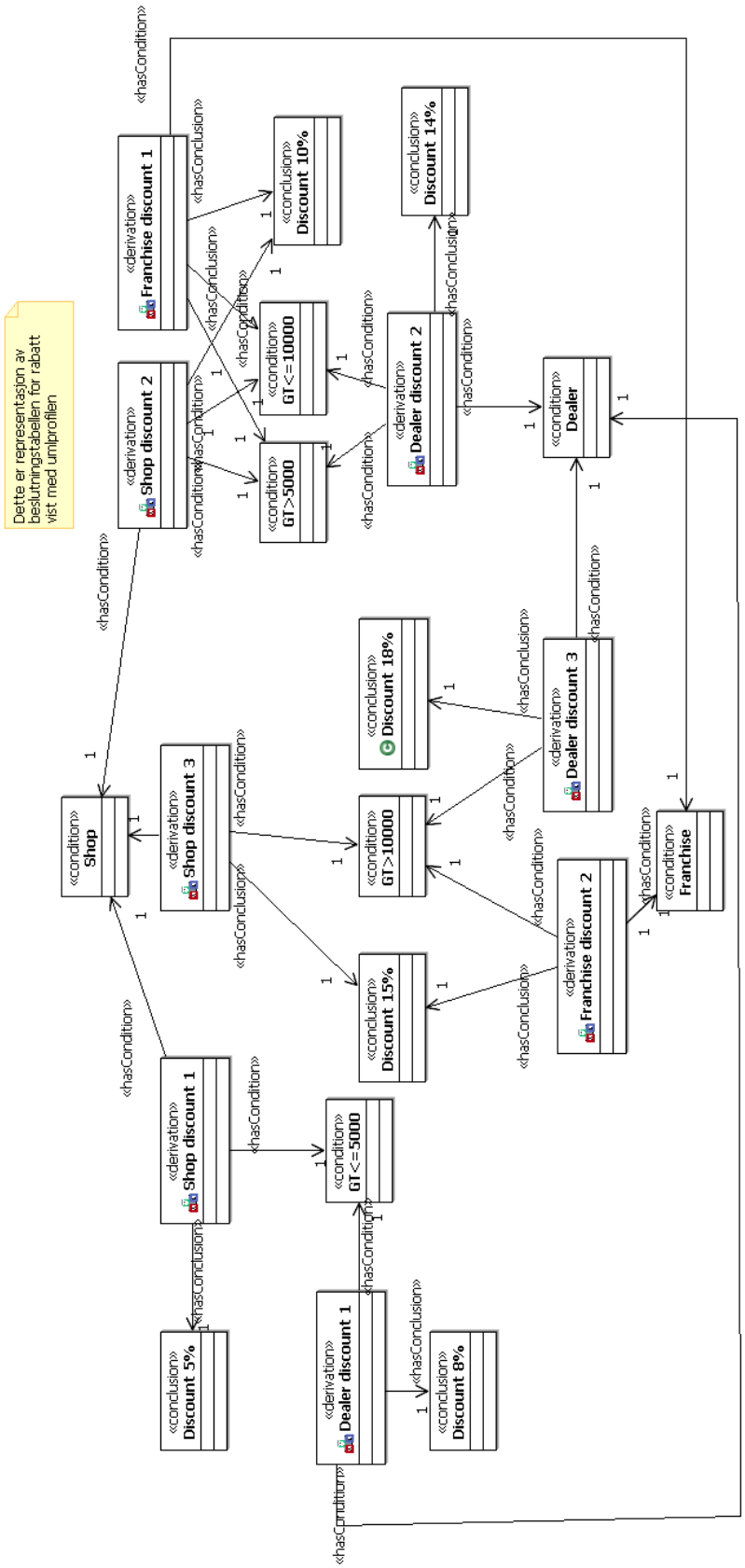
Etter at beslutningstabellene var ferdigstilte, gjorde jeg dem om til regler representert med en UML-profil. Profilen laget jeg i RSM, og er implementert i henhold til metamodellen definert i forrige kapittel. I figur 8.4 og figur 8.5 er forretningsreglene modellert med UML-profilen vist.

8.2.4 Prosessmodell

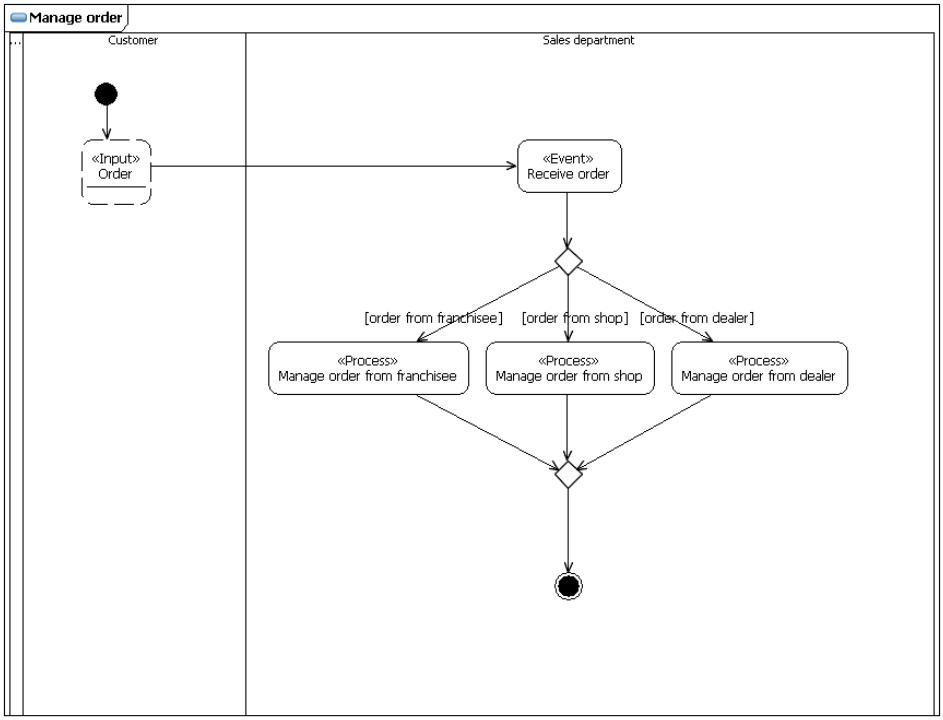
Prosessmodellen består av fire modeller. Her har jeg brukt UML 2.0 aktivitetsdiagram med UML-profilen for POP*.

For ordrehåndteringen er det en overordnet prosess. Kunden sender en ordre til salgsavdelingen, og salgsavdelingen håndterer den. Deretter håndteres ordren etter hva slags kunde det er som sender den. I figur 8.6 på side 83 er denne prosessen vist.

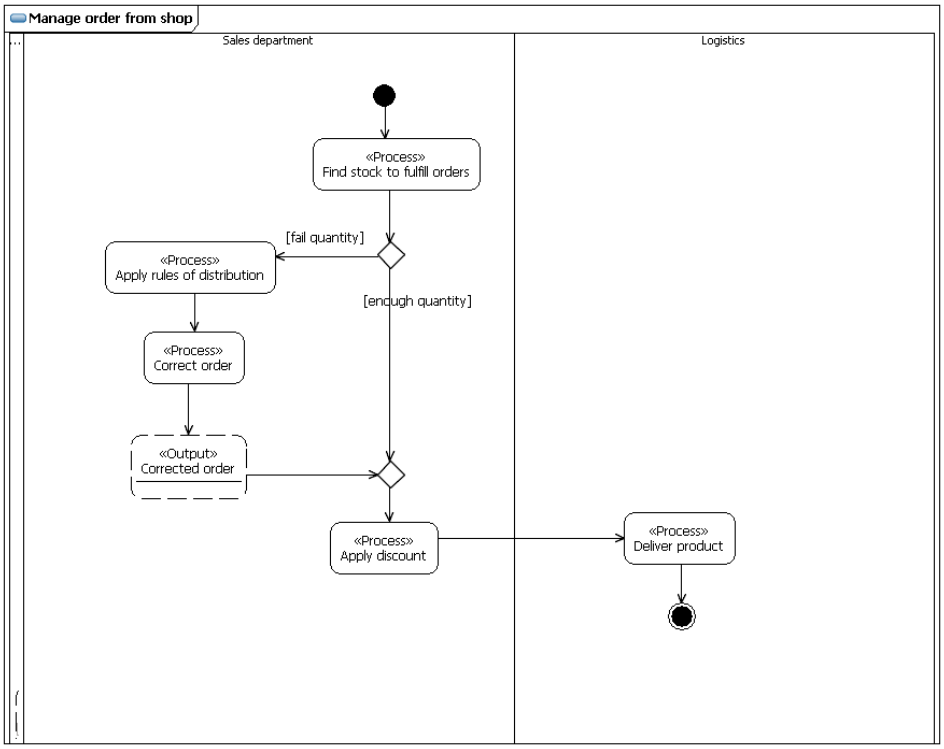
Dersom det er en ordre fra en butikk, vil prosessen vist i figur 8.7 på side 83 inntreffe. Først vil salgsavdelingen sjekke om det er nok på varelager for å fullføre ordren. Dersom det ikke er nok på varelager vil gitte regler for distribusjon bli anvendt. Det som da skjer er at alle ordre vil bli samlet, og varene som er tilgjengelig vil bli fordelt proporsjonalt med hva de ulike har bestilt. Dette er regelen som jeg aldri fikk tak på helt konkret, som nevnt tidligere, men er illustrert som en del av forretningsprosessen. Etter at varen har blitt ordnet vil regler for rabatt legges til, og så vil ordren sendes over til logistikkavdelingen som leverer ordren bestilt, til kunden.



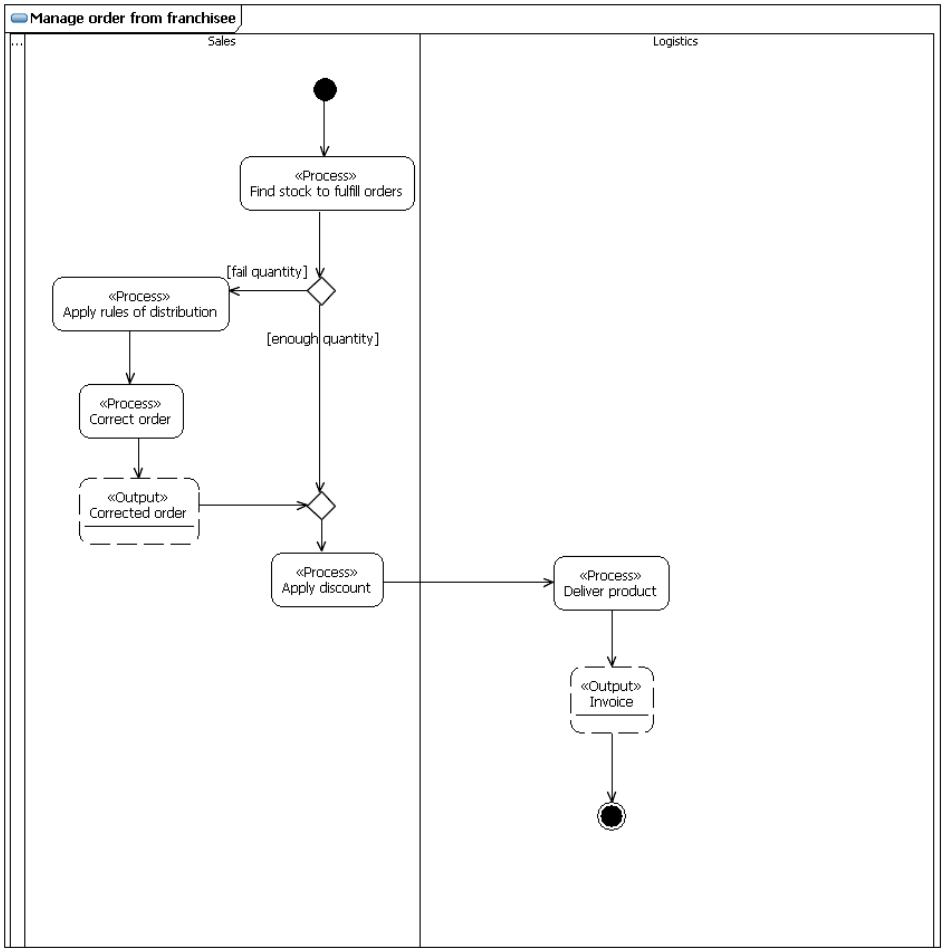
Figur 8.5: Forretningsregler for rabattberegning



Figur 8.6: CIM Processmodel for prosessen "manage order"

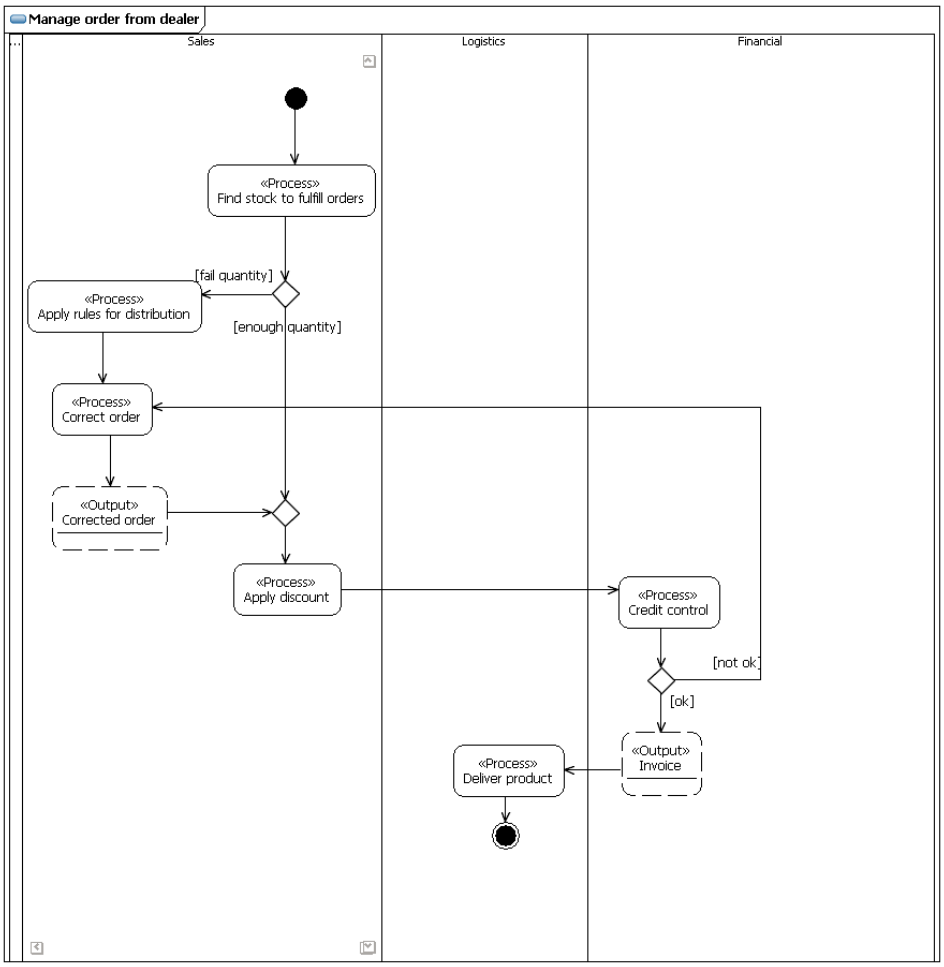


Figur 8.7: CIM Processmodel for prosessen "manage order from shop"



Figur 8.8: CIM Processmodel for prosessen "manage order from franchisee"

Når det er en ordre fra en franchise vil prosessen være omtrent lik som for butikk. Den største forskjellen er en faktura som vil bli lagt ved fra logistikkavdelingen til franchisebutikken. Betaling av varene en kunde bestiller er annerledes når denne kunden er franchise. Prosessmodellen er vist i figur 8.8 på forrige side.

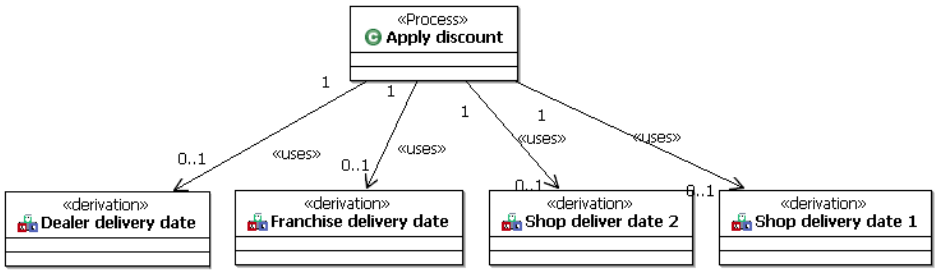


Figur 8.9: CIM Processmodel for prosessen “manage order from dealer”

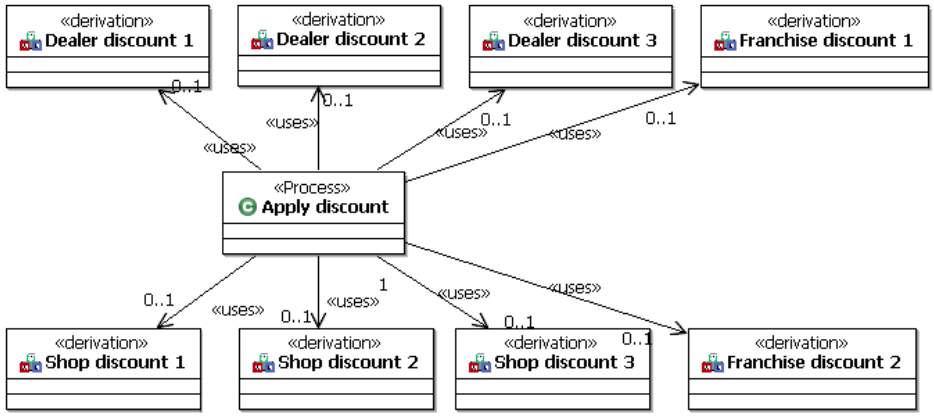
Prosessen for å håndtere ordre fra forhandlere (figur 8.9) er litt ulik ordrehåndtering for de andre kundene. For forhandlere må også en kredittkontroll kjøres på ordren. Dersom denne kredittkontrollen ikke er ok, så må ordren justeres.

For å vise hvordan forretningsreglene og forretningsprosessene henger sammen har jeg laget to klassediagram som viser det. I figur 8.10 på neste side er sammenhengen mellom reglene for leveransedato og prosessen vist. Her ser vi hvordan prosessen apply discount bruker disse reglene. Grunnen til at leveransedatoen blir satt først under “apply discount” er fordi at da er ordren korrekt i henhold til hva som er tilgjengelig på varelager. I figur 8.11 på neste side er sammenhengen mellom rabattreglene og prosessen “apply discount” vist. Reglene vises her med bare navnet på

dem, og selve regelen er illustrert i regelmodellen. I begge figurene kan en se at prosessen bruker reglene ved at assosiasjonen har fått stereotype "uses" satt på. Dermed fremgår dette tydelig.



Figur 8.10: Sammenheng mellom prosess og regler for leveransedato

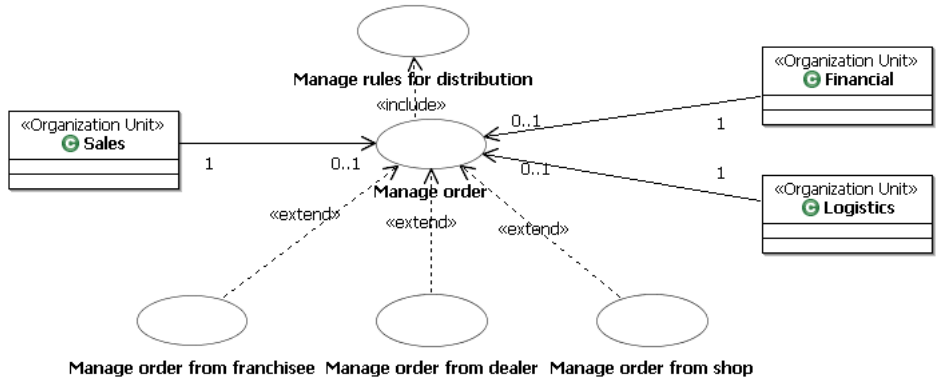


Figur 8.11: Sammenheng mellom prosess og regler for rabatt

8.2.5 Kravmodell

Kravmodellen har jeg valgt å utforme på et overordnet nivå. Ordrehåndteringsprosessen er ment å være en liten bit av et større system. For ordrehåndtering er det ønskelig med et system som kan gjøre det enklere å håndtere de tre variantene av ordre. Dette er vist ved et business use case diagram i figur 8.12 på neste side.

I dette use case diagrammet er hvert use case i samsvar med hver prosess beskrevet gjennom prosessmodellen. Dette diagrammet viser også hvordan de ulike organisasjonsenhetene henger sammen med systemets funksjonalitet. *Logistics* og *financial* kan sees på som sekundære aktører. Salgsavdelingen er de som har hovedansvaret.



Figur 8.12: CIM Business Use Case Model

8.3 Transformasjon fra CIM til PIM

REMO gir foreløpig kun regler for transformasjon av forretningsregler. Transformasjon av forretningsprosessmodellen er dermed utelatt i denne anvendelsen.

Transformasjonen av modellene ble gjort i verktøyet XMF-Mosaic. Transformasjonsreglene som ble anvendt er beskrevet i kapittel 7 og i tillegg H. De andre modellene er blitt videreført slik de er til PIM, med unntak av organisasjonsmodellen. Organisasjonsmodellen har jeg utelatt i PIM.

All modellering for CIM er gjort i RSM, og på grunn av manglende utvekslingsformat mellom RSM og XMF-Mosaic var jeg nødt for å tenge forretningsregelmodellen inn i XMF-Mosaic. Deretter anvendte jeg transformasjonsreglene som tidligere definert. Ved å anvende disse transformasjonsreglene på forretningsregelemodellen, har jeg fått ut deler av arkitekturmodellen, i hvert fall detaljeringen av regelkomponenten.

8.4 Plattformuavhengig modell

Den plattformuavhengige modellen skal hovedsakelig bestå av modeller av tre forhåndsbestemte komponenter (regelkomponent, prosesskomponent og informasjonskomponent), eventuelle funksjonalitetskomponenter og ulike modeller som viser hvordan det hele henger sammen. Informasjon, regler og prosesser er elementer som er med fra CIM. Informasjonsmodellen på CIM brukes direkte videre, men med eventuelle utvidelser ettersom mer informasjon blir oppdaget på PIM nivå. Regelmodellen er resultatet av at forretningsregelmodellen i CIM er transformert til produksjonsregler i PIM. Prosessmodellen skal i likhet med regelmodellen, være resultat av transformasjon fra CIM til PIM, men her skal resultater fra ATHENA anvendes når de foreligger. Komponentene presenteres her, og også deres sammenheng presentert med komponentstrukturmo-

dell, komponentinteraksjonsmodell og interfacespesifisering. Funksjonaliteten til systemet er vist i en funksjonskomponent. Denne komponenten ville blitt mye større, eller til flere, dersom hele DemoTelco-casen hadde vært modellert, men nå er det bare en funksjon den skal tilby, og er også derfor kun én komponent.

8.4.1 Komponentstruktur og interfacespesifisering

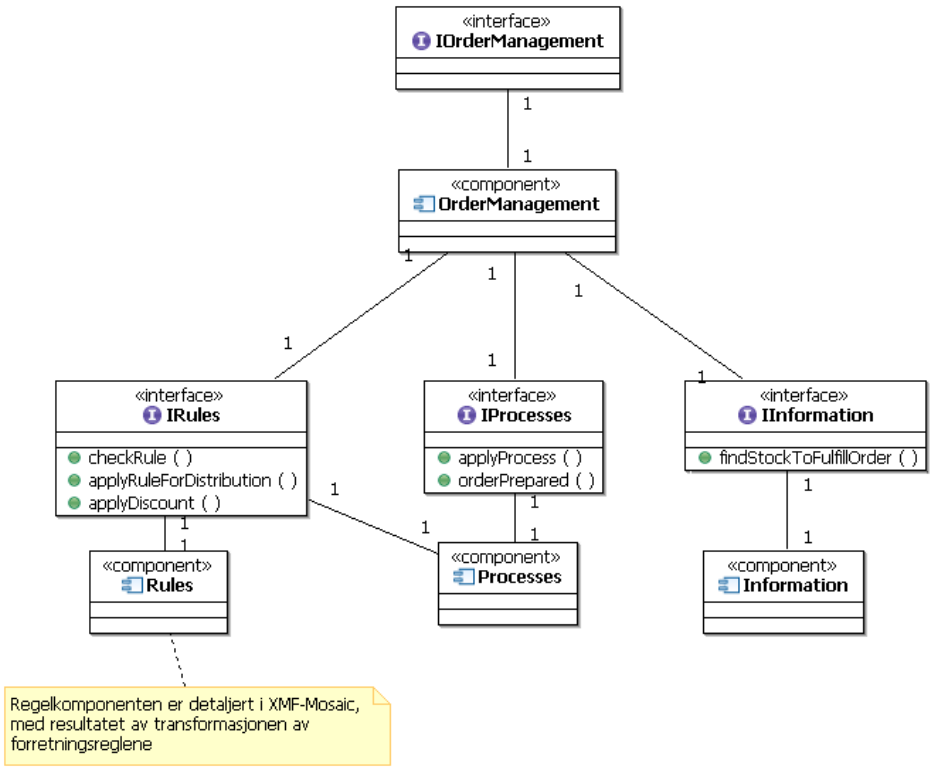
Komponentstrukturen (figur 8.13 på neste side) viser hvilke komponenter det vil være i arkitekturen. Det er laget fire komponenter, hvor tre av dem var delvis gitt av metoden (regelkomponenten, prosesskomponenten og informasjonskomponenten). Det vil si at i REMO er det definert at det skal være en regelmotor, prosessmotor og en database, og da må også sammenhengen mellom disse komponentene vises i komponentstrukturen. For hver komponent er det lagt på et interface, som skal ta seg av all kommunikasjon med andre komponenter. Slik er interfacemodellen innbakt i komponentstrukturen. I en større case ville det nok vært lurt å vise detaljene til interfacene i et eget diagram, men i og med at det er såpass enkelt her så er det tatt med i komponentstrukturdiagrammet.

Regelkomponenten er den som skal håndtere alle regler i IT-systemet, og detaljene for denne vises i figur 8.15 på side 91 og i tillegg H. Helst burde reglene vært vist i RSM, slik at det var mulig å visualisere det i selve komponenten, men på grunn av manglende utvekslingsformat mellom XMF-Mosaic og RSM, var ikke dette mulig. Dermed er regelkomponenten detaljert med det som ble transformert fra CIM-nivå. På den måten er også forretningsreglene brukt direkte videre i systemspesifikasjonene, og samtidig holdes reglene separat fra resten av systemet.

Regelkomponenten er koblet til prosesskomponenten gjennom deres tilhørende interface. Dette er for at det skal bli enklest mulig å legge inn regler som en del av prosessene. Prosessen skal ha mulighet for å gjøre bruk av regler, og reglene skal ha mulighet for å sette i gang prosesser.

8.4.2 Komponentinteraksjon

I komponentinteraksjonsdiagrammet (figur 8.14 på side 90) vises det hvordan komponentene samspiller. Her er det detaljert hva som skjer når en ordre skal håndteres. Interaksjonen viser bare hva som skjer på et relativt overordnet nivå siden komponentene i seg selv har en del logikk som legges på. En av fordelene med et slikt komponentinteraksjonsdiagram er at man får abstrahert noe, og detaljene vises heller i komponentene.



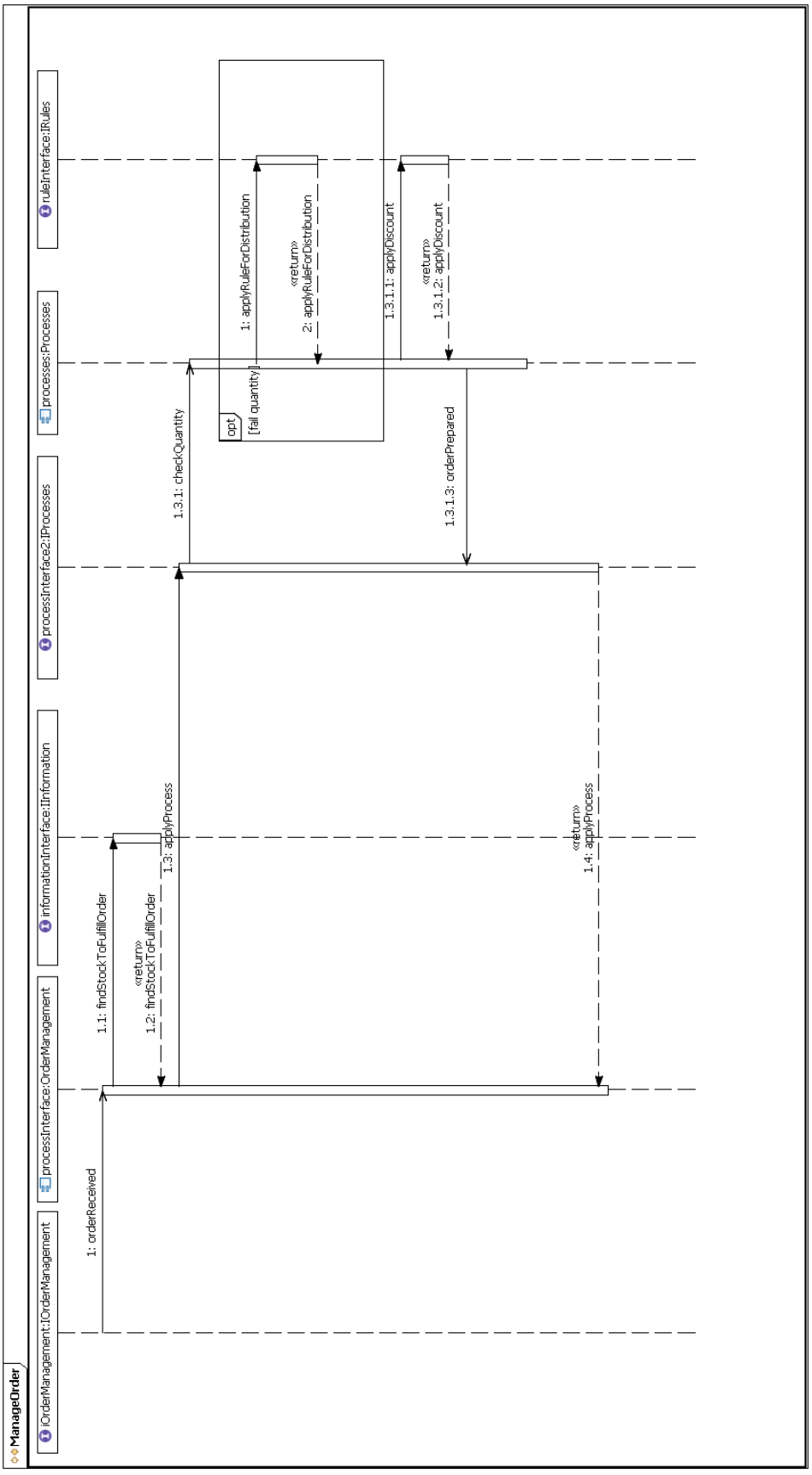
Figur 8.13: Komponentstruktur og interfacespesifisering

8.4.3 Regelkomponent

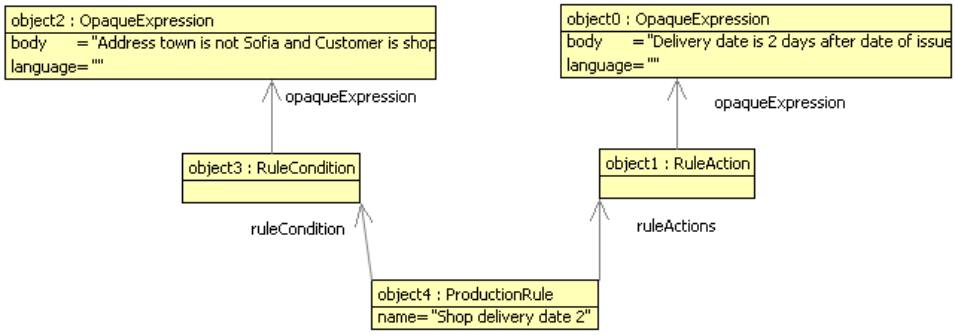
Forretningsregelmodellen (som er på CIM-nivå) består av to modeller vist i figur 8.4 på side 81 og figur 8.5 på side 82. Disse modellene har jeg tegnet inn i XMF-Mosaic, og transformert derfra. (Alle modellene er vist i tillegg H.) Resultatet av transformasjonen med reglene som er gjengitt over ble en modell bestående av produksjonsregler. I figur 8.15 på side 91 er transformasjonen av forretningsregelen *ShopDeliveryDate2* vist. Der ser vi hvordan den har blitt til en regel av type *ProductionRule*, med samme navn som forretningsregelen hadde. Produksjonsregelen har en betingelse og en hendelse tilknyttet seg. Betingelsen består av en *OpaqueExpression* som har *body* med innholdet som tilsvarer konkateneringen av de forretningsregelens betingelser. Attributtet *language* vil typisk være et element som settes i den plattformspesifikke modellen.

8.4.4 Prosesskomponent

Prosesskomponenten skal detaljeres på bakgrunn av transformasjon av forretningsprosesser fra CIM-nivå. Siden REMO satser på å bruke resultatene fra ATHENA når de foreligger, så er ikke denne transformasjonen gjort. Dersom den transformasjonen hadde blitt gjort, ville prosesskopmonenten



Figur 8.14: Komponentinteraksjonsmodell



Figur 8.15: Produksjonsregelrepresentasjon av en forretningsregel

bestått av disse resultatene, men med utvidelser for å håndtere bruken av regler o.l. i samsvar med komponentinteraksjonsmodellen.

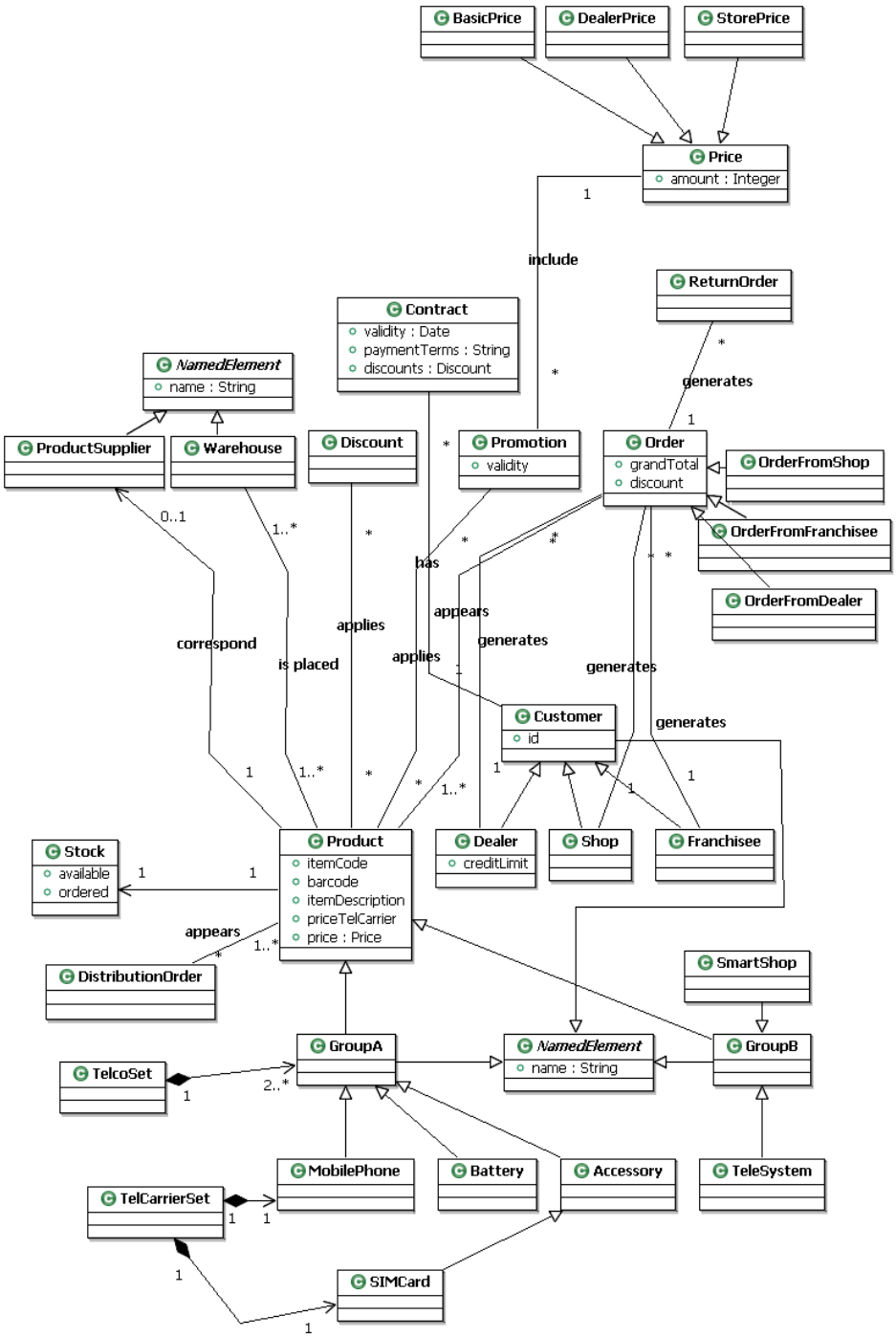
8.4.5 Informasjonskomponent

For informasjonskomponenten har jeg videreført informasjonsmodellen fra CIM. På PIM-nivå har jeg utvidet denne, slik at den blir mer fokusert mot systemet, og dataene som til slutt vil lagres i en database. Det jeg har gjort er å ta noen av *has*-relasjonene og gjøre dem om til attributter i klassen hvor relasjonen gikk ut i fra. Informasjonsmodellen er vist i figur 8.16. Modellen viser også at flere attributter er lagt til. Dette er elementer som er med på å detaljere begrepene som tidligere ble identifisert. Se f.eks. på klassen *order*. Denne er nå detaljert med *grandTotal* og *discount*, som er to attributter nødvendig for å få med i henhold til hva som er nevnt i casebeskrivelsen. I tillegg er det lagt til en abstrakt klasse *NamedElement*, som mange andre klasser er satt til å arve fra, slik at de kan ha navn. Klassen *NamedElement* er vist to plasser i modellen for å unngå at modellen blir for uoversiktlig.

Denne informasjonsmodellen krever noe videre arbeid slik at den blir detaljert tilstrekkelig, slik at databasen kan implementers til slutt. Informasjonsmodellen i PIM har, som nevnt i avgrensningen i kapittel 2, ikke vært fokus for metoden.

8.4.6 Funksjonsmodell

Funksjonaliteten i systemet består kun av en ting, nemlig det å håndtere ordre. Denne har jeg ikke detaljert noe mer, i og med at denne kun skal håndtere kommunikasjonen mellom de andre komponentene foreløpig. I det endelige systemet vil det være denne komponenten brukeren vil snakke mot, brukeren vil ikke se direkte de andre komponentene som ligger under. Tjenestekomponenten fremgår av komponentstrukturdiagrammet i figur 8.13 på side 89, og også noe fra interaksjonsmodellen i figur 8.14 på forrige side.



Figur 8.16: Informasjonsmodellen i PIM

Kapittel 9

Evaluering av REMO

Metoden presentert i kapittel 7 er laget med den hensikt å oppfylle alle krav som ble stilt i kapittel 3 og i kapittel 6. Metoden er anvendt på DemoTelco-casen og denne anvendelsen er grunnlaget for evalueringen som gjøres i dette kapittelet. Evalueringen er gjort på tilsvarende måte som i kapittel 5, hvor eksisterende løsninger ble evaluert.

9.1 Evaluering av CIM

For å evaluere CIM i REMO skal to sett av krav sees på. Ett av disse kravene, det om forretningsregelmodell, er blitt detaljert til et nytt sett av krav i kapittel 6. Evaluering i henhold til første sett med krav er oppsummert i tabell 9.1. REMO ble utformet på bakgrunn av disse kravene, og derfor bør alle krav være oppfylte. Vi skal allikevel gå gjennom hvert krav for å vise hvordan og i hvilken grad det er oppfylt.

Krav til CIM	**REMO**
CIM1: Brukervennlighet	2
CIM2: Verktøystøtte	1
CIM3: Informasjonsmodell	2
CIM4: Organisasjonsmodell	2
CIM5: Prosessmodell	2
CIM6: Funksjonalitetsbeskrivelse	1
CIM7: Forretningsregelmodell	2

Tabell 9.1: Krav til CIM

9.1.1 Brukervennlighet

Inntrykket fra casen er at REMO er forståelig og enkel å bruke. Det er lagt vekt på å bruke enkle diagrammer for å fremstille de ulike elementene. Ved å hovedsakelig bruke klasser, stereotypede klasser og aktivitetsdiagram

er det ikke så mange forskjellige modellelementer å holde oversikt over. Det gjenstår å prøve dette ut overfor forretningsfolk for å få bekreftet brukervennligheten. Imidlertid tilsier erfaringen fra casen at REMO bør ha høy skår for brukervennlighet.

9.1.2 Verktøystøtte

Alle metoder vi har sett på, har hatt verktøystøtte, men den har ikke alltid vært tilfredsstillende. Dette gjelder også for REMO på CIM-nivå. Her har vi støtte i verktøy ved bruk av UML-profiler kombinert med vanlig UML, men det er nødvendig med et verktøy som samler alt sammen på en oversiktlig måte, det vil si CIM, PIM og transformasjonsverktøy, og som også tilbyr bruk av beslutningstabeller for forretningsregler. Dette er et element som bør sees på i videre arbeid.

9.1.3 Informasjonsmodell

Informasjonsmodellen består av et klassediagram hvor all informasjon relevant for bedriften er modellert. Ved å bruke klassediagram kommer informasjon klart og tydelig frem. Informasjonsmodellen krever ikke noe bruk av UML-profil, da det er tilstrekkelig med et vanlig UML-klassediagram, noe som også kan være en fordel.

9.1.4 Organisasjonsmodell

Organisasjonsmodellen er et UML-klassediagram med tilhørende UML-profil. I REMO er UML-profilen for POP* anvendt. Det å vise organisasjonen til en virksomhet er veldig enkelt, da dette ofte er fastlagt og ferdig nedskrevet i forretningspolicies, eller andre dokumenter som beskriver virksomheten. I REMO er det også enkelt å få det fram ved et stereotypet klassediagram. Kravet om organisasjonsmodell er oppfylt.

9.1.5 Prosessmodell

For å lage prosessmodell er POP* valgt som grunnlag i REMO, og her er også UML-profilen for POP* anvendt. Ved å bruke et aktivitetsdiagram med tilhørende stereotyper får man beskrevet forretningsprosessene på en enkel og klar måte. Ved å bruke et aktivitetsdiagram kan en enkelt også vise eventuelle deler av organisasjonsmodellen som kan være en del av prosessen. Prosessmodellen er tydelig adskilt fra det andre, samtidig som det fremgår hvilke andre elementer som er involvert i prosessen. Kravet om prosessmodell er tydelig oppfylt i REMO.

9.1.6 Funksjonalitetsbeskrivelse

Funksjonalitetsbeskrivelse er ikke vektlagt i stor nok grad i REMO. Denne delen er i REMO kalt for kravmodell. Det er mer opp til brukeren eller utvikleren å velge hvordan funksjonaliteten skal beskrives. REMO gir veiledning på at en skal beskrive noe, men ikke i nok detalj. REMO kunne nok gitt en bedre veiledning på koblingen mellom virksomhetsmodellene og kravmodellen. Derfor får REMO kun 1 for funksjonalitetsbeskrivelse.

9.1.7 Forretningsregelmodell

Forretningsregelmodellen er bra i REMO. Forretningsregelmodellen må evalueres nærmere i henhold til de nye kravene som ble stilt i kapittel 6. Der ble det stilt krav om at en forretningsregelmodell skulle inneholde prosessorienterte forretningsregler og slutningsregler, og at reglene skulle visualiseres. Fra forretningsregelmodellen skulle det være gode muligheter for å transformere til produksjonsregler, og det ble også her stilt krav om god støtte i verktøy for forretningsreglene. Kravene her var et viktig grunnlag for utviklingen av REMO. Evalueringen er oppsummert i tabell 9.2 på neste side.

Prosessorienterte forretningsregler og slutningsregler er de to typer av forretningsregler som REMO har i en egen modell. Slutningsregler er enkle og klare regler, som er lette å identifisere og representere i forretningsregelmodellen. REMO får derfor 2 for krav CIM7FR2. De prosessorienterte reglene er også klart skilt ut i REMO, men i casestudien var det vanskelig å identifisere prosessorienterte regler, og det er derfor ikke grunnlag for å gi denne skår 2. Prosessorienterte regler er altså til stede i REMO, men det er behov for andre casestudier for å kunne gi dette kravet 2.

Et av kravene som er stilt spesielt for forretningsregler er visualisering. Forretningsregler er vanskelige å fremstille visuelt. Måten det er løst på i REMO er å først representere reglene med tekst og beslutningstabeller, og deretter oversette dem og representere dem i stereotypede klassediagram i henhold til en gitt UML-profil for forretningsregler. Dette gjør det enklere enn ved bare å gjøre det tekstlig. Når man lager en beslutningstabell kommer man lettere fram til de ulike alternativene som en slutningsregel skal presentere.

Kravet om transformasjonsmulighet er oppfylt i REMO. Metamodellen som er definert for forretningsregler, er enkel å mappe til metamodellen for produksjonsregler, og dermed er transformasjon av forretningsregelmodellen enkel.

REMO har til en viss grad verktøystøtte for å spesifisere forretningsregler. Ved å bruke UML-profilen definert i RSM kan man uttrykke alle reglene som man har kommet frem til. Det en må passe på, er at modellen lett kan bli stor og kompleks, så derfor er det lurt å dele forretningsregelmodellen

opp i flere diagram. Det mangler verktøystøtte for å lage beslutningstabeller. I casestudiet ble beslutningstabellene gjort med penn og papir, noe som viste seg greit for å få oversikt over de ulike utfallene av reglene. Deretter oversatte jeg beslutningstabellene til klassediagram med bruk av stereotypene definert i UML-profilen for forretningsreglene. REMO har ikke noe komplett verktøy som kan håndtere alle delene i metoden, det vil si både CIM, PIM og transformasjon. For selve forretningsreglene er det greit nok, med tanke på å uttrykke regler slik som det er definert ved metamodellen, men det å transformere regler som er definert i RSM med å bruke XMF-Mosaic er relativt tungvint. Jeg har derfor gitt kravet om verktøystøtte skår 1.

Krav til CIM7 - Forretningsregelmodell	**FR i REMO**
CIM7FR1: Prosessorienterte forretningsregler	1
CIM7FR2: Slutningsregler	2
CIM7FR3: Visualisering	1
CIM7FR4: Transformasjonsmulighet	2
CIM7FR5: Verktøystøtte	1

Tabell 9.2: Krav til CIM7 - Forretningsregelmodell

9.2 Evaluering av plattformuavhenig modell

Den plattformuavhengige modellen skal beskrive arkitekturen til IT-systemet som er under utvikling. Det var kun ordrehåndteringsprosessen som har vært grunnlag for evaluering av REMO på den plattformuavhengige modellen. Det første som ble gjort var å transformere de modellene som var mulige å transformere, i dette tilfellet var det kun forretningsregelmodellen i CIM. Etter det var gjort ble det laget en komponentstruktur og komponentinterksjonsdiagram. Fra disse modellene ble interfacemodellen laget, og informasjonsmodellen ble detaljert. Tabell 9.3 oppsummerer evalueringen.

Krav til PIM	**REMO**
PIM1: Komponentstruktur	2
PIM2: Komponentinteraksjon	2
PIM3: Funksjonsmodell	2
PIM4: Informasjonsmodell	2
PIM5: Regelmodell	2
PIM6: Prosessmodell	1
PIM7: Interfacespesifisering	2
PIM8: Verktøystøtte	1

Tabell 9.3: Evaluering av REMO på PIM-nivå

9.2.1 Komponentstruktur og -interaksjon

Komponentstrukturmodellen var enkel å få på plass. Dette var både fordi at noen av komponentene allerede var gitt, og på grunn av at IT-systemet som ble modellert hadde veldig lite funksjonalitet. I komponentinteraksjonsmodellen ble forretningsprosessene spesifisert i CIM, brukt som grunnlag. Ut fra prosessen analyserte jeg hvordan komponentene ville bruke hverandre. Resultatet ble at interaksjonsmodellen ble relativt lik forretningsprosessene. Etter å ha laget interaksjonsmodellen, måtte jeg revurdere komponentstrukturen. Spørsmålet som jeg stilte meg selv da var om ordrehåndteringskomponenten egentlig var nødvendig, eller om regel- eller prosesskomponenten kunne overta det lille ansvaret som ordrehåndteringskomponenten hadde. Resultatet ble at ordrehåndteringskomponenten ble stående siden det jeg modellerte skulle bli en del av et større system når eventuelt hele DemoTelco-casen skulle realiseres.

Begge kravene er på bakgrunn av dette gitt 2, fordi de er klart til stede, og har samtidig en viktig funksjon.

9.2.2 Interfacespesifisering

Kravet om interfacespesifisering har jeg også gitt 2. Ved å lage komponentinteraksjonsmodellen kommer det frem hva som er nødvendig i et interface. Dette er meget enkelt i RSM fordi hver melding som tegnes inn i sekvensdiagrammet blir opprettet som en metode i det objektet som meldingen går til. Dersom meldingen går til et interface blir interfacet detaljert med den metoden.

9.2.3 Funksjonsmodell

Funksjonsmodellen har jeg ikke godt nok grunnlag for å vurdere. Tjenestemodellen skal være en samling av den funksjonaliteten som systemet skal tilby. Siden det i anvendelsen kun ble vist for en funksjon (håndtere ordre), så er det ikke grunnlag nok for å si så mye om tjenestemodellen. Kravet om funksjonsmodell får 2, selv om det burde vært undersøkt for et større system, men i anvendelsen var i hvert fall funksjonsmodellen til stede.

9.2.4 Informasjonsmodell

Informasjonsmodellen kommer tydelig frem. Denne modellen er i anvendelsen omtrent den samme som for CIM. Kravet om informasjonsmodell er oppfylt. Det må sies at det ble mye enklere å lage informasjonsmodellen i PIM, når det allerede var påstartet i CIM.

9.2.5 Prosessmodell

Prosessmodellen er her ikke spesifisert i og med at resultatene fra ATHENA skal brukes for å gjøre denne transformasjonen. Selv om ikke denne modellen er blitt detaljert i casestudiet, så er kravet oppfylt. Det er en prosessmodell til stede som en del av PIM, slik det er definert i REMO. Kravet om prosessmodell får derfor 1, siden prosessmodellen er delvis til stede, men kan ikke utprøves godt nok før resultatene fra ATHENA foreligger.

9.2.6 Regelmodell

Produksjonsregelmodellen kommer tydelig frem i og med at forretningsreglene er transformert til produksjonsregler. En mangel her er at det ikke er lenket godt nok opp til informasjonsmodellen, men det er et element som kan sees på i videre arbeid. Selve transformasjonen av reglene ble gjort i XMF-Mosaic, og det gjorde det litt tungvindt, når da alle andre modeller som ble laget i PIM, ble laget i RSM. Dersom en kunne ha utvekslet modeller mellom disse to verktøyene så ville det ha vært veldig bra. Kravet om regelmodell får 2, da reglene er til stede og komponenten detaljert, men også her gjenstår det å prøve det ut på et større system.

9.2.7 Verktøystøtte

Verktøystøtte gir jeg bare 1. Det er støtte for alt vi ønsker å gjøre, men det er en generell tendens at ikke alt er direkte støttet i verktøy. Det bør lages et verktøy for hele REMO, som håndterer CIM, transformasjon fra CIM til PIM og PIM. Da først vil kravet om verktøystøtte være 2. Problemene har vært det at en hele tiden må oversette modeller, slik at de kan brukes i ulike verktøy.

9.3 Evaluering av modelltransformasjon

I anvendelsen av min metode, presentert i forrige kapittel, har jeg valgt å bruke XMF-Mosaic. Grunnen til at det ble valgt er fordi XMF-Mosaic er bedre for å vise tankegangen bak modelltransformasjonen. Evalueringen av XMF-Mosaic i henhold til kravene stilt i kapittel 3 er allerede gjort, det ble gjort i kapittel 5. Den er ikke forandret. Jeg vil derfor her si mer generelt om de erfaringene som er gjort med XMF-Mosaic og modelltransformasjon for DemoTelco-casen.

Transformasjonsreglene som er definerte er enkle å bruke. Disse kan lett tilpasses til andre transformasjonsverktøy, dersom det er ønskelig. Så lenge

transformasjonsverktøyet er metamodellbasert og har syntaks lik XMF-Mosaic (eller QVT), er det bare noen justeringer med syntaksen som må gjøres.

Ved å bruke XMF-Mosaic sammen med RSM er det noen utfordringer. I og med at det per i dag ikke er mulig å utveksle modeller mellom disse verktøyene, så må modeller oversettes. Det vil si at etter at modellene er laget i RSM, må de modeller som skal transformeres lages på ny i XMF-Mosaic. Dette er både tidkrevende og litt irriterende. For å kunne bruke min metode i sin helhet er det viktig å utvikle et fullstendig verktøy, slik at en slipper dette.

I anvendelsen er det kun forretningsregelmodellen som er blitt transformert. Mye gjenstår på transformasjonsfronten. Forhåpentligvis vil ATHENA foreslå en mappingmodell med tilhørende transformasjonsregler for å transformere prosesser i POP* til prosesser i PIM4SOA. I dokumentasjonen som beskriver PIM4SOA [7] er det sagt at det skal gjøres.

9.4 Sammenligning av REMO og andre metoder

Er REMO bedre enn de andre metodene som er blitt evaluert? Basert på evalueringstabellen så er REMO helt klart den beste. For evalueringen av CIM, får REMO totalt 12 poeng av 14 mulige og for PIM får REMO 14 poeng av 16 mulige. REMO har basert seg på positive elementer fra andre metoder, og dermed er det egentlig forventet at den skal komme best ut poengmessig.

Tabell 9.4 på neste side viser evalueringen av alle metoder som er evaluert for CIM. Foruten om REMO, så kommer ARIS bra ut av evalueringen med 10 av 14 mulig poeng. Kravene om brukervennlighet, informasjonsmodell, organisasjonsmodell og prosessmodell er alle krav som er greit oppfylt av tilnærmet alle metoder som ble evaluert. Kravet om verktøystøtte er derimot et krav som de fleste kun har fått 1 for. Den generelle tendensen er at hver metode har sitt eget verktøy, gjerne et verktøy som koster mye eller som ikke er kompatibelt med andre verktøy. POP* er den eneste som har fått 2 for denne, og det er fordi den baserer seg på UML og bruken av stereotyper. Dette gjør forsovidt REMO også, men den mangler en godt utarbeidet UML-profil som kan anvendes, og i tillegg er det ikke mulig å lage en beslutningsrabell i RSM (verktøyet som hovedsakelig er anvendt) som jo er ønskelig i REMO. Funksjonalitetsbeskrivelse er det andre kravet som mange metoder skårer lavt på. Grunnen til dette er det at metodene som er valgt ut er basert på virksomhetsmodellering. I REMO er det kun gitt en liten veilending på hva en bør tenke på her, og dermed er det mer opp til systemutvikleren hvordan kravmodellen skal se ut. Kravet om forretningsregelmodell er den store mangelen i alle metoder utenom REMO. Det eneste er SBVR som får 1 for dette kravet. SBVR er laget nettopp for å representere forretningsregler, men har som vi har sett noen mangler på det.

REMO har basert seg noe på SBVR, fordi tankegangen bak SBVR er bra. I SBVR skal forretningsvokabularet lages først og deretter forretningsregler. I REMO skal det lages en informasjonsmodell og en organisasjonsmodell, og dermed får en forretningsvokabularet frem gjennom disse modellene. Forretningsreglene i REMO er mer begrenset enn det de er i SBVR. Noen av forretningsreglene som ville vært med i SBVR, er i REMO tatt med i prosessmodellen. Dette er regler som angir generelle prosesser, og derfor kommer bedre frem i en modell enn med ren tekst.

Krav til CIM	**ARIS**	**EEML**	**POP***	**SBVR**	**REMO**
CIM1: Brukervennlighet	2	2	1	1	2
CIM2: Verktøystøtte	1	1	2	1	1
CIM3: Informasjonsmodell	2	2	1	2	2
CIM4: Organisasjonsmodell	2	2	2	1	2
CIM5: Prosessmodell	2	2	2	0	2
CIM6: Funksjonalitetsbeskrivelse	1	0	1	0	1
CIM7: Forretningsregelmodell	0	0	0	1	2
SUM	10	9	9	6	12

Tabell 9.4: Evaluering av CIM

REMO viser seg å være en god metode for PIM også, sammenlignet med de andre som er blitt evaluert. Nå må det nevnes at for utviklingen av REMO ble det gjort noen avgrensninger for PIM, som nevnt i kapittel 2, og av den grunn er ikke alle delene av PIM utviklet like bra helt enda. I tabell 9.5 på neste side er evalueringen av REMO og andre metoder vist. REMO er også her metoden som får flest poeng, totalt 14 av 16 mulige. Det som er gjort i REMO for PIM, er å plukke ut alle de positive elementene fra de andre metodene som ble evaluert, og sette dem sammen slik at en god arkitekturmodell er mulig å lage. COMET gir en god veiledning på hvilke modeller som bør lages for å vise hvordan komponentene i et system henger sammen. Disse er tatt med i REMO. PIM4SOA har gode inndelinger som passer for å ta i bruk virksomhetsmodeller på CIM-nivå, f.eks. prosessdimensjonen, og REMO viderfører derfor disse inndelingene. Det som mangler i COMET og PIM4SOA er modellering av en regelkomponent, men i PRR er denne komponenten dekket. PRR har derfor blitt brukt for å detaljere reglene på PIM-nivå i REMO.

I REMO har fremdeles noen av kravene bare fått 1. Dette er kravet om prosessmodell og kravet om verktøystøtte. Prosessmodellen i REMO ble ikke utprøvd godt nok til å gi den 2. Tanken er å bruke resultatene fra ATHENA med tanke på å transformere prosessmodeller i POP* til prosessmodeller i PIM4SOA. Når det er på plass blir det enkelt å spesifisere prosesskomponenten i REMO, og da blir også virksomhetsmodellen i høyere grad anvendt for spesifiseringen av IT-systemer.

Verktøystøtte er også for PIM noe som ikke er helt på plass. Det er et generelt behov for et komplett verktøy som støtter opp om REMO og også MDA, slik at det blir enkelt å modellere og transformere modeller. Det trenger ikke være et helt nytt verktøy, men i hvert fall en utvidelse til RSM

eller Eclipse, slik at det er mulig å få modellert alt som er ønskelig, og enkelt transformere disse modellene.

Krav for PIM	PRR	PIM4SOA	COMET	REMO
PIM1: Komponentstruktur	0	0	2	2
PIM2: Komponentinteraksjon	0	0	2	2
PIM3: Funksjonsmodell	0	2	1	2
PIM4: Informasjonsmodell	0	2	2	2
PIM5: Produksjonsregelmodell	2	0	0	2
PIM6: Prosessmodell	0	2	0	1
PIM7: Interfacespesifisering	0	0	2	2
PIM8: Verktøystøtte	1	1	2	1
SUM	3	7	11	14

Tabell 9.5: Evaluering av PIM

Selv om REMO er metoden som kommer best ut av evalueringen, betyr ikke det nødvendigvis at REMO alltid skal anvendes. Det er viktig å tenke på hva som er behovet for systemet som skal utvikles, og dersom det behovet er lignende målet til REMO, så er REMO metoden som bør anvendes. For større utviklingsprosjekter, hvor forretningsregler står som et sentralt element, vil REMO være en nyttig metode for å kunne utnytte virksomhetmodellen på et plattforspesifikt nivå.

9.5 REMOs videreutvikling

REMO ser ut til å være en godt egnet metode for å bruke virksomhetsmodeller for å spesifisere IT-systemer og som samtidig skiller forretningsregler ut som en egen komponent, men REMO har noen problemer også. MDA er ikke nødvendigvis den beste måten å angripe problemet på slik som vi har gjort. Dette er på grunn av blant annet mangel på et godt verktøy, og også fordi MDA noen ganger kan gjøre ting litt mer komplisert enn hva som er nødvendig.

REMO baserer seg på MDA, og har så langt vært lojal mot tankegangen i MDA. I REMO lages CIM, deretter gjøres transformasjoner av de elementene i CIM som kan transformeres, så får vi et utgangspunkt for PIM som igjen videreutvikles. PIM skal, i en utvidelse av REMO, deretter transformeres til PSM, og til slutt transformeres til kjørbar programkode. Det kan da diskuteres om PSM vil være en nødvendig modell. Er det mulig å gå direkte fra PIM til programkode? Blir det for mange modeller med CIM, PIM og PSM? Eventuelt, går det an å gå fra CIM til PSM? PIM er utformet på en plattformuavhengig måte, så dersom vi skal gå direkte fra PIM til kode, så blir transformasjonsreglene litt mer kompliserte. Transformasjonsregler er noe som er gjenbrukbart, så dersom vi ønsker å gjøre det på denne måten trenger vi kun å definere reglene en gang og deretter bruke dem om igjen til andre anledninger. Den andre muligheten er å gå direkte fra CIM til PSM, men dette er for meg mer vanskelig fordi jeg er så vant til å modellere generell UML, ikke UML fokusert mot f.eks.

et bestemt programmeringsspråk. Det kan være greit å beholde et klart skille mellom virksomhet, arkitektur og plattformspesifikk arkitektur. Før en kan ta noen avgjørelse på dette bør det utvides med CIM, PIM, PSM og til slutt kode, og analysere om dette blir for komplekst eller om det er hensiktsmessig med alle disse abstraksjonsnivåene.

REMO trenger også en del utvidelser når det gjelder dette med transformasjon av forretningsprosesser i CIM til prosesser i PIM. Det er foreløpig gitt at resultatene fra ATHENA skal anvendes, men disse resultatene må uansett tilpasses resten av metoden. Så før REMO blir komplett må dette gjøres.

Kapittel 10

Konklusjon og videre arbeid

I denne masteroppgaven har vi sett hvordan vi kan bruke virksomhetsmodellering for å spesifisere IT-systemer og hvordan forretningsregler kan skilles ut som et eget element allerede på virksomhetsmodelleringsnivå. I introduksjonen ble tre spørsmål stilt, og i dette kapittelet skal vi konkludere. Dette kapittelet tar også for seg hva som bør gjøres av videre arbeid.

10.1 Konklusjon

Virksomhetsmodellering kan brukes for å få en oversikt over virksomheten, dens organisering, prosesser og regler, men denne oppgaven ønsket å finne ut om virksomhetsmodellering kunne brukes til noe mer enn det. Målet var å undersøke om virksomhetsmodeller kunne brukes til å spesifisere IT-systemer. Følgende spørsmål ble stilt for å undersøke om det var mulig:

> **Spørsmål 1:** *Kan virksomhetsmodeller transformeres til arkitekturmodeller og videre til programkode og dermed sikre at IT-systemet tilfredsstiller forretningsfolkenes behov?*

I oppgaven har jeg evaluert ulike metoder og modelleringsspråk for å lage virksomhetsmodeller, to språk for å gjøre modelltransformasjoner, og i tillegg har jeg sett på noen metoder for å lage arkitekturmodeller. Metoden REMO som er introdusert foreslår en måte å transformere deler av virksomhetsmodellen til arkitekturmodell. På bakgrunn av dette kan vi si at virksomhetsmodeller kan transformeres til arkitekturmodeller. Imidlertid er det ikke alle elementer i en virksomhetsmodell som skal transformeres. Noen av elementene i virksomhetsmodellen skal videreføres slik de er, og noen skal ikke videreføres i det hele tatt. Men uansett så kan vesentlige deler av virksomhetsmodellen oversettes til arkitekturmodellen og dermed bidra til å spesifisere IT-systemer. Virksomhetsmodellens hensikt er å uttrykke forretningsfolkenes behov. Når virksomhetsmodellering gjøres grundig, og resultatene oversettes til arkitekturmodellen, har vi altså grunn til å tro at IT-systemet vil oppfylle det behovet som er fremsatt. Overgangen

til programkode og kjørbart system er diskutert i avsnitt 10.2.2

Det andre problemet som ble nevnt i introduksjonen, var det å kunne håndtere regler på en effektiv måte i IT-systemer. Forretningsregler er regler som endres ofte, og derfor er det behov for å håndtere endringene effektivt. Metoden REMO ble utformet nettopp for å håndtere frorretningsregler, siden vi oppdaget mangelen på slik regelhåndtering i eksisterende løsninger. Det ble i introduksjonen stilt to spørsmål om forretningsregler:

> **Spørsmål 2:** *Kan forretningsregler skilles ut som et eget element i virksomhetsmodellen og transformeres derfra til en arkitekturmodell og videre til programkode?*
>
> **Spørsmål 3:** *Kan separasjon av forretningsregler fra annen programkode forenkle vedlikeholdet av regler?*

Vi har sett hvordan REMO bidrar til å bruke virksomhetsmodellering som grunnlag for spesifiseringen av IT-systemer, og at metoden samtidig fokuserer på forretningsregler, og skiller reglene ut som egne modeller. Ved å identifisere forretningsreglene tidlig, og hele tiden ha fokuset på reglene i virksomheten, er det mulig å skille reglene ut som en egen komponent. Regelmotorer har mulighet for å implementere produksjonsregler, regler som er rettet mot oppførsel. Ved å transformere forretningsregler til produksjonsregler, er det mulig å implementere reglene i en regelmotor. Dette er fordi produksjonsregler kan transformeres til input til regelmotorer. Ved å ha reglene i en regelmotor, får vi forretningsreglene skilt ut som en egen komponent. Etterprøvingen av REMO ble gjort ved et casestudium. Da så vi at reglene kunne skilles ut som en egen modell i virksomhetsmodellen, og reglene ble transformert til produksjonsregler ved hjelp av modelltransformasjoner.

Det er noen problemer med den foreslåtte metoden. For det første så er det vanskelig å identifisere gode forretningsregler. Forretningsregler er ikke alltid eksplisitt gitt i virksomheter, og da krever det mye tid og arbeid å komme frem til alle regler. Det betyr at det ikke alltid vil være riktig å fokusere i like stor grad på forretningsregler som vi har gjort i denne metoden, og at metoden derfor bør anvendes på typisk regelstyrte virksomheter. Et eksempel på en regelstyrt virksomhet er et forsikringsselskap. I forsikringsselskaper er det mange regler som er forbundet med det å beregne forsikringspremier, utstede forsikringer o.l. Her er det viktig å kunne oppdatere reglene på en enkel måte, og dermed vil det være lurt å ha reglene i en regelmotor. I slike tilfeller vil REMO være en god metode for å komme frem til dette. Det er ikke bare for REMO at det er vanskelig å identifisere forretningsreglene og formulere disse, men det er et generelt problem for andre tilnærminger også.

Det andre er at det ikke er god nok støtte i verktøy for å bruke metoden. Dette er en generell mangel ved alle deler av metoden, og man vil få et større utbytte av metoden med et verktøy på plass.

Det tredje er at REMO mangler en veiledning på transformasjon av forretningsprosesser i CIM til prosesser i PIM. Forretningsreglene i en virksomhet står sentralt, men det gjør også forretningsprosessene, og REMO blir en komplett metode først når denne transformasjonen er på plass. Det er forventet at ATHENA skal komme med resultater som REMO kan anvende på dette området.

10.2 Videre arbeid

Denne oppgaven er et godt utgangspunkt for å bruke virksomhetsmodellering for å spesifisere IT-systemer og for å håndtere forretningsregler, men det kreves en del videre arbeid rundt dette. Det er hovedsakelig tre ting det bør jobbes videre med: verktøy for REMO, overgangen til et kjørbart system og en større etterprøving for å anvende REMO i praksis. I tillegg så er det noen mindre elementer i REMO som bør utbedres.

10.2.1 Verktøystøtte for REMO

En suksessfaktor for en metode er at det er verktøystøtte for den. Uten det vil ingen anvende metoden i praksis. For å kunne bruke REMO og andre MDA-metoder trengs det et verktøy som gjør det enkelt å modellere og transformere modellene. En mulig grunn til at MDA ikke er blitt så populært som det var tenkt til, er nok fordi det ikke eksisterer et godt verktøy for det. Det er flere muligheter for utvikling av verktøy. Det er behov for et komplett verktøy, gjerne hvor en kan definere metamodellene som en bruker i utviklingen selv, men som lett bruker disse og kan gjøre transformasjoner mellom dem. Slik det er nå har vi kun verktøy som kan brukes for enkelte deler av REMO.

En mulighet er å bruke Rational Software Modeler (RSM) til å lage modeller for CIM og PIM, og gjøre modelltransformasjoner ved å bruke Atlas Transformation Language (ATL) som plugin til RSM. Imidlertid er det en mangel ved ATL, slik at når en skal transformere modeller må modellene først lages tekstlig, for så å transformeres, og deretter må resultatet fra transformasjonen oversettes til en modell. Det er ekstremt tungvint å sitte og skrive modellene som en lager. Da faller litt av poenget som ligger bak modellbasert systemutvikling bort. En annen mulighet er å generere en editor for ønsket metamodell i XMF-Mosaic. Som vi har sett tilbyr XMF-Mosaic modelltransformasjonsspråk, og i tillegg er det mulighet for å lage en egendefinert editor i dette programmet. Muligheten for å utvikle verktøy for REMO er tilstede i XMF-Mosaic, men dette bør undersøkes nærmere. En tredje mulighet er Graphical Modeling Framework (GMF) [78]. Med dette rammeverket er det mulig å utvikle grafiske editorer basert på Eclipse Modeling Framework (EMF) og Graphical Editing Framework (GEF).

Flere av verktøyene jeg har prøvd ut har hatt støtte for å utveksle UML-modeller ved å bruke XMI, men sjelden har den funksjonaliteten fungert. Det ser ut til at man ikke har klart å bli enige om hvordan XMI skal fungere, og da er noe av poenget med XMI borte. Dersom et slikt utvekslingsformat hadde fungert, ville det vært enkelt å f.eks. ha et verktøy egnet for CIM, eksportert modellene til et egnet transformasjonsverktøy, og videre til et egnet verktøy for PIM, men da kreves det klare avtaler om hvordan utvekslingsformatet skal være.

10.2.2 Overgangen til kjørbart system

Vi har underveis sett på hvordan virksomhetsmodellene kan brukes i arkitekturmodellene. Forretningsregler har blitt transformert til produksjonsregler, men ikke videre til kjørbar kode. Vi har antatt at det å bruke arkitekturmodellen videre, ikke skal være et problem fordi REMO forholder seg til standarder på PIM-nivå. Flere av disse standardene har veiledning for hvordan en kan transformere til PSM, og for de andre er denne veiledningen på vei. REMO gir kun en veiledning på CIM, PIM, og transformasjoner mellom dem. Tanken er å utvide REMO med et PSM-nivå, og transformasjoner fra PIM til PSM, og fra PSM til kjørbar kode.

REMO bruker Production Rule Representation (PRR) på PIM-nivå. I arbeidsgruppen i OMG som har utarbeidet PRR, arbeides det med transformasjonen av produksjonsregler og til kode for regelmotorer, og dermed er det mulig å bruke deres resultater etterhvert som en videreutvikling av REMO. Utvidelsen til programkode blir enkel å gjøre dersom resultatene fra PRR er egnet for REMO.

Det bør også undersøkes om det i det hele tatt er nødvendig med et PSM-nivå. Kanskje er det mulig å gå direkte fra PIM til programkode. Det er ikke gitt at MDA med CIM, PIM og PSM er løsningen. Et annet alternativ kan være å gå fra CIM til PSM. I bedrifter er de ganske klare på hvilke plattformer de ønsker, og når da IT-systemer utvikles, så ønsker de ikke nødvendigvis å kunne bytte plattform. For firma som utvikler bestemte systemer, kan det være lurt med PIM-nivået fordi da kan de lettere tilby samme produkt for flere plattformer. Dermed kan de lettere selge systemet til ulike kunder som anvender ulike tekniske plattformer.

10.2.3 REMO i praksis

På grunn av tidsrammen for denne masteroppgaven er REMO etterprøvd på et relativt lite eksempel. Dette eksempelet ble valgt ut fordi det er brukt som felles eksempel i prosjektet INTEROP, og dermed hadde jeg mulighet for å diskutere disse modellene med deltakere i dette prosjektet.

I og med at dette eksempelet var såpass lite, så vi blant annet at det var få regler som ble identifisert, og det var ikke mye funksjonalitet som måtte modelleres. Større eksempler kan gi et bedre grunnlag for evalueringer og

diskusjoner. Det er derfor behov for å prøve ut REMO i en større skala. Det vil være en fordel å prøve ut REMO på caser som omhandler mer regelstyrte virksomheter, som f.eks. et forsikringsselskap.

10.2.4 Forbedring av REMO

Slik REMO per dags dato er utviklet er det noen småelementer som trenger utbedring. Det første som bør gjøres, er å lage en komplett metamodell for REMO. Akkurat nå gjør REMO bruk av ulike metamodeller for de ulike delene, men dette bør samles. Da kan vi også lage en UML-profil for REMO, og da er vi et steg nærmere en mer helhetlig verktøystøtte for metoden.

Det andre er å jobbe mer med koblingen mellom reglene og informasjonsmodellene, i hvert fall på PIM-nivå. Det vil si at vi må utvide det hvite feltet i avgrensningsfiguren (figur 2.4 på side 9). Det må opprettes en kobling mellom produksjonsregler og informasjonsmodellen. En regel bruker flere objekter som er modellert i informasjonsmodellen, og dette må vises på en eller annen måte. Da vil reglene bli bedre tilpasset til hva som finnes i virksomheten. Slik REMO er nå, går koblingen mellom informasjon og regler i PIM gjennom eventuelle funksjonskomponenter som lages.

Det tredje er å jobbe med transformasjonen av forretningsprosesser til prosesser i PIM. Det er antatt at ATHENA vil komme med resultater på dette området som man kan bruke, men det er ikke gitt at disse resultatene vil være egnet. Dersom det ikke kommer noe egnet fra ATHENA, er det verdt å arbeide videre med regler for transformasjon av forretningsprosesser i CIM til prosesser i PIM.

Bibliografi

[1] Adaptive, Business Rule Solurions LLC, Business Semantics Ltd, Hendryx, Associates, MEGA, and Neumont University. Semantics of Business Vocabulary and Business Rules (SBVR), 2005. http://www.omg.org/cgi-bin/doc?/05-08-01.pdf, accessed 30.08.05.

[2] Naci Akkøk. *Towards the Principles of Designing Diagrammatic Modeling Languages: Some Visual, Cognitive and Foundational Aspects*. PhD thesis, University of Oslo, 2004.

[3] Kristrun Arnarsdottir. Semantic mapping: ontology-based vs. model-based. Master's thesis, University of Oslo, 2005.

[4] ATHENA. Model and Report: First Set of Requirements. Technical report, ATHENA, February 2005.

[5] ATHENA. Report on Methodolgy description and guidelines definition. Deliverable DA1.3.1. Prosjektdokument fra ATHENA, March 2005.

[6] ATHENA. Report on Methodology description and guidelines definition. Technical report, ATHENA, March 2005.

[7] ATHENA. Model-driven and Adaptable Interoperability Infrastructure. Deliverable A6.4. Prosjektdokument fra ATHENA. WP A6.2/3/4/5/6/7, January 2006.

[8] ATHENA. ATHENA overview. http://www.athena-ip.org/index.php?option=com_docman&Itemid=46&task=vie%w_category&catid=36&order=dmdate_published&ascdesc=DESC, accessed 10.01.06.

[9] Don Baisley. OMG and Business Rules, 2005.

[10] Arne-Jørgen Berre, Brian Elvesæter, Jan Øyvind Aagedal, Jon Oldevik, Arnor Solberg, and Bjørn Nordmoen. *COMET Methodology Handbook*. SINTEF ICT, 2.4 edition, April 2004.

[11] Arne J. Berre, Manfred Jeusfeld, Michael C. Jaeger, Kurt Geihs, Jean-Perre Bourey, Michel Bigand, and Ecole de Lille. *State of the Art of Software Engineering - with focus on Enterprise/Business Model to System Model mappings*, 2004. http://interop-noe.org/workspaces/wp7/copy_of_documents/progress/task72%/, accessed 17.10.05.

[12] Arne J. Berre. *SD7.3 Part 2 - Model transformation - Illustrative Example*, 2004.

[13] Arne Berre and Guy Doumeingts. Work Package Description. http://interop-noe.org/workspaces/tg2/documents/dow/dow, accessed 10.11.05.

[14] Boldsoft, Rational Software Corporation, IONA, and Adaptive Ltd. Response to the UML2.0 OCL RfP. Technical report, OMG, January 2003. http://www.omg.org/cgi-bin/doc?ad/03-01-07.pdf, accessed 06-03-2006.

[15] Anis Charfi and Mira Mezini. Hybrid web service composition: business processes meet business rules. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 30–38, 2004.

[16] Angelo Corallo. Semantics of Business Vocabulary and Rules Approach To Business Modelling Language for the Digital Business Ecosystem Project. OMG website, August 2005. http://www.omg.org/docs/omg/05-04-10.pdf accessed 25.08.05.

[17] Sofie de Flon Arnesen. Evaluering av språk for arbeidsprosessmodellering. Master's thesis, NTNU, institutt for datateknikk, 2001.

[18] Ana Belèn Garcia Diez. ATHENA - First Version of State of the Art in Enterprise Modelling Techniques and Technologies to Support Enterprise Interoperability. http://www.athena-ip.org/index.php?option=com_docman&Itemid=46&task=vie%w_category&catid=56&order=dmdate_published&ascdesc=DESC, accessed 14.06.05.

[19] Guy Doumeingts, Arne J. Berre, Jean-Pierre Bourey, and Reyes Grangel. Report on Model Establishment. Technical report, INTEROP, 2005. http://interop-noe.org/workspaces/tg2/documents/working/DTG2.1/TG2_1_de%liverable_v1_5.doc.

[20] Guy Doumeingts, Arne Berre, Reyes Grangel, Manfred Jeusfeld, Kurt Geihs, Yves Ducq, Jean-Pierre Bourey, and Michel Bigand. *State of the art of methods to derive IT såecifications from models and to develop IT specific components based on IT modelling*, January 2005. http://interop-noe.org/workspaces/wp7/copy_of_documents/deliverables/D71final, accessed 17.10.05.

[21] Ernest Friedman-Hill. *JESS in Action, Rule-Based Systems in Java*. Manning, 2003.

[22] ATLAS group. Atlas Transformation Language User Manual v0.7. Technical report, ATLAS group LINA and INRIA, February 2006. http://www.eclipse.org/gmt/atl/doc/, accessed 06-03-2006.

[23] Business Rules Group. Business Rules Manifesto. BRGs website. http://www.businessrulesgroup.org/brmanifesto.htm, accessed 30.08.05.

[24] Object Management Group. *Request for Proposal: MOF 2.0 Query / View / Transformations RFP*, October 2002. http://www.omg.org/docs/ad/02-04-10.pdf, accessed 06-03-2006.

[25] Object Management Group. Business Semantics of Business Rules, Request for Proposal. http://www.businessrulesgroup.org/second_paper/BRG-BMM.pdf, June 2003.

[26] Curtis Hall and Paul Harmon. The 2005 Enterprise Architecture, Process Modeling and Simulation Tools Report. Technical report, Business Process Trends, 2005. http://www.bptrends.com/members/deliver.cfm?target=BPTrendsModelingReport1-1CompleteNov05.pdf, accessed 31.08.05.

[27] John Hall, Keri Anderson Healy, and Ronald G. Ross. The Business Motivation Model. Technical report, The Business Rules Group, 2005. http://www.businessrulesgroup.org/second_paper/BRG-BMM.pdf, accessed 05.10.05.

[28] Paul Harmon. The OMG Continues its Work on Business Process Standards. Technical report, OMG, 2004. http://www.omg.org/attachments/pdf/PaulHarmonBParticle.pdf, accessed 25.08.05.

[29] David Hay and Keri Anderson Healy. Defining Business Rules - What Are They Really? Technical report, The Business Rules Group, 2000. http://www.omg.org/docs/ad/02-10-03.pdf accessed 25.08.05.

[30] Stan Hendryx. BP standards at the OMG's Montreal Meeting. Technical report, Business Process Trends, 2004. http://www.bptrends.com/publicationfiles/10-04COLOMGMontreal-Hendryx.pdf, accessed 25.08.05.

[31] ILOG. Enterprise Business Rule Management with JRules. ILOG website, February 2005. http://www.ilog.com/products/jrules/whitepapers/index.cfm?filename=JRul%es50TechWhitePaper.pdf, accessed 26.01.06.

[32] ILOG. ILOG JRules: Leading the Way in Business Rule Management Systems. ILOG website, March 2005. http://www.ilog.com/products/businessrules/whitepapers/index.cfm?filena%me=WP-JRules50Strengths.pdf, accessed 26.01.06.

[33] ILOG. ILOG website for Business Rules. http://www.ilog.com/products/businessrules/.

[34] ILOG. Initial Response to Product Rule Representation RFP. http://www.omg.org/techprocess/meetings/schedule/Prod._Rule_Representat%ion_RFP.html, accessed 26.01.06.

[35] Interop. Interop - Presentation of the Project. http://interop-noe.org/INTEROP/presentation, accessed 10.11.05.

[36] Peter Krendev. Blue Print - Demo Telco. INTEROP document, see www.interop-noe.org, 2003.

[37] John Krogstie. Overview of eeml and process knowledge layers.

[38] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. *Object Oriented Analysis and Design*. Marko Publishing ApS, 1. edition, 2000.

[39] James McGovern, Scott W. Ambler, Michael E. Stevens, James Linn, Vikas Sharan, and Elias K. Jo. *A Practical Guide to Enterprise Architecture*. Prentice Hall, andre edition, 2004.

[40] Stephen J. Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison Wesley Professional, March 2004. From Safari Tech Books Online.

[41] Jan Mendling and Markus Nüttgens. EPC Markup Language. http://wi.wu-wien.ac.at/Wer_sind_wir/mendling/publications/TR05-EPML.pd%f.

[42] Jan Mendling and Markus Nüttgens. Transformation of ARIS Markup Language to EPML. http://wi.wu-wien.ac.at/Wer\{_}sind\{_}wir/mendling/publications/04-EPK%-AML.pdf.

[43] Joaquin Miller and Jishnu Mukerji. MDA Guide Version 1.0.1. Technical report, OMG, June 2003. http://www.omg.org/docs/omg/03-06-01.pdf, accessed 15.11.05.

[44] Tony Morgan. *Business Rules and Information Systems: Aligning IT with Business Goals*. Addison-Wesley, 2002.

[45] Conrad Bock (NIST), Tracy Gardner (IBM), and Jim Amsden (IBM). BPDM Context and Status. Technical report, OMG BEI, 2005. http://www.omg.org/docs/bei/05-06-04.pdf accessed 25.08.05.

[46] Anna Gunhild Nysetvold. Prosessorientert IT-arkitektur. Master's thesis, NTNU, institutt for datateknikk, 2004.

[47] Jon Oldevik and Arne J. Berre. *SD7.3 Part 1 - Model transformation. State of the art.*, 2004.

[48] OMG. *Business Process Definition Meta-model - Request For Proposal*, 2003. http://www.omg.org/docs/bei/03-01-06.pdf, accessed 10.04.2006.

[49] OMG. Production Rule Representation RFP. http://www.omg.org/docs/br/03-09-03.pdf, September 2003.

[50] OMG. Business Process Definition Meta-model - Revised submission. http://www.bpmn.org/Documents/BPDM/OMG-BPD-2004-01-12-Revision.pdf, accessed 10.04.2006, 2004.

[51] OMG. Business Rule Management Request for Information. http://www.omg.org/docs/bei/04-06-03.pdf, October 2004.

[52] OMG. Production Rule Representation Proposal Draft 1. http://www.omg.org/techprocess/meetings/schedule/Prod._Rule_Representat%ion_RFP.html, accessed 26.01.06, August 2004.

[53] OMG. *Semantics of Business Vocabulary and Business Rules (SBVR)*, November 2005. http://www.omg.org/techprocess/meetings/schedule/SBVR_FTF.html, accessed 26.01.06.

[54] OMG. Productin Rule Representation Proposal. Draft Response to OMG RFP br/2003-09-03, January 2006.

[55] OMG. Production Rule Representation Proposal. http://www.omg.org/techprocess/meetings/schedule/Prod._Rule_Representat%ion_RFP.html, accessed 26.01.06, January 2006.

[56] OMG. EAML: A MOF-Based, Common Enterprise Architecture Metamodel. http://www.omg.org/news/whitepapers/isp_ea_paper.pdf.

[57] OMG. OMG PRR Update. http://www.omg.org/docs/bmi/06-02-08.pdf.

[58] OMG. *UML TM Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*. http://www.omg.org/docs/ptc/04-09-01.pdf, accessed 10.04.2006.

[59] Merriam-Webster OnLine. *WWWebster Dictionary*. Merriam-Webster Incorporated, http://www.m-w.com/, 2000.

[60] Thilo Ried. Event-driven process chains for better flows. http://www.tekom.de/weiter.jsp?id=285, accessed 15.06.05.

[61] Ronald G. Ross. *Business Rule Concepts*. Business Rule Solutions, LLC, second edition, 1998.

[62] August-Wilhelm Scheer. *ARIS - Business Process Modeling*. Springer, tredje edition, 1999.

[63] Ian Sommerville. *Software Engineering*. Addison Wesley, sixth edition, 2001.

[64] Business Rules Team. Semantics of Business Voacabulary and Business Rules. OMG website, June 2005. http://www.omg.org/cgi-bin/doc?bei/05-06-02.pdf accessed 30.08.05.

[65] Unisys. Agile Business Suite - Getting Started with Developer. Dokumentasjon til Unisys Rules Modeler.

[66] Unisys. Unisys Business Rules Methodology and Concepts. Documentation for the unisys Rules Modeler.

[67] Jan Vanthienen. Decision table experiences in business rules and business processes: Building agility. From business rules forum 2005.

[68] Jan Vanthienen. Ruling the Business: About Business Rules and Decision Tables. http://www.econ.kuleuven.be/tew/academic/

infosys/members/VTHIENEN/downl%oad/Papers/br_dt.pdf, accessed 21.03.2006.

[69] Francois B. Vernadat. *Enterpris Modeling and Integration*. Chapman and Hall, 1996.

[70] Barbara von Halle. *Business Rules Applied*. Wiley, 2001.

[71] Gerd Wagner. Rule Modeling and Markup. In Norbert Eisinger and Jan Maluszynski, editors, *Reasoning Web*, volume 3564 of *Lecture Notes in Computer Science*, pages 251–274. Springer, 2005.

[72] Stephen A. White. Using BPMN to Model a BPEL Process. Technical report, Business Process Trends, March 2005. http://www.bptrends.com/publicationfiles/03-05WPMappingBPMNtoBPEL-White.pdf, accessed 31.08.05.

[73] Wikipedia. Wikipedia - rule engine. http://en.wikipedia.org/wiki/Rule_engine, accessed 26.01.06.

[74] MAPPER - Model-based Adaptive Product and Process Engineering, April 2005.

[75] ATHENA hjemmeside. www.athena-ip.org.

[76] BEI hjemmeside. www.omg.org/bei.

[77] Business Rules Group hjemmeside. www.businessrulesgroup.org.

[78] Eclipse GMF hjemmeside. http://www.eclipse.org/gmf/.

[79] INRIA hjemmeside. www.inria.fr.

[80] INTEROP hjemmeside. www.interop-noe.org.

[81] LINA hjemmeside. http://lina.atlanstic.net/en/index.html.

[82] MDA hjemmeside. www.omg.org/mda.

Tillegg A

Liste over akronymer

ADT ATL Development Tools

AKM Active Knowledge Model

ARIS Architecture of Integrated Information Systems

ATHENA Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications

ATL Atlas Transformation Language

BEI Business Enterprise Integration

BPDM Business Process Definition Metamodel

BPEL Business Process Execution Language

BPMI Business Process Management Initiative

BPML Business Process Modeling Language

BRG Business Rule Group

BRMS Busniness Rule Management System

BRT Business Rules Team

CIM Computationally Independent Model

CIMOSA CIM Open Systems Architecture

COMET Component and Model-based Development Methodology

CRM Customer Relationship Management

DoDAF Department of Defense Architecture Framework

EAI Enterprise Application Integration

ebXML Electronic Business using eXtensible Markup Language

EDOC Enterprise Distributed Object Computing

EEML Extended Enterprise Modeling Language

EJB Enterprise JavaBeans

EMF Eclipse Modeling Framework

EPC Event-driven Process Chains

ER Entity Relationship

ERP Enterprise Resource Planning

ESA Enterprise Software Applications

GEF Graphical Editing Framework

GERAM Generalised Enterprise Reference Architecture and Methodology

GIM GRAI Integrated Modelling

GMF Graphical Modeling Framework

GRAI Graphs with Results and Activities Interrelated

IDE Integrated Development Environment

IDEF Integrated DEFinition methodology

IEM Integrated Enterprise Modelling

INTEROP Interoperability Research for Networked Enterprises Applications and Software

ISO International Standards Organization

MDA Model Driven Architecture

MDI Model Driven Interoperability

MEML Monesa Enterprise Modelling Language

Metis BPM Metis Business Process Modeling

Metis ITM Metis Information Technology Management

Metis UML Metis Unified Modeling Language

MOF Meta Object Facility

MOMO Models Morphism

OASIS Organization for the Advancement of Structured Information Standards

OCL Object Constraint Language

OMG Object Management Group

PIF Process Interchange Format

PIM Platform Independent Model

PIM4SOA Platform Independent Model for Service Oriented Architecture

POP* Process Organisation Product *

PRR Production Rule Representation

PSM Platform Specific Model

QoS Quality of Service

QVT Query View Transformation

REMO Regel- og modelldrevet metode

RFP Request For Proposal

RMM Rule Modeling and Markup

RSA Rational Software Architect

RSM Rational Software Modeler

RUP Rational Unified Process

SBVR Semantics of Business Vocabulary and Rules

SOA Service Oriented Architecture

TEAF Treasury Enterprise Architecture Framework

TG Task Group

TOGAF The Open Group Architecture Framework

UEML Unified Enterprise Modeling Language

UML Unified Modeling Language

URM Unisys Rules Modeler

W3C World Wide Web Consortium

WARM Work Analysis Refinement Model

WP Work Packages

XMI XML Metadata Interchange

XML eXtensible Markup Language

XOCL eXtensible Object Command Language

XPDL XML Process Definition Language

Tillegg B

Virksomhetsmodellering

B.1 Rammeverk for virksomhetsmodellering

Det finnes mange ulike rammeverk for virksomhetsmodellering. Et modelleringsrammeverk er "en samling av modelleringsprinsipper, metoder og verktøy som er relevante for et gitt domene eller applikasjon" (fritt gjengitt etter Vernadat [69])

CIMOSA Målet til CIM Open Systems Architecture (CIMOSA) er å hjelpe bedrifter i å håndtere forandringer og å integrere deres fasiliteter og operasjoner for å møte verdensomspennende konkurranse og for å konkurrere på pris, kvalitet og leveringstid [69, side 40]. For å oppnå dette ligger hovedvekten i rammeverket på å lage en integrert virksomhetsmodell. Rammeverket består av tre hovedkomponenter: et virksomhetsmodelleringsrammeverk, en integrert infrastruktur og en CIM system livssyklus.

GERAM ble laget av "Task Force" rundt 1990. "Task Force" var en gruppe som skulle studere feltet som omhandlet virksomhetsreferansearkitektur, og finne den beste metoden for dette. Dette endte med at det ble laget et nytt rammeverk, Generalised Enterprise Reference Architecture and Methodology (GERAM), som ble bygget på bakgrunn av resultater fra CIMOSA, GRAI Integrated Modelling (GIM) og PERA [69]. Det er mange likheter mellom GERAM og CIMOSA.

ARIS er et rammeverk som baserer seg på view-konseptet. Målet for dette rammeverket er å redusere kompleksiteten ved å dele virksomheten inn i individuelle betraktninger. Et viktig element her er ARIS-huset. ARIS er et av rammeverkene jeg har evaluert.

Zachman-rammeverket er en logisk struktur for å klassifisere og organisere de beskrivende representasjonene av en virksomhet som er av betydning for håndteringen av virksomheten og også for utviklingen av virksomheten. Dette rammeverket blir beskrevet som et enkelt rammeverk som er lett å anvende [69].

Foruten om de overnevnte rammeverkene, har vi også GRAI, Department of Defense Architecture Framework (DoDAF), The Open Group Architecture Framework (TOGAF), Treasury Enterprise Architecture Framework (TEAF), Active Knowledge Model (AKM), ISO 15745 (rammeverk for applikasjonsintegrasjon) og MISSION approach [69].

B.2 Virksomhetsmodelleringsspråk

Virksomhetsmodellering kan også defineres som "the art of externalising enterprise knowledge, i.e. representing the enterprise in terms of its organisation and operations"[18].

Det er blitt utviklet utallige grafiske språk for virksomhetsmodellering. Flere av språkene har store likheter, og flere er bare en videreutvikling av et annet språk, eller en tilpasning av et språk til en metodologi. Språk som er kjent i dag, er Integrated Enterprise Modelling (IEM), Metis Information Technology Management (Metis ITM), Metis Business Process Modeling (Metis BPM), EEML, Metis Unified Modeling Language (Metis UML), Monesa Enterprise Modelling Language (MEML), Petri Nets, CIMOSA, GRAI, Integrated DEFinition methodology (IDEF), XML Process Definition Language (XPDL), Business Process Modeling Language (BPML), Enterprise Distributed Object Computing (EDOC), UML profile for Enterprise Application Integration (EAI), Electronic Business using eXtensible Markup Language (ebXML), EPC, Process Interchange Format (PIF), Unified Enterprise Modeling Language (UEML) og Business Process Definition Metamodel (BPDM), for å nevne noen [18, 69].

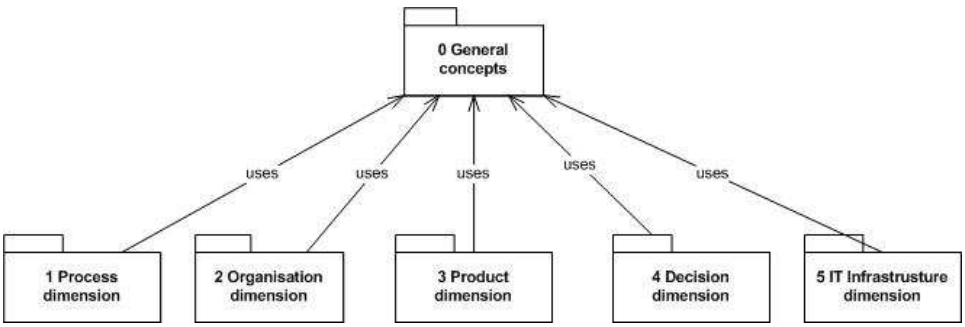
Tillegg C

Metamodeller

Dette tillegget gir en beskrivelse av metamodellene for POP* og PIM4SOA.

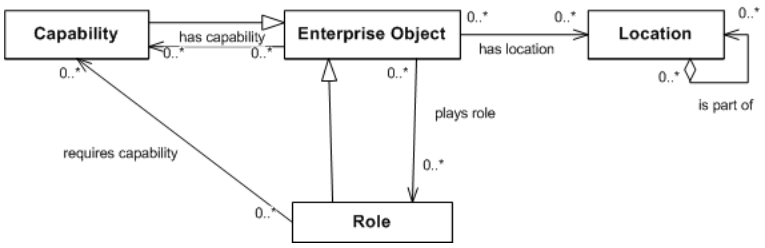
C.1 POP*

POP* har fem dimensjoner (prosess, organisasjon, produkt, beslutning og IT-infrastruktur). For hver dimensjon er det laget en metamodell. I tillegg er det laget en metamodell for generelle konsepter, som alle dimensjonene også bruker. I de generelle konseptene finner vi *Capability*, *Enterprise Object*, *Location* og *Role*. *Enterprise object* kan presentere hva som helst eller hvem som helst som utfører arbeid, gjør at noe skjer, eller er bare en del av en eller annen aktivitet i virksomheten. Personer, roller, håndfaste ting og vage ting (som informasjon og kunnskap) er inkludert i konseptet *Enterprise Object* [5]. Et objekt i virksomheten tar del i virksomheten gjennom å spille en rolle (konseptet *Role*). Et *Enterprise Object* kan også ha *Capability* og *Location* tilknyttet seg. Figur C.2 på neste side viser denne metamodellen.



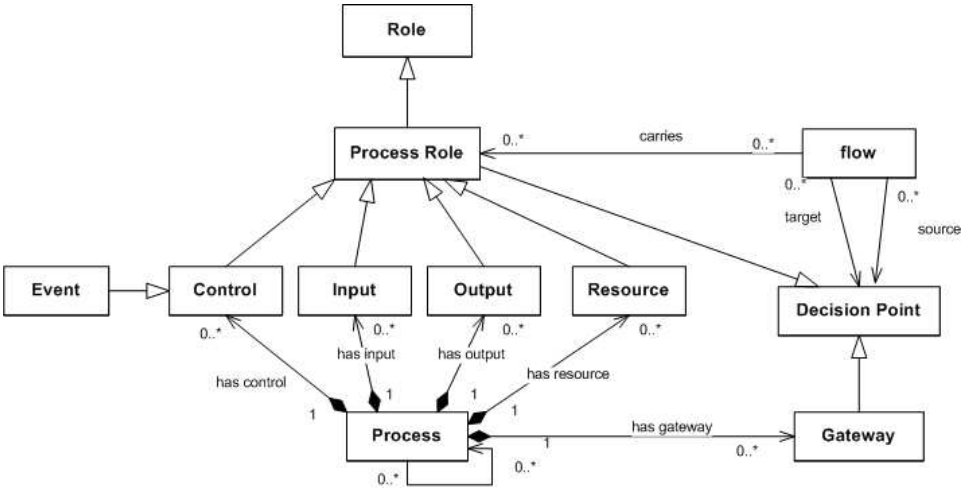
Figur C.1: Oversikt over POP*-metamodellen [5]

Prosessdimensjonen gir konseptene *Process Role*, *Control*, *Input*, *Output*, *Resource*, *Event*, *Process*, *Gateway*, *Decision Point* og *Flow*. Dette er elementer som kan relateres til aktiviteter, oppgaver og prosesser som pågår i virksomheten. Når ulike virksomhetsobjekter deltar i en prosess, beskrives dette med ulike typer prosessroller. Selve logikken i prosessen uttrykkes med



Figur C.2: De generelle konseptene i POP* [5]

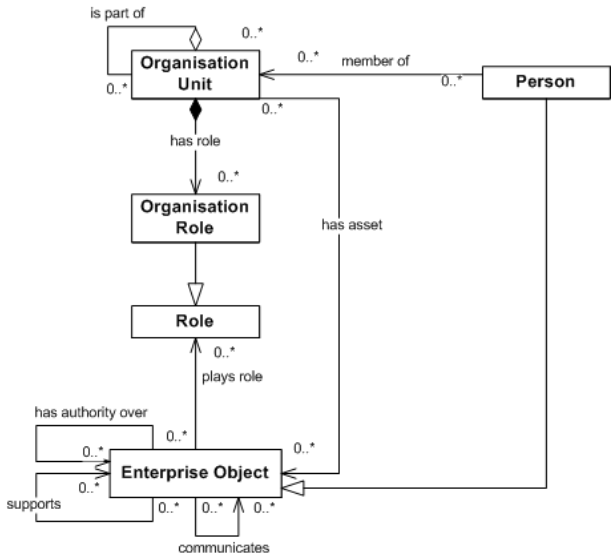
flyt og beslutningspunkt. Prosessdimensjonen er veldig viktig i beskrivelsen av virksomheten fordi den gir en forståelse av hvordan virksomheten fungerer: f.eks. hva som skjer når en gitt hendelse inntreffer, eller hva slags beslutninger som tas i bestemte situasjoner. I denne dimensjonen kan vi dermed danne oss et helhetlig bilde av virksomheten. Prosessen kan modelleres ved å bruke UML 2.0 aktivitetsdiagram med stereotyper definert fra denne metamodellen. Figur C.3 viser prosessdimensjonen.



Figur C.3: Prosessdimensjonen i POP* [5]

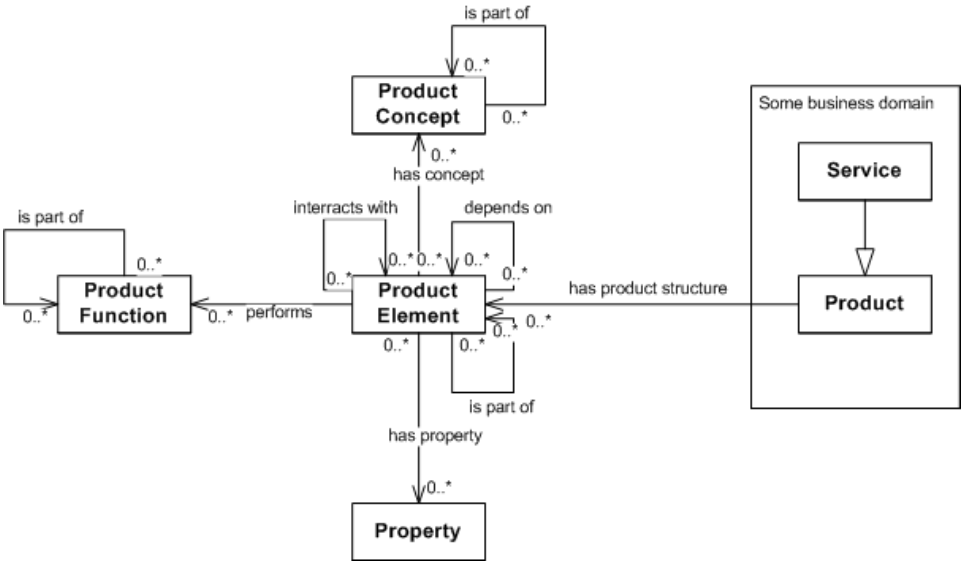
Figur C.4 på neste side viser organisasjonsdimensjonen. Her er det meningen å beskrive organisasjonsstrukturen i virksomheten. Her kan både struktur, medlemmer i organisasjonen og hvilke posisjoner medlemmene har vises. Konseptene som defineres her er *Organisation Unit*, *Person* og *Organisation Role*. Denne dimensjonen lages med UML 2.0 klassediagram med tilhørende stereotyper.

Produktdimensjonen brukes for å modellere produktarkitekturer og produktstrukturer for å kunne vise design, utvikling, datahåndtering og andre elementer. Produktet, eller tjenesten i en forretningsdimensjon, kan relateres til den arkitekturrettede beskrivelsen av produktet eller tjenesten i produktdimensjonen. Konseptene i produktdimensjonen er *Product Concept*, *Product Element* og *Product Function*. I metamodellen (se figur C.5) vises det også hvordan forretningsdomenet relateres til produktdimensjonen. Den-



Figur C.4: Organisasjonsdimensjonen i POP* [5]

ne dimensjonen er ikke så bra til å beskrive generell informasjon i virksomheten, men brukes mer spesifikt mot konkrete produkter og tjenester som virksomheten tilbyr.

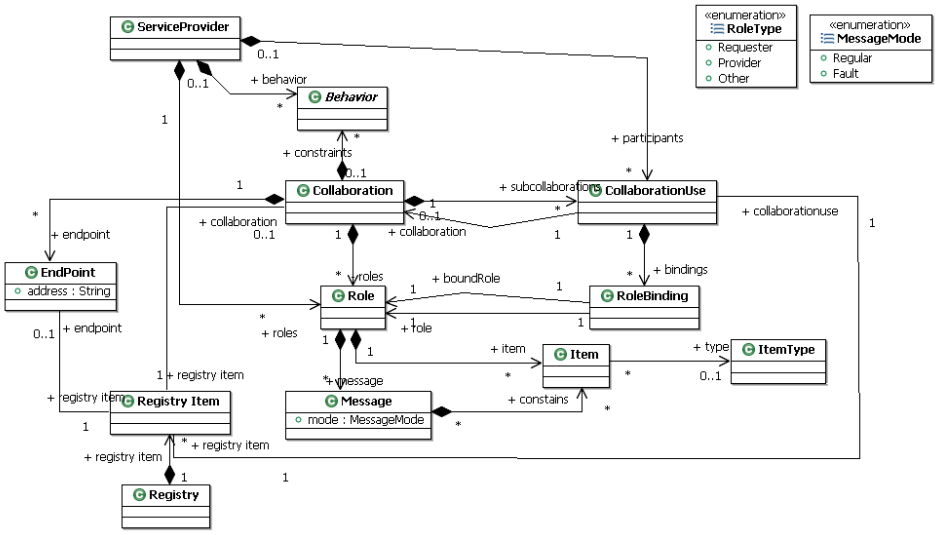


Figur C.5: Produktdimensjonen i POP* [5]

De to siste dimensjonene er beslutningsdimensjonen og dimensjonen for IT-infrastruktur. Disse er ikke ansett som relevante og jeg refererer til beskrivelsen av metamodellen i [5].

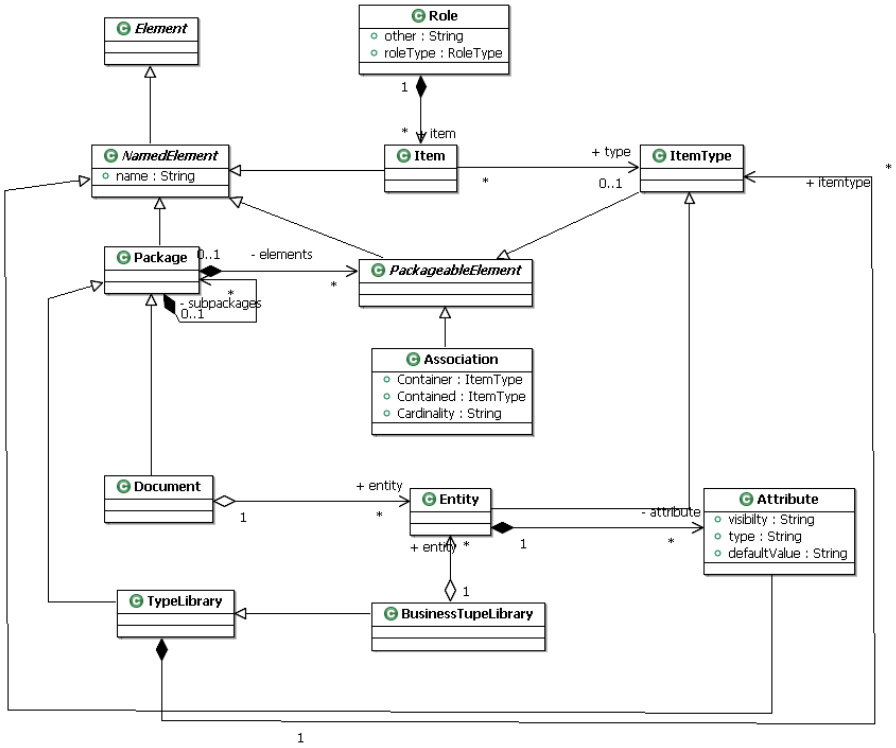
C.2 PIM4SOA

I den tjenesteorienterte metamodellen, som er vist i figur C.6, er det flere sentrale konsepter. En *collaboration* inneholder et sett av roller og et sett av *CollaborationUses*. En *collaboration* er relatert med et register hvor endepunktene (*EndPoint*) er spesifisert. *CollaborationUse* er det modelleringselementet som representerer bruken av en tjeneste. En rolle representerer en part som er involvert i tjenesten. Rollen relateres til tjenesten med *RoleBinding*, og hver rolle har en type som er *requester* eller *provider*. I metamodellen har vi et element som heter oppførsel (*behavior*) og er en abstrakt klasse som brukes for spesifiseringen av meldingssekvensen innen en tjeneste. For å representere en adresse som identifiserer en tjeneste har vi endepunkt. Et av de sentrale elementene i en tjeneste og tjenestemodell er de meldingene (*message*) som sendes til og fra tjenesten. En meldingsmodell definerer en mengde med informasjon som sendes fra en rolle til en annen rolle i en *collaboration* [7].



Figur C.6: PIM4SOA Tjenesteorientert metamodell [7]

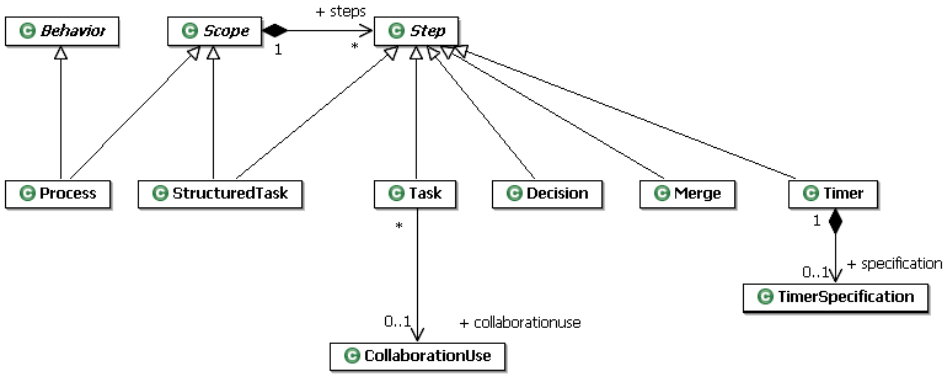
Den informasjonsorienterte metamodellen (figur C.7 på neste side) gir en veiledning på hva som er nødvendig for å modellere informasjonen på et plattformuavhegig nivå. Informasjonsmetamodellen bruker stort sett vanlige UML-elementer som vi kjenner igjen fra klassediagrammer. Elementene i den informasjonsorienterte metamodellen som er hentet fra UML 2.0 er *Element*, *NamedElement*, *PackageableElement* og *Package*. Videre ser vi i figur C.7 at en artikkel (*item*) definerer settet av elementer som en rolle håndterer, og at hver artikkel har en type assoiert til den. *ItemType* kan være string, int og boolean. Neste element i denne metamodellen som jeg vil fremheve er dokument (*document*). Dokumentet representerer et objekt med en bestemt struktur og består av entiteter. En entitet (*Entity*) representerer et strukturert element med informasjon.



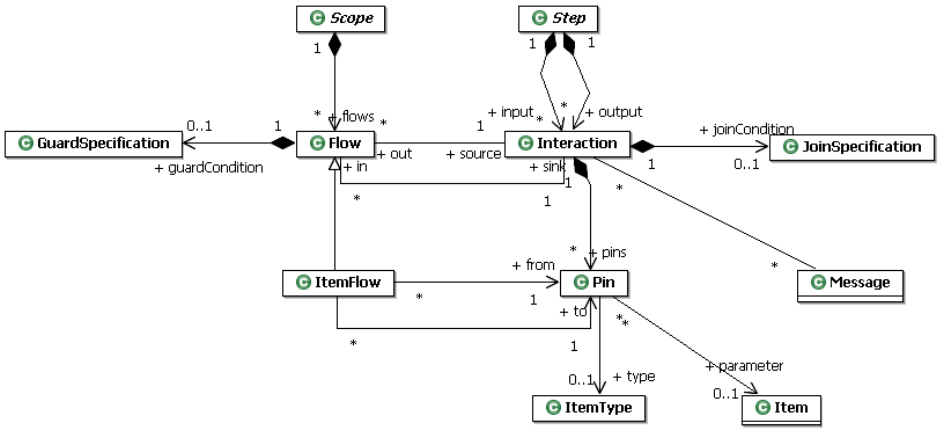
Figur C.7: PIM4SOA Informasjonsorientert metamodell [7]

Den prosessorienterte metamodellen i PIM4SOA er basert på pågående arbeid [50] for BPDM RFP [48] fra OMG, men med noen modifikasjoner som tillater integrering med forretningsprosessmodeller og eksisterende pakker i PIM4SOA. Prosessmetamodellen er nært lenket med tjenestene, og den viktigste lenken mellom prosess og tjeneste er den abstrakte klassen *Scope* (se figur C.9 på neste side). En *Scope* kan instansieres som en prosess, og den kan lenkes til en tjenestetilbyder fra dette aspektet. En prosess inneholder en mengde steg (som oftest oppgaver), og den representerer hendelser som utføres gjennom prosessen. En prosess består av *StructuredTask*, *Steps* og *Interactions/Flows* som lenker oppgavene sammen. Prosessen inneholder et sett av flows mellom dets hendelser, som kan spesialiseres til å vise overføringen av bestemte data [7]. Dermed kan en begynne med å bare vise kontrollflyten, og deretter utvide prosessen med å vise informasjonen som er en del av prosessen.

Metamodellen som er rettet mot ikke-funksjonelle krav er neste element i PIM4SOA. Denne metamodellen baserer seg på en OMG-standard som heter UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [58]. For forklaring av den metamodellen henvises det til [7].



Figur C.8: PIM4SOA Prosessorientert metamodell – prosesselementer [7]



Figur C.9: PIM4SOA Prosessorientert metamodell – prosessaspekt [7]

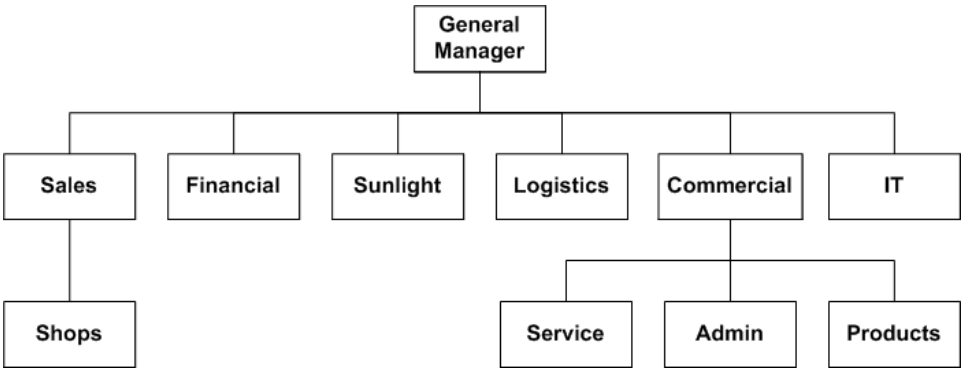
Tillegg D

DemoTelco

DemoTelco-casen har jeg fått fra Singular gjennom Interop-prosjektet. Singular er et gresk firma som leverer ERP systemer, og som også er med i TG2 i Interop. Singular har implementert et ERP system for DemoTelco, og det har hjulpet meg med arbeidet rundt casestudiet.

DemoTelco S.A. er en del av den greske gruppen Telco. Telco er spesialiserte innen telekommunikasjon, i produksjon og distribusjon av batterier og salg av hverdagslige teknologiprodukter.

Per i dag har Telco i bulgaria rundt 50 butikker rundt omkring i landet. I tillegg distribuerer Telco mobilprodukter og tjester gjennom et nettverk på over 150 forhandlere. DemoTelco består både av vanlige butikker som er en del av kjeden, samt franchiser. Stort sett gjelder de samme forretningsreglene for franchisene som det gjør for de vanlige butikkene. Demo Telco har 6 hovedavdelinger: Commercial, Sales, Financial, Logistics, Sunlight og IT. I figur D.1 ser vi organisasjonskartet til DemoTelco.



Figur D.1: Demo Telco organisasjonskart

Vi vil nå se nærmere på hvilke prosesser som vi finner i DemoTelco og på hver avdeling og oppgavene som blir løst der. Det er mange detaljer rundt avdelingene, men vi skal her fokusere på de største og viktigste prosessene i bedriftene. Se ellers [36] for Singulars beskrivelse av Demo Telco.

D.1 Forretningsprosesser i Demo Telco

I Demo Telco finner vi forretningsprosesser for håndtering av innkjøp, ordre, salg, varehus, finans og faste ressurser. Det er 6 hovedavdelinger som stort sett har ansvaret for sine prosesser, men vi finner også enkelte prosesser som går på tvers av avdelingene. Ordrehåndteringsprosessen er en sentral prosess, og er også den prosessen som har blitt brukt som grunnlag i oppgaven. Denne prosessen vil være den mest sentrale i Demo Telco, i og med at det kreves spesiell håndtering av det blant annet pga manglende fysisk varelager. Det er hovedsaklig logistikk- og salgsavdelingen som er av betydning for prosessen håndtere ordre. Vi vil derfor se nærmere på disse, men vi skal også se på de andre avdelingene for å få et helhetlig inntrykk av virksomheten. Salgsavdelingen vil deretter utdypes mer i detalj, da det er her vi finner mest informasjon for ordrehåndteringsprosessen.

D.2 Avdelinger i Demo Telco

Den kommersielle avdeling Denne avdelingen er som nevnt delt inn i 3 underavdelinger: produkter, tjenester og administrasjon.

Produktavdelingen håndterer alt som har å gjøre med produktene i Demo Telco. Produkter skal kjøpes av leverandør og priser skal settes. Det er flere elementer en må ta hensyn til i forbindelse med produktene. Produktene i Demo Telco kan deles inn i 2 grupper: 1. Mobiltelefoner, tilbehør og batterier og 2. Telesystemer og smartbutikker.

Det som skal håndteres i produktavdelingen er følgende: Nye priser, rabatt og reklame, grupper og sett av produkter, demontering av kjøpte varer (f.eks. selge kun mobiltelefon selv om det hører en handsfree til), gaver (motta gaver og gi bort gaver), ordre til leverandør, håndtering av data om leverandører, avtaler med leverandører og kjøpe varer på konnossement [1] (må gjøres i forhold til gitte protokoller).

En av aktivitetene i Telco, foruten om salg, er sørvis[2]. Sørvisene i selskapet kan deles i to kategorier: 1. Batterisørvis og 2. sørvis på telefoner.

Sørvisen på telefonene kan igjen skje på to måter: sørvis i butikkene eller ved sørvis sentralt. I Telcos sørvisavdeling finner vi følgende business case: Motta defekte telefoner i butikkene, fra kundene, sende og motta ødelagte telefoner til og fra sentralen for reperasjon, sende

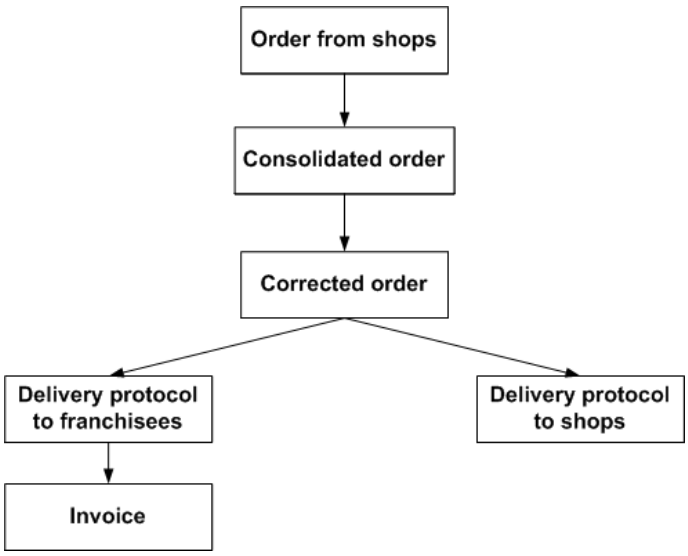
[1] dokument som utstedes av el. på vegne av bortfrakteren ved internasjonal sjøveis befordring som bevis på at gods av bestemt art og mengde er mottatt (mottaks-k.) el. innlastet til befordring til oppgitt destinasjonssted (ombord-k.). (Fra cappelen.no)

[2] Sørvis er definert av språkrådet som den norske betegnelsen for service

og motta til og fra leverandør for reperasjon, bestille reservedeler fra sentralen, bestille reservedeler fra leverandøren, erstatning av reservedeler og telefoner, skraping av ødelagte telefoner og reservedeler og utstede dokumenter for garantikrav.

Administrasjonen har følgende business case: Forhåndsbetalte SIM kort aktiveringer og SIM kort aktiveringer betalt i etterkant.

Logistikkavdelingen har ansvaret for å levere og motta produkter. Telco har ikke sitt eget varehus, og derfor bruker de Orbit Ltd for lagring og levering av sine produkter. Av den grunn blir mye av funksjonen til logistikkavdelingen å bestille og håndtere Orbit. De ulike business casene i denne avdelingen er: Motta produkter , retur, ordre fra butikker og franchiser (se figur D.2), ordre fra forhandlere, beholdning på veien (virituelt varehus), bevegelse av produkter mellom butikker, importering av filer (innkjøpsdokumenter) i butikkene og varelagerhåndtering. Prosessene i logistikkavdelingen har også en del med ordrehåntering å gjøre.



Figur D.2: Ordre fra butikker og franchiser

Finansavdelingen har en kontrolleringsfunksjon ovenfor de andre avdelingene. Hovedoppgavene består av regnkspasføring, lage rapporter og forberede og overvåke budsjettene til de ulike avdelingene. I finansavdelingen har vi følgende business case: Grafisk fremstilling av konti vedlikehold, kostnadshåndtering for solgte varer, kostnadshåndtering for innkjøp, avgiftsdokumenter for innkjøp, motta penger, gi penger, pengebeholdningsflyt, kredittkontroll, innkjøp av fastsatte ressurser, avskrivning av fastsatte ressurser, valutahåndtering, avgiftsrapport eller skatterapport, kostnadssentre, bonuskalkulering (bonus fra TelCarrier for aktivering av SIM kort) og kompansering for franchiser.

Avdelingen Sunlight Denne avdelingen er omtrent et eget selskap. Sel-

skapet selger hovedsakelig batterier og driver med vedlikehold av batterier. Sunlight har hovedsakelig med store kunder å gjøre, og på grunn av det har kontrakter som må håndteres. I Sunlight har vi fire business case: Ordre til Hellas, ordre til lokale leverandører, opprette og vedlikeholde data om leverandører og ordre fra kunder. Her har vi også med ordre å gjøre, men da en annen form for ordre.

IT avdelingen IT avdelingen har ansvaret for systemadministrasjon, importering og eksportering av data i ulike databaser og lage rapporter som andre avdelinger trenger. Her har vi kun to business case: Eksporter og importer data til og fra butikker og utveksling av data mellom Telco og Orbit.

Salgsavdelingen er beskrevet i detalj i neste avsnitt.

D.3 Salgsavdelingen

Salgsavdelingen i Telco har ansvaret for salg gjennom butikker rundt omkring i landet, franchiser og forhandlerne. Denne avdelingen håndterer kundene, forhandlerne og franchisene. Hovedprosessen i salgsavdelingen består i at 1. interessenten plasserer en ordre hos salgsavdelingen og 2. salgsavdelingen sjekker tilgjengelighet av produktet og fullfører ordren. Denne prosessen vil vi nå se på i mer detalj basert på business case beskrevet i [36].

Initiell plassering av produkter i butikkene og franchisebutikkene. Når det introduseres et nytt produkt som skal legges ut for salg i butikkene, sender salgsavdelingen ut lister over distribusjon til logistikkavdelingen. Her står det gitt hvor mange av hvert produkt de ulike butikkene skal ha.

Ordrekorreksjoner for butikk. Telcobutikker sender ordre til hovedavdelingen når de trenger produkter. Butikkene kan sende inn ordre hver dag, men de oppfylles kun en gang i uka. Dersom det er nok tilgjengelig antall av ønsket produkt blir ordren oppfyllt, men når det ikke er tilstrekkelig antall av ønsket produkt, må salgsavdelingen avgjøre hvilke ordre som skal oppfylles. For at det skal være mest mulig rettferdig for alle butikker, har salgsavdelingen regler for oppfylling av ordre. Tanken er at alle ordre skal oppfylles så godt som mulig, proporsjonalt med antallet de bestilte. I tabell D.1 ser vi et eksempel på distribusjon av 3 tilgjengelige produkter for butikker.

	Bestilt	Oppfylt
Butikk 1	3	2
Butikk 2	2	1

Tabell D.1: Korreksjon av ordre fra butikk

Begrenset kvantitet for forhandlere. Forhandlere bestiller til Telco hver dag. Deres ordre blir oppfyllt øyeblikkelig, dersom det er nok tilgjengelig. Det er derfor salgsavdelingen alltid bør vite eksakt tilgjengelig kvantitet og reservere en del av det til forhandlere.

Ordre fra forhandlere. Det er omtrent 300 ordre fra forhandlere i måneden. Salgsavdelingen plasserer disse ordrene. De følger oppfyllingen av dem, ser om det er en faktura og om kvantiteten er oppfyllt. For at hele ordren skal bli oppfyllt, skal ordren sendes til finansavdelingen for kreditsjekk. Dersom verdien går over kredittgrensen, går ordren tilbake til salgsavdelingen for korreksjoner.

Holde orden på kundeinformasjon. Telco håndterer omtrent 400 kunder. De holdes orden på ved hjelp av en 8-sifret kode som unikt identifiserer hver kunde. Salgsavdelingen håndterer data om forhandlere og franchiser. Foruten om koden og data som er påkrevd av loven, trenger de noe tilleggsinformasjon om kunden. Hver kunde er relatert til et distrikt og til en bestemt representant.

Kontrakter, avtaler med kunder. Telco har kontrakter med noen av deres kunder. Kontrakten inneholder varighet, betalingsformer og rabatter.

Returordre fra kunde. Av og til ønsker kunder å returnere enkelte varer tilbake til Telco. I det tilfellet vil salgsavdelingen gi sin tillatelse for denne returen. De godkjenner returprotokolloen. Når det er en forandring i prisen i løpet av den perioden fra salget til returen, vil salgsavdelingen bestemme prisen for retur.

Håndterig av salgspersonell. Salgsavdelingen holder orden på avtalene som salgsrepresentantene håndterer. Bonuser blir kalkulert for hver representant. Bonusene er basert på ulike kriterier: turnover, kvantitet på salg av SIM-kort, kvantitet på salg av mobiltelefoner og tilbehør. Bonuser blir kun kalkulert dersom tre av kriteriene blir oppfylt.

D.4 Demo Telcos behov

DemoTelco har behov for et informasjonssystem som kan hjelpe dem i hverdagen. På bakgrunn av business casene ser vi at vi trenger et IT system som håndterer innkjøp, salg, varehus, finans og faste ressurser. Det må også håndtere rapporter for ulike nivå.

Tillegg E

Semantics of Business Vocabulary and Rules (SBVR)

SBVR er svaret på en RFP fra OMG som BRT har laget. BRT er navnet på en gruppe bestående av ulike aktører, bl.a. personer fra BEI[1] og BRG[2]. Bakgrunnen for at SBVR var ønskelig var det faktum at forretningsregler var et konsept som også OMG ønsket å se nærmere på, i forbindelse med arbeidet rundt MDA. Det var mye diskusjoner rundt dette, men forretningsregler var ikke særlig standardisert. Det som da ble ønskelig var at forretningsfolk skulle kunne definere regler med et språk som de selv forsto. Disse reglene kunne da brukes til å styre bedriften. RFP-en [25] som da ble formulert foreslo å lage følgende for å standardisere opprettelsen og bruken av forretningsregler:

- En metamodell som forretningsfolk kan bruke til å spesifisere forretningsregler.
- En metamodell for å uttrykke vokabularer og definisjoner av termene som brukes i forretningsregler.
- Representasjon med XML.

Svaret på denne RFP-en resulterte i SBVR [53]. Dette avsnittet presenterer SBVR og fundamentene for metamodellen.

E.1 Fem aspekter i SBVR

SBVR definerer semantikken for forretningsvokabular og for forretningsregler. Med SBVR får vi beskrevet hele virksomheten med forretningsregler, konsepter, fakta o.l. dersom vi ønsker det. Vi skal nå se nærmere på de fem hovedaspektene i SBVR, men først må vi se på hva vokabular og regler egentlig er. De fem hovedaspektene er illustrert i figur E.1 på neste side.

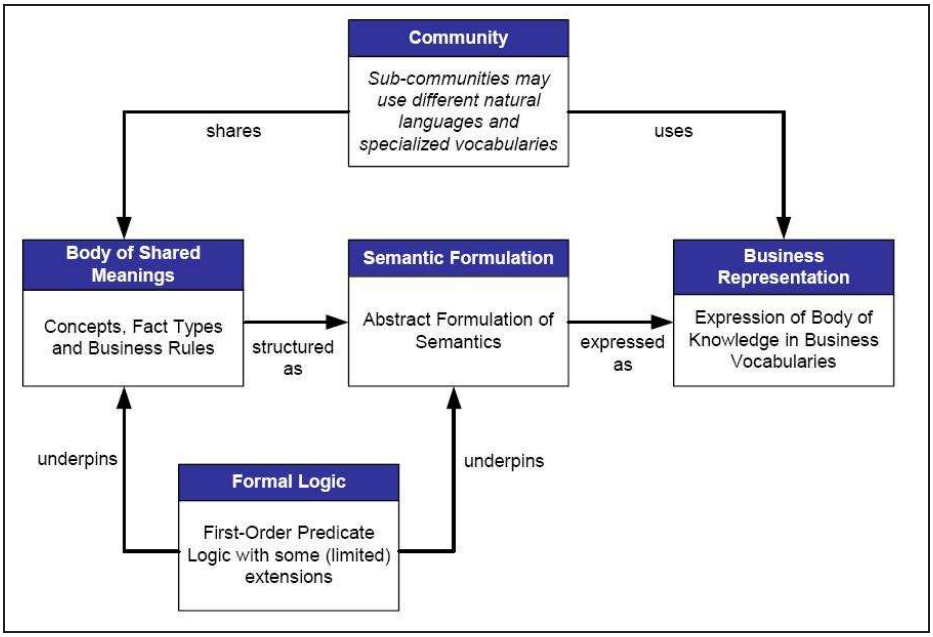
[1] www.omg.org/bei
[2] www.businessrulesgroup.org

E.1.1 Vokabular og regler

Semantikk er "meningen eller forholdet til meninger eller et tegn eller en samling av tegn" [1]. I SBVR kan tegnene være på hvilken som helst form: ord, fraser, koder, nummer, ikoner, lyder etc. SBVR har to spesialierte vokabularer: *"Vocabulary for Describing Business Vocabularies"* og *"Vocabulary for Describing Business Rules"*.

Et forretningsvokabular inneholder alle spesialtermer og definisjoner av konsepter som en gitt organisasjon eller samfunn bruker når de snakker sammen om denne forretningen eller når de skriver om den. I DemoTelco har vi konsepter fra telekomindustrien, og disse konseptene skal da forretnignsvokabularet spesifisere. SBVR gir et kraftig metode og metamodell for å definere disse elementene, samtidig som det gjør det enkelt å beskrive vokabularet til en virksomhet.

SBVR definerer en forretningsregel til å være: "a rule that is under business jurisdiction" [1]. Å være under forretningsdomsmyndighet vil si at det er regler som gjelder for denne virksomheten. En regel er definert til å være et element av veiledning som introduserer en plikt eller en nødvendighet. Regler som kan være relevante for en virksomhet er f.eks. regler som sier noe om hvordan prisen på et produkt beregnes.



Figur E.1: Hovedaspektene i SBVR [1]

E.1.2 "Community"

Basisen for et forretningsvokabular er samfunn. Samfunn som er av primær interesse er virksomheter hvor forretningsreglene blir etablert, når vi ser på virksomhetsnivået. Et samfunn ha flere undersamfunn (subcommunities).

Disse undersamfunnene kan f.eks. være avdelinger i en stor virksomhet som igjen kan være plassert i ulike land. I tillegg kan vi definere hvilke språksamfunn vi har. Dersom en del av en virksomhet er plassert i Norge, vil gjerne det samfunnet ha to språksamfunn tilknyttet seg; norsk og engelsk språksamfunn.

E.1.3 "Body of Shared Meanings"

Et samfunn har et sett av felles ord og uttrykk, som omfatter konsepter og forretningsregler. Hva som deles er meningen, ikke formen av utsagnet. For at felles meninger skal kunne utveksles, må de først uttrykkes. SBVR separerer meningen fra en bestemt form på utsagnet. Strukturen i denne 'Body of shared meanings' er definert ved å samle abstrakte konsepter, fakta typer og forretningsregler, ikke ved å samle utsagn på et bestemt språk.

E.1.4 "Semantic Formulation"

Logisk formulering gir en formell, abstrakt, språkuavhengig syntaks for å fange semantikken til Body of Shared Meanings. Det støtter multiple former av representasjoner i begge retninger.

Logisk formulering støtter to vesentlige kjennetegn i SBVR. Først er det mappingen av 'body of shared meanings' til vokabularer brukt i samfunnene. Det andre er mapping til XML Metadata Interchange (XMI) som gjør det mulig å utveksle konsepter, fakta og forretningsregler mellom verktøy og også mellom virksomheter.

E.1.5 "Business Representation"

Konseptene og forretningsreglene i 'body of shared meanings' trenger å bli representert i vokabularer som er akseptert av, og forståelige for, språksamfunn som deler deres mening. Disse vokabularene kan formuleres i forskjellige naturlige språk, i kunstig språk som f.eks UML, eller i spesialiserte subset av naturlige språk som f.eks. brukes av ingeniører.

E.1.6 "Formal Logic"

SBVR har et grundig teoretisk grunnlag av formell logikk, som ligger under logisk formulering og strukturene i 'body of shared meanings'. Denne basen er førsteordens predikat logikk, med noe begrensede ekstensjoner til modal logikk. Den formelle logikken benyttes som oftest i forbindelse med mapping til Meta Object Facility (MOF)/XMI.

E.2 Fundamentene for SBVR metamodell

Metamodellen for SBVR bygger på standardiserte måter å fomulere seg på. Hovedgrunnen for en så høy grad av standardisering er for å kunne gjøre mappingen til MOF/XMI på en enkel måte. Grunnen til at en ønsker å representere forretningsregler ved MOF-modeller eller XML dokumenter er for å lettere kunne utveksle forretningsregler på tvers av plattformer. I august innleveringen [1] hvor hele svaret på RFPen "Business Semantics of Business Rules"[25] er beskrevet, poengterer de fem viktige punkt som må implementeres av SBVR for at den skal kunne stemme overens med spesifikasjonen. Denne publikasjonen er nå offisielt inkludert i OMG. De fem punktene er:

1. Logical Formulation of Semantics Vocabulary
2. Business Vocabulary
3. Business Vocabulary and Rules
4. MOF 2 Generation Vocabulary
5. XMI Generation from Vocabulary

E.2.1 Logical Formulation of Semantics Vocabulary

For å diskutere den semantiske strukturen som ligger til grunn for konsepter, fakta og regler som det snakkes om i en bedrift, trenger vi logisk formulering. F.eks: Forretningsfolk snakker ikke om konjuksjoner, negasjoner o.l., men dette er logikken som ligger bak deres tankegang. SBVR gir ikke et logikkspråk for å formulere forretningsregler, men det gir oss en måte å beskrive strukturen og meningen av reglene i et naturlig språk som forretningsfolk bruker. For å eventuelt kunne formulere utsagn logisk anvendes førsteordens predikatlogikk. Ved å bruke SBVR blir meningen til en definisjon eller et utsagn meddelt som fakta om den sematiske formuleringen til meningen, ikke som gjengivelse i et formelt språk som kun et utvalg av personer forstår.

E.2.2 Business Vocabulary

Når vi ønsker å beskrive forretningsvokabularet til en virksomhet, trenger vi et felles sett med ord for å beskrive det. En full beskrivelse av forretningsvokabularet involverer dets forhold til semantiske samfunn og språksamfunn, dets forhold til andre vokabularer, konseptene representert, deres definisjoner og annen informasjon om dem.

E.2.3 Business Vocabulary and Rules

"Rules build on facts, and facts build on concepts as expressed by terms" [23]. Denne regelen er tankegangen bak SBVR; regler er bygged på fakta, og fakta bygger på konsepter. Dette hører til en av dem fem hovedaspektene, "Body of Shared Meanings", og går under business vocabulary and rules.

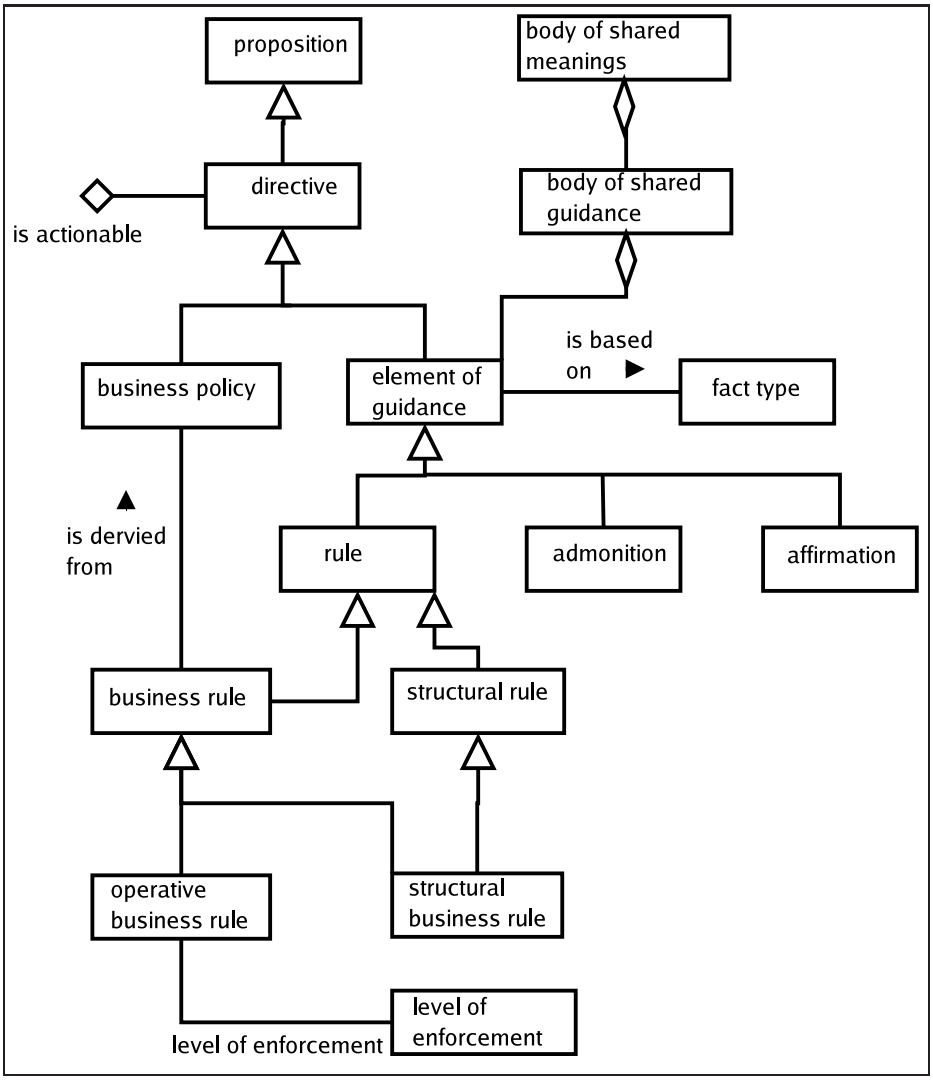
Et konsept er en enhet av kunnskap som er laget av en unik kombinasjon av karakteristikker [1, side 71]. Undergrupper av konsepter er 'noun concept' og 'fact type'. Man definerer alle konsepter ved å si hvilken concept type de er. De mest sentrale konsepttypene er *'role'*, *'individual concept'*, *'characteristic'* og *'binary fact type'*. En rolle ('role') er et *'noun concept'* som korresponderer til ting basert på at de spiller en del, påtar seg en funksjon eller blir brukt i en situasjon. Alle roller må høre til minst en fact type. Et individuelt konsept (*'individual concept'*) er et konsept som korresponderer til kun et objekt (ting). F.eks. Konseptet 'Oslo' er et individuelt konsept, da instansen av konseptet er en bestemt by i Norge. En karakteristikk (*'charcteristic'*) er en fakta type som har eksakt en rolle tilknyttet seg. Men en må merke seg at en karakteristikk allikevel kan defineres med fakta typer som har multiple roller, så lenge den fakta typen man definerer har kun en rolle tilknyttet seg så er det en karakteristikk. En binær fakta type (*'binary fact type'*) har eksakt to roller tilknyttet seg. Et eksempel på dette er "tall er større enn tall".

På bakgrunn av konsepter og faktatyper får vi forretningsregler. Figur E.2 på neste side viser hvordan en forretningsregel forholder seg til andre konsepter i SBVR, og hvordan forretningsreglene kategoriseres. Fra denne figuren ser vi også hvordan reglene defineres som en del av body of shared meanings.

En forretningsregel kan kategoriseres som strukturell forretningsregel eller operativ forretningsregel. En strukturell forretningsregel er en regel som har den hensikt å opptre som et kriterium for definisjon. En operativ forretningsregel er en regel som har den hensikt å produsere en passende designeffekt. Man må ikke skille mellom disse kategoriene, men dersom man finner det hensiktsmessig bør man gjøre det.

En forretningsregel er en spesialisering av regel som igjen er en spesialisering av et element av veiledning. Det gir oss igjen at en forretningsregel er basert på en "fact type". Ikke alle faktatyper som vi spesifiserer blir til forretningsregler. Ut fra forretningspolicien til virksomheten kan vi utarbeide forretningsreglene. Dersom bedriften har en klart definert forretningspolicy, blir jobben med å identifisere alle forretningsreglene mye enklere.

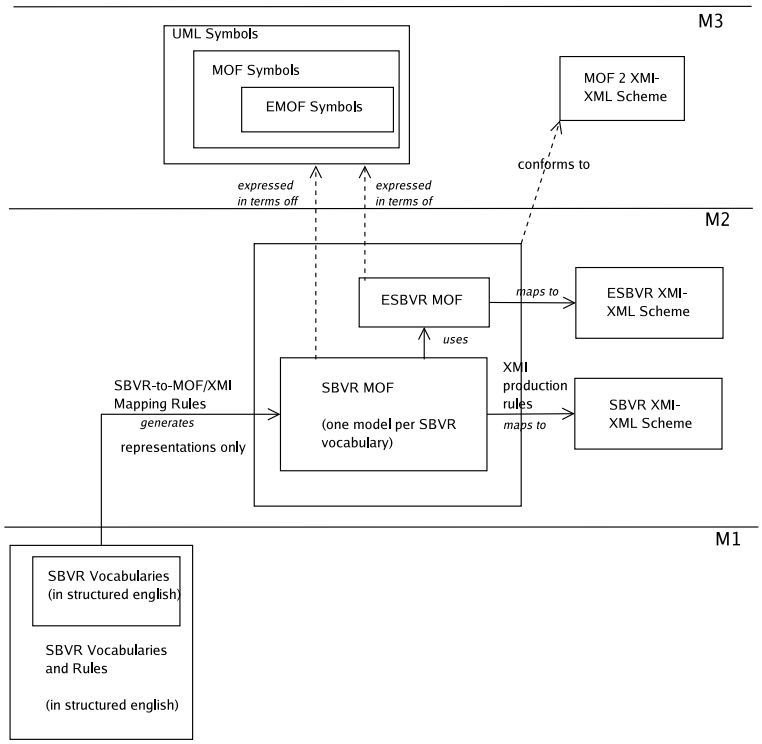
Et annet aspekt fra figuren (figur E.2 på neste side) som er viktig å fremheve er "level of enforcement". Alle operative forretningsregler kan settes til en gitt grad av håndhevelse. De ulike nivåene som anvendes i SBVR er (engelske termene er angitt i parantes): streng (strict), utsatt (deferred), pre-autorisert (pre-authorized), (post-justified),tilegne seg (override) og styringsregel (guideline).



Figur E.2: Forretningsregler i SBVR [1]

E.2.4 MOF og XMI

I SBVR har vi "Vocabulary-to-MOF/XMI Vocabulary" og "Vocabulary-to-MOF/XMI Mapping Rule Set". Vokabularene inneholder regler for transformering av et hvilket som helst vokabular, definert i form av "Logical Formulation of Semantics Vocabulary", til MOF-modell og et hvilket som helst XMI-basert skjema. Dette gjøres ved å definere en rekke transformasjonsregler. Se forøvrig [1] for nærmere dokumentasjon av dette. Figuren E.3 viser hvordan transformasjonen skjer.



Figur E.3: Mapping til MOF/XMI [1]

E.3 SBVR i systemutviklingsprosessen

SBVR er utviklet med MDA-tanken i bakhodet. OMG lagte RFPen i forbindelse med deres arbeid på MDA. MDA består av 3 elementer: Computationally independent model (CIM), platform independent model (PIM) og platform specific model (PSM). I forretningsdelen av MDA (CIM) finner vi da innslag av SBVR.

BRT som har utviklet SBVR har sett på hvordan SBVR mapper til PIM. De ser på SBVR som viktig input til PIM, men ikke fullstendig. Foruten om forretningsregler trenger man også å modellere forretningsprosesser, bruker interface og arbeidsflyt. Med andre ord, en full virksomhetsmodell er en viktig input for å lage arkitekturmodeller for informasjonssystemer.

Selve transformasjonen fra SBVR til PIM har BRT sett litt på. Forretnings konsepter (inkludert fakta typer) kan transformeres til et UML klassediagram. Noen av konseptene i SBVR vil mappe til klasser, andre til attributter. Enkelte strukturelle forretningsregler vil mappe til constraints som kardinalitet o.l. OCL vil være et naturlig språk for å drive med slike transformasjoner.

Tillegg F

Teknologier for regelmotorer

Det finnes mange ulike regelmotors prdukter på markedet i dag, de fleste tilbyderne er kommersielle og vi må derfor ut med en betydelig sum for å anskaffe seg det. Foreløpig er det svært få open source tilbydere. Per i dag er det ikke mange tilbydere som har tatt til seg OMGs standardiseringer på dette området, både når det gjelder SBVR og PRR, men erfaring tilsier at det bare er snakk om tid før det er støtte for standarder i mange av produktene. OMGs standardiseringer har alltid hatt mye å bety for utvikling av teknologier.

Jeg har sett på noen av tilbyderne og vil her gi en kort oversikt over dem. De som jeg har hatt mulighet for å teste ut gir jeg også en beskrivelse av min erfaring med dem. Først skal vi se nærmere på hve en regelmotor er.

F.1 Regelmotor

En regelmotor (rule engine), er et software system som hjelper til med håndtering av regler [73]. En regelmotor kan hjelpe oss med å registrere, klassifisere og håndtere forretningsregler; den kan verifisere konsistens mellom regler; slutte seg til regler basert på de registrerte reglene; og den kan relatere enkelte regler til applikasjoner som blir påvirket av reglene eller som trenger å bruke en eller flere regler for å fungere.

En vanlig regelmotor består av tre elementer: slutningsmekanisme (inference engine), en regelbase og et virksomt minne (working memory)[21]. Slutningsmekanismen er den viktigste delen av en regelmotor, denne delen legger regler til data. Den slutter seg til hvilke regler som skal anvendes hvor. Det er med andre ord denne som kontrollerer hele regelbruken i en rule engine. Alle reglene blir lagret i regelbasen. Denne basen inneholder alle regler som systemet kjenner til. Hvordan reglene blir lagret er forskjellig fra regelmotor til regelmotor. Noen ganger lagres reglene som tekst, men som oftest er det en regelkompilator som kompilerer reglene til et format som er enkelt for systemet å tolke [21]. Dataene i systemet som regelen skal

operere på ligger i det virksomme minnet. Dette minnet blir av og til kalt for faktabasen. Hva som lagres i det virksomme minnet er litt forskjellig i de ulike regelmotors, noen kan holde objekter av en spesiell type, mens andre kan inkludere f.eks. Javaobjekter [21].

F.2 ILOG JRules

ILOG JRules er et komplett håndteringssystem for forretningsregler. En av elementene er en regelmotor, men med ILOG får vi også andre verktøy som gjør det mulig å lett kunne utvikle større regelbaserte applikasjoner.

Målene til et Busniness Rule Management System (BRMS) prosjekt vil som regel inkludere følgende [31]:

- Gjøre det mulig for kunnskapsrike forretningseksperter å skrive forretningsregler eller policies direkte ved å bruke et kjent og komfortabelt språk og utsagnsstruktur
- Støtte høy variabilitet i forretningsregler gjennom tid, produkter, domsmyndighet, kunder og andre domener.

ILOG JRules addresserer software utvikling og forretningsregler for applikasjoner ved å gi passende verktøy for hver av interessentene i prosjektet. Vi har et oppbevaringssted for regler (tree organization, metadata extension, permission management, version management, history, concurrency control, language definition and extension), JRules builder og web builder, rule engine (RETE and Sequential mode execution, debugging interface, profiling interface, J2SE og J2EE deployment) og JRules business rule eksekveringsserver. Ved å bruke Business Rule Execution Server og SOA på en J2EE plattform, kan arkitekter fremstille et hvilket som helst regelbasert beslutningsprosess som en webtjeneste eller en Enterprise JavaBeans (EJB) session eller meassage bean interface. JRules tilbyr evnen til å abstrahere business logikk bort fra en bestemt java representasjon av en forretningsmodell, og det passer perfekt i en service orientert arkitektur.

Ved å flytte virksomhetens forretningslogikk ut av tradisjonell applikasjonskode og inn i et forretningsregel håndteringssystem, gjør at man får et stort potensiale for å øke effektiviteten til en datadrevet bedrift. Men, ved å gjøre dette kreves det både at IT og forretningsinteressenter blir gitt passende verktøy i henhold til rollen de utfører, og dette verktøyet kan ILOG gi oss [31].

ILOG JRules Rule engine er delt inn i tre deler, slik som de fleste regelmotors er:

1. Ruleset
2. Working memory
3. Agenda

Ruleset er det vi har sett på som regelbasen, her lagres alle regler. Agenda er det samme som en slutningsmekanisme, den bestemmer hvilke regler som skal anvendes når, og på hvilke objekter i det virksomme minnet de skal anvendes på.

ILOG JRules regelmotor kan vi få tilgang på på flere måter. Det er laget en rekke BRMS som vi kan få kjøpt, men i tillegg finnes det også ILOG Business Rules Studio som er eclipse[1] basert, og som er gratis for bruk til akademiske sammenhenger.

F.2.1 Om ILOG utifra egne erfaringer

ILOG er enkelt å integrere med eksisterende plattformer som eclipse og RSM/RSA. Dette er programmer som de fleste har tilgang på, eclipse er ihvertfall et opensource verktøy som ligger tilgjengelig for alle på nettet.

ILOG forholder seg som nevnt til standarder som MOF og XMI, og det bør derfor være mulig å få en representasjon av reglene en utvikler i en av disse standardene og deretter kunne transformere videre det en ønsker. ILOG tilbyr i hovedsak kun tekstlig håndtering av reglene, dette er litt dumt, i og med at ved å visualisere dem grafisk med umllignende språk, så er det enklere å holde oversikt over dem. Den tekstlige håndterinen var det i den versjonen jeg hadde mulighet til å prøve ut. I de mer kommersielle verktøyene er det andre muligheter med mer grafisk håndtering av regler.

ILOG baserer seg på java, og når man utvikler javaapplikasjoner er det enkelt å inkludere applikasjonene vi får fra ILOG JRules. Derfor ser det ut til å være god støtte for å kombinere denne regelmotoren med andre større applikasjoner.

F.3 JESS

JESS er en regelmotor og et scriptespråk utviklet av Sandia National Laboratories i Livermore California, på slutten av 1990. Det er skrevet i Java, så det er et bra verktøy for å legge til reglbasert teknolgi til Javabaserte software systemer [21].

JESS har blitt brukt til å utvikle et bredt felt av kommersielle software systemer, som f.eks.: ekspertsystemer for å evaluere forsikringskrav og lånesøknader, designassistenter for å hjelpe mekanikere, servere for å eksekvere forretningsregler, spill, m.m.

Vi kan bruke JESS som en regelmotor, men også som et generelt programmeringsspråk. I og med at JESS er javabasert, får vi tilgang til alle javaklasser og -bibliotek.

[1] www.eclipse.org

Det virksomme minnet til JESS er der hvor vi representerer informasjon. Her kan vi lagre fakta, og i JESS skilles det mellom tre typer fakta; uordnede fakta, ordnede fakta og skyggefakta. Uordnede fakta er generelle fakta, mens ordnede fakta er nyttige små biter med informasjon. Skyggefakta blir brukt for å koble til en JavaBean fra en Javaapplikasjon til det virksomme minnet i JESS, slik at reglene kan brukes til ting som skjer utenfor JESS-programmet [21].

I JESS kan man hovedsaklig skrive to typer regler: "forward-chaining rules" og "backward-chaining rules". "Forward-chaining rules" oppdager konklusjonen som kan utledes fra eksisterende sett av fakta, og "backward-chaining rules" søker etter premissene fra der hvor eksisterende fakta kan utledes fra. I tillegg kan man skrive spørringer for å påvirke det virksomme minnet direkte.

Tillegg G

EU-prosjekter

G.1 ATHENA

Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications (ATHENA) [75] er et integrert prosjekt sponset av den Europeiske kommisjonen (European Commission). Prosjektets formål er å være det mest omfattende og systematiske europeiske forsknings intitiativet på området som omhandler interoperabilitet mellom virksomhetsapplikasjoner, ved å fjerne barrierer for å utveksle informasjon i og rundt organisasjoner. ATHENA prosjektet består av 19 uavhengige forskningssentere som arbeider sammen i en venture. Prosjektet ble startet 1.februar 2004, og er planlagt å vare i 36 måneder.

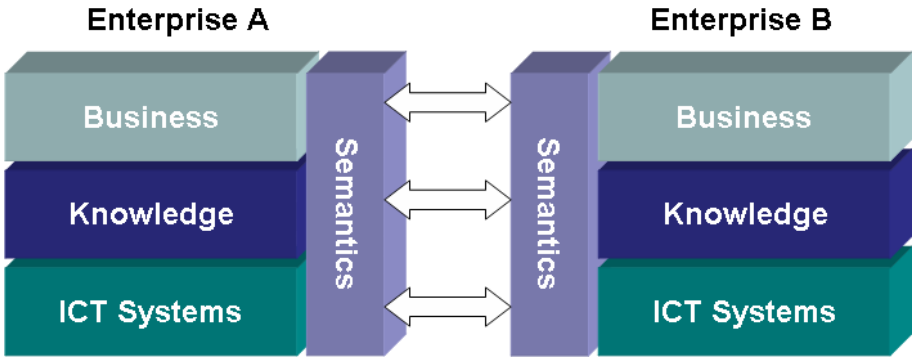
ATHENA skal møte flere mål innen integrering av virksomhetsapplikasjoner. Hovedsaklig skal de se på interoperabilitet mellom applikasjoner, men det gjelder ikke bare det tekniske i IT-systemer. Forretningsprosesser og konteksten til en virksomhet er også viktige elementer. Utfra visjonen foreslått av Ideas Consortium i 2003 : "by 2010, enterprises will be able to seamlessly interoperate with others", har ATHENA formulert dets spesifikke mål og visjon: "to be main contributor in the European efforts to enabling enterprises to seamlessly interoperate" [75].

ATHENA består av 6 forsknings- og utviklingsprosjekter som blir betegnet som "Action Line A", og 6 aktiviteter som tar for seg bygging av samfunn, som kalles "Action Line B". Hele programmet blir håndtert og gitt infrastrukturer for virksomheter og teknisk support gjennom det som kalles for "Action Line C". Vi skal kun se nærmere på action line A, da det er prosjektene her som er av størst interesse for oss.

Forsknings- og utviklingsprosjektene under Action Line A vil gi forskningsresultater som passer inn i et interoperabilitetsrammeverk. Det generelle rammeverket er en konseptuell modell som ser på de ulike aspektene ved interoperabilitet.

G.1.1 ATHENA rammeverket og A-prosjekter

Det generelle rammeverket (som vist i figur G.1) til Athena er representert ved tre hovedlag: business, kunnskap og IKTløsninger.



Figur G.1: ATHENA-rammeverket [75]

På forretningsnivået blir alle elementer som er relatert til organisasjonen og operasjonen til en virksomhet addressert. Interoperabilitet på dette nivået blir sett på som den organisasjonelle og operasjonelle evnen for en virksomhet å samarbeide med en annen. Laget inkluderer beslutningsmodell, forretningsmodell og forretningsprosesser.

Kunnskapslaget tar for seg det å forstå virksomheten i seg selv. Dette inkluderer kunnskap om interne aspekter i virksomheten som produkter, måten administrasjonen opererer og håndterer bedriften, hvordan personellet blir veiledet, o.l. Interoperabilitet på dette nivået blir sett på som likheten ved ferdighetene, kompetansen m.m.

Det siste nivået, IKT-systemer fokuserer på de konkrete løsningene som gjør at en virksomhet kan operere, ta avgjørelser, utveksle informasjon innenfor og utenfor dets grenser, o.s.v.

Underprosjektene for action line A er nomerert fra 1 til 6, og disse refereres til som A1, A2, ..., A6. Under er en liste over alle A-prosjektene.

- A1: Enterprise Modeling in the context of Collaborative Enterprises
- A2: Collaborative Business Process Execution
- A3: Knowledge Support and Semantic Mediation Solutions
- A4: Interoperability Framework and Sevices for Networked Enterprises
- A5: Planned and Customizable Service-Oriented Architectures
- A6: Model-driven and adaptive interoperability architectures

Hovedsaklig er det A1 prosjektet som er av interesse for oss, da det tar for seg temaet virksomhetsmodellering. Det generelle målet til A1 er å utvikle

metodologier, språk og arkitekturer, modellgenererte workplaces og tjenester og eksekveringsplattformer for etablering av samarbeidende utvidede virksomheter og nettverk organisasjoner. Målet vil de nå gjennom flere undermål. Det første undermålet er utviklingen av en metode for å lage modellgenererte og tilpassede workplaces. Det andre undermålet er å utvikle metoder for virksomhetsmodellering som er tilpasset individuelle brukere. Det tredje subobjektivet er å gi en virksomhetsmodellerings plattform for å direkte støtte etableringen og opereringen av samarbeidende virksomheter.

G.2 INTEROP

Interoperability Research for Networked Enterprises Applications and Software (INTEROP) [80] er et prosjekt som jobber innenfor området som har å gjøre med interoperabilitet for virksomhetsapplikasjoner og software. Deres mål er å lage betingelsene for innovativ og kunkurrerende forskning innen dette domenet [35]. Dette vil de gjøre med å slå sammen tre kunnskapskomponenter:

- **Architectures and Enabling Technologies** for å gi implementasjons rammeverk
- **Virksomhetsmodellering** for å definere krav for interoperabilitet.
- **Ontologi** for å definerer interoperabilitetssemantikken i virksomheten.

Interoperability Research for Networked Enterprises Applications and Software (INTEROP) er delt inn i 12 arbeidspakker (Work Packages (WP)) og 7 Task Group (TG). Det som er av interesse for oss er TG2 (Modeldrevet interoperabilitet).

G.2.1 Task Group 2

Formålet for denne gruppen er å analysere og foreslå løsninger for Model Driven Interoperability (MDI). Gruppen skal bidra med en metode for å muliggjøre interoperable virksomhets software applikasjoner (Enterprise Software Applications (ESA)) som starter fra virksomhetsmodelleringsnivået [13].

MDA er utgangspunktet for arbeidet til TG2. De bruker MDAs tre synspunkt; CIM, PIM og PSM. De har sett at CIM nivået korresponderer til domenet for virksomhetsmodellering. I den modelldrevne utviklingsprosessen er det mulig å utvikle et sett av forskjellige internt relaterte modeller på ulike abstraksjonsnivå ved å se på forskjellige elementer ved virksomhetsmodellering, som prosess, produkt, avgjørelser, regler, motivasjon o.l.

Tillegg H

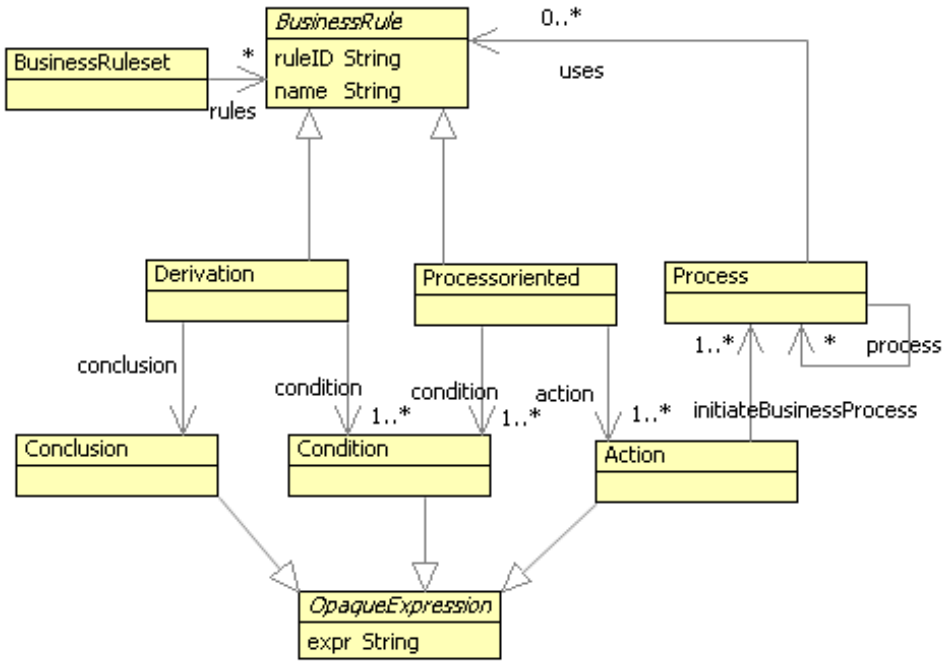
Transformasjon av forretningsregler

Dette tillegget beskriver transformasjonen gjort av forretningsregler i forbindelse med casestudiet. Først presenteres kilde- og målmodellen for transformasjonen. Deretter presenteres mappingene og alle transformasjonsreglene. Til slutt vises hva som ble transformert og resultatet av transformasjonen.

H.1 Kildemodell

Kildemodellen er metamodellen for forretningsregler. Se figur H.1 på neste side.

Metamodellen er laget på bakgrunn av definisjonene for prosessorienterte forretningsregler og slutningsregler fra forrige kapittel. Figur H.1 på neste side viser denne metamodellen. Den sentrale klassen i modellen er den abstrakte klassen *BusinessRule*. Denne definerer hva slags egenskaper en regel skal ha tilknyttet seg. Den skal ha en id og et navn. Forretningsregler kan deles i fire kategorier, hvorav to skal være med i forretningsregelmodellen. Disse to kategoriene er slutningsregler og prosessorienterte forretningsregler. De to, vist som klassen *Derivation* og *Processoriented*, er subklasser av *BusinessRule*. En slutningsregel kan ha en eller flere betingelser (*Condition*) tilknyttet seg, og en *conclusion*. Betingelser og konklusjoner består av et utsagn (*expr*) som de arver fra den abstrakte klassen *OpaqueExpression*. En prosessorientert forretningsregel kan også ha en eller flere betingelser tilknyttet seg, og kan ha en eller flere hendelser (*action*) som resultat. En hendelse kan igangsette en prosess. Klassen *process* har vi fra prosessmetamodellen (avsnitt 7.2.4). En prosess kan bruke forretningsregler underveis i prosessen, og disse kan være både slutningsregler og prosessorienterte regler.



Figur H.1: Metamodell for forretningsregler

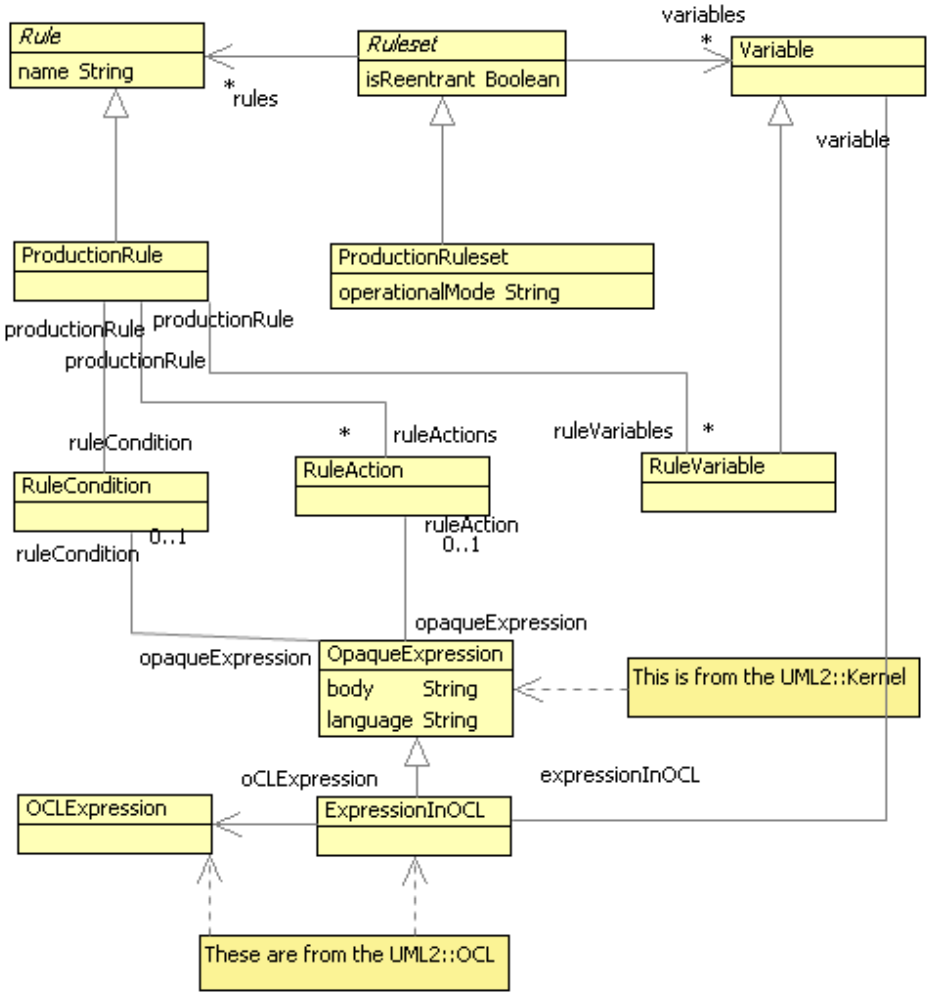
H.2 Målmodell

Målmodellen er metamodellen for produksjonsregler. Se figur H.2 på neste side.

PRR::Rule er et generelt konsept for regler. I fremtiden kan det være at man standardiserer flere typer regler, og da vil de komme som en kategori av PRR::Rule. En regel representerer en betinget del med programmeringslogikk, inkludert da produksjonsregler. Et regelsett fungerer som en beholder for regler, og gir en kontekst for eksekvering av regler. Regelsett kan ha variabler tilknyttet seg. I figur H.2 ser vi dette som PRR:Ruleset.

PRR::ProductionRuleset representerer et regelsett for produksjonsregler. Produksjonsregelsett har et attributt som beskriver den operasjonelle semantikken til regelsettet. PRR::ProductionRule er et utsagn av programmeringslogikk som spesifiserer eksekveringen av en eller flere hendelser i det tilfellet hvor betingelsene er oppfylte [54]. En produksjonsregel er på formen: *if [condition] then [action-list]*. En produksjonsregel må ha en betingelse og en eller flere hendelser tilknyttet seg.

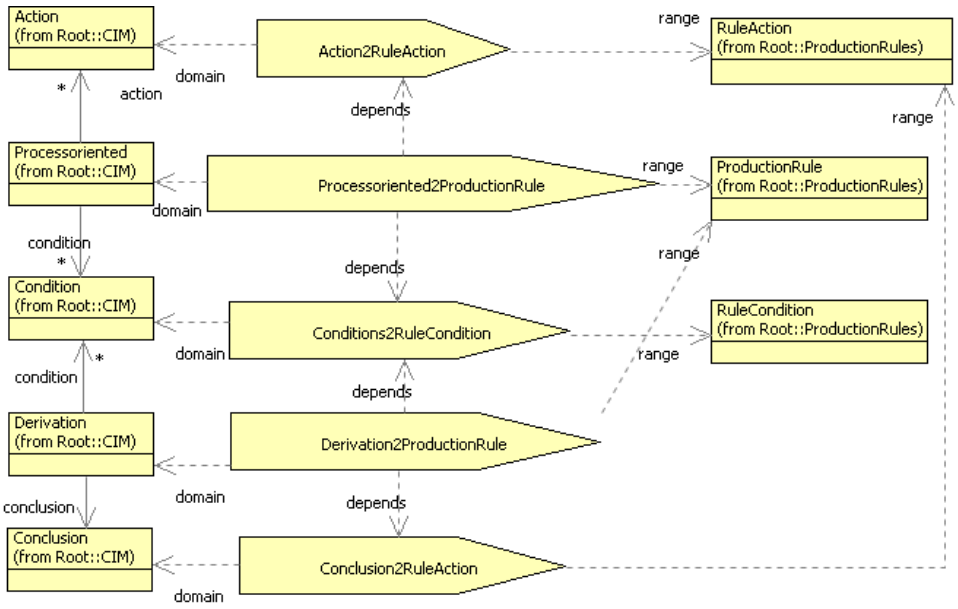
PRR::Variable representerer en programmeringskonstrukt som holder en eller flere verdier for å brukes til eksekvering av regler. PRR::RuleCondition er et boolsk uttrykk som avgjør om gitt hendelse skal utføres eller ikke. I PRR kan det kun være en betingelse knyttet til en regel. PRR::RuleAction gir en liste av ordnede hendelser som skal inntreffe dersom en gitt betingelse evaluerer til sann. PRR::RuleVariable angir domenet som skal brukes i eksekveringen av regler.



Figur H.2: Metamodell for produksjonsregler

H.3 Mappingmodell

Mappingmodellen viser hvordan elementer i kildemodellen kan transformeres til elementer i målmodellen. Figur H.3 viser mappingmodellen. Her er det definert fem mappinger. For å mappe prosessorienterte forretningsregler er "Processoriented2ProductionRule" definert. Denne går gjennom alle tilhørende betingelser og hendelser som er tilknyttet den prosessorienterte forretningsregelen, noe som vi ser i figuren med pilen *depends* som er satt mot "Condition2RuleCondition" og "Action2RuleAction". Den første mapper betingelsen for den prosessorienterte forretningsregelen til en betingelse for en produksjonsregel. Den andre mapper den hendelsen som er resultatet av en prosessorientert forretningsregel til en hendelse som er forbundet med en produksjonsregel. For å mappe slutningsregler om til produksjonsregler er "Derivation2ProductionRule" definert. Denne er laget på tilsvarende måte som for prosessorienterte forretningsregler, ved at den går gjennom betingelsene og konklusjonene som er knyttet til en slutningsregel og mapper disse over til tilsvarende elementer for produksjonsregler med mappingene "Condition2RuleCondition" og "Conclusion2RuleAction".



Figur H.3: Mappingmodell

H.4 Transformasjonsregler

Transformasjonsreglene er skrevet for å enkelt transformere en forretningsregel til en produksjonsregel, som da er representasjonen av regler som vi har på PIM-nivå. For hver mappingelement er det definert clauses, og i hver clause er det selve koden for mappingen.

Clausen under definerer hvordan en action i forretningsregelmetamodellen

transformeres til en RuleAction i produksjonsregelmetamodellen. Dette er en veldig enkel mapping, som ikke gjør noe mer enn å opprette nye objekter av RuleAction og OpaqueExpression, og setter navn og parameter slik som det er satt i forretningsregelmodellen.

Action2RuleAction

```
@Clause Action2RuleAction
  CIM::Action[expr=EX]
do
  ProductionRules::RuleAction[opaqueExpression=OE]
where
  OE = ProductionRules::OpaqueExpression[body=EX]
end
```

I denne clause, er det skrevet transformasjonskode for å transformere en *conclusion* i forretningsregelmodellen til en *RuleAction* i regelmodellen. Denne er laget tilnærmet likt som for Action2RuleAction.

Conclusion2RuleAction

```
@Clause Conclusion2RuleAction
  CIM::Conclusion[expr=EX]
do
  ProductionRules::RuleAction[opaqueExpression=OE]
where
  OE = ProductionRules::OpaqueExpression[body=EX]
end
```

Conditions2RuleCondition er mer komplisert enn de foregående transformasjonsreglene. Her skal all *conditions* i forretningsregelmodellen gjennomgås og konkateneres med ordet *and* som skilleord. Til slutt blir det til en *RuleCondition*.

Conditions2RuleCondition

```
@Clause Conditions2RuleCondition
  C->including(CIM::Condition[expr=E])
do
  ProductionRules::RuleCondition[opaqueExpression=OE]
where
  OE=ProductionRules::OpaqueExpression[
                                body=self.makeString(C)]
end

@Operation makeString(C : Set(CIM::Condition))
                                          :XCore::Element
      let expression = ""
      in @For c in C do
           expression := expression + c.expr;
           if not isLast
           then
             expression := expression + " and "
           else
             false
           end
         end;
         self.con_expr := expression;
         expression
```

```
    end
  end
```

Derivation2ProductionRule er regelen som går gjennom alle slutningsregler og lager produksjonsregler. Fra denne regelen blir de andre transformasjonsreglene kalt. Det er her *depends*-pilen er anvendt. Det samme gjelder for neste clause som er vist, *Processoriented2ProductionRule*.

Derivation2ProductionRule

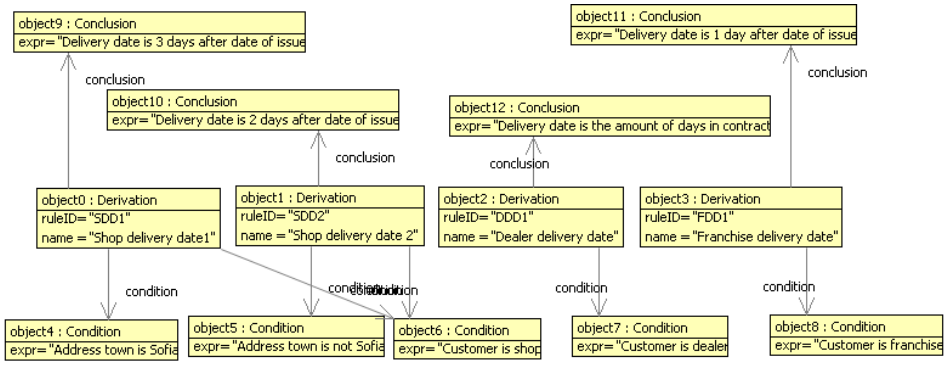
```
@Clause Derivation2ProductionRule
  CIM::Derivation[name=N, condition = C, conclusion = CON]
do
  ProductionRules::ProductionRule[name=N,
                                  ruleActions = RA,
                                  ruleCondition = RC]
where
  RA = Conclusion2RuleAction()(CON);
  RC = Conditions2RuleCondition()(C)
end
```

Processoriented2ProductionRule

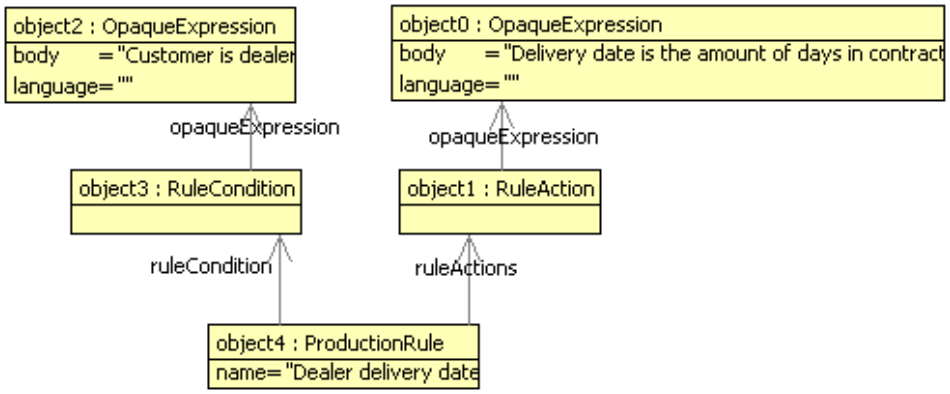
```
@Clause Processoriented2ProductionRules
  CIM::Processoriented[name=N, condition = C, action = A]
do
  ProductionRules::ProductionRule[name=N,
                                  ruleActions = RA,
                                  ruleCondition = RC]
where
  RA = A->collect(a | Action2RuleAction()(a));
  RC = Conditions2RuleCondition()(C)
end
```

H.5 Transformasjon fra CIM til PIM

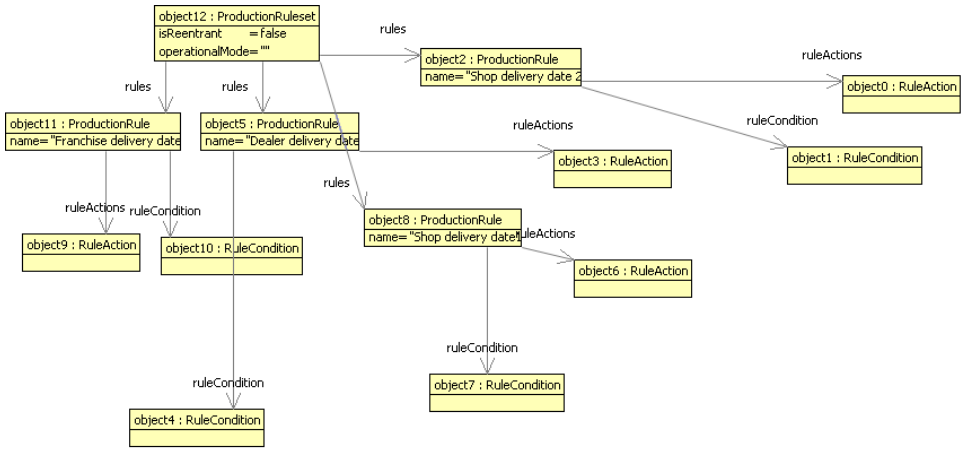
Under er det vist alle resulterende modeller etter at transformasjonsreglene som er beskrevet over er anvendt. Figur H.4 viser forretningsreglene, modellert i XMF-Mosaic, som var utgangspunktet for transformasjonen. Figur H.5 til figur H.9 viser alle produksjonsreglene som er resultatet av transformasjonen.



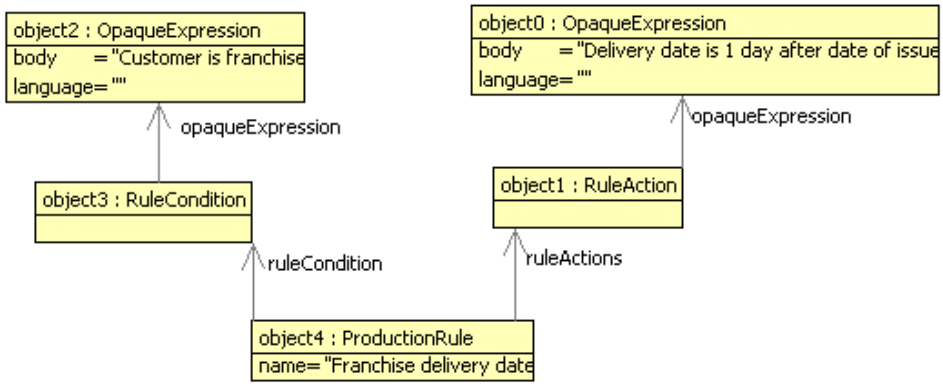
Figur H.4: Forretningsregelmodell i CIM



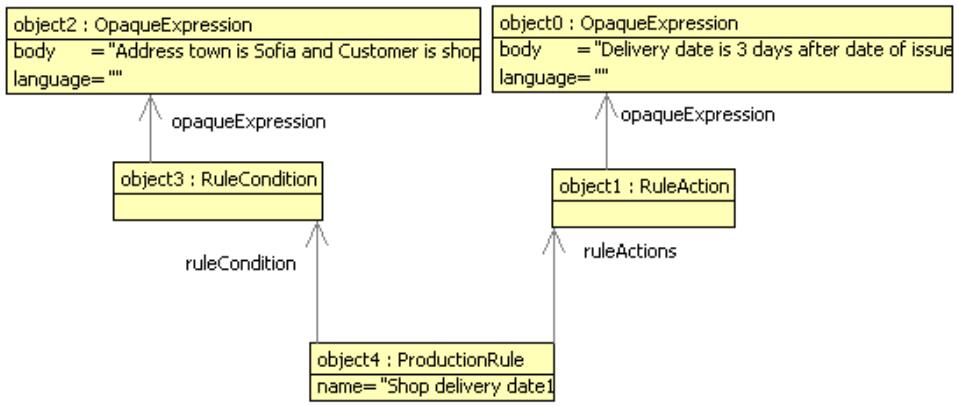
Figur H.5: Produksjonsregelrepresentasjon av DealerDeliveryDate



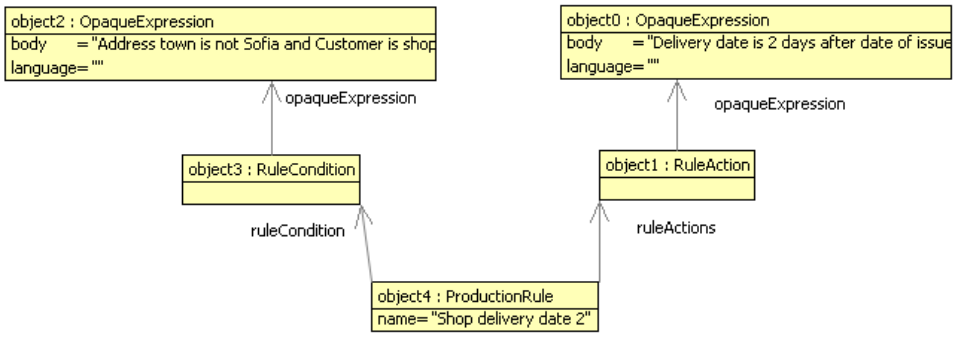
Figur H.6: Produksjonsregelrepresentasjon av leveransedatoregler



Figur H.7: Produksjonsregelrepresentasjon av FranchiseDeliveryDate



Figur H.8: Produksjonsregelrepresentasjon av ShopDeliveryDate1



Figur H.9: Produksjonsregelrepresentasjon av ShopDeliveryDate2