HÖGSKOLAN
I SKÖVDE
Institutionen för kommunikation och information
Examensarbete i Systemvetenskap 30hp
C-nivå
Vårterminen 2011

Från modell till modell

Transformering av information från ett modellspråk till ett annat

Johan Bjurén

Modell till modell

Examensrapport inlämnad av Johan Bjurén till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Anne Persson.

2011-05-23

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Förord

Detta examensarbete är skrivet vid Institutionen för kommunikation och Information (IKI) vid Högskolan i Skövde. Examensarbetet motsvarar 30hp. Arbetet har handletts av Anne Persson och examineras av Marcus Nohlberg.

Jag vill tacka min handledare för nedlagd tid, att ha stöttat mig, gett bra och innehållsrik kritik och hjälp mig när jag haft alla funderingar kring ämnet. Jag vill tacka Marcus Nohlberg för nyttig och bra feedback de gånger jag har presenterat och hjälp mig att utvecklas som en bättre presenterare. Jag vill även tacka övriga lärare på IKI som har hjälp mig i min personliga utveckling och lärande. Jag vill även tacka släkten och mina vänner för att ni har stått ut med att jag sagt nej till i princip allt roligt senaste halvåret, det kommer en tid då vi kan ta igen det. Jag vill även tacka mina klasskamrater, Lise-Lotte och Jonny, för ett gott samarbete. Ett extra tack för att ha stått ut med mig under dessa omständigheter får gå till flickvännen, tack. Ett extra tack går även till far som stöttar mig i alla lägen och till min vän Tobias Karlsson som alltid ställt upp och läst vad jag gjort och gett hård men bra kritik.

Arbetet som jag har utfört har varit en väldigt intressant och stimulerande resa för mig. Min uppfattning om området har förändrats och desto mer jag lär mig desto intressantare blir det. Jag hittar nya lösningar, fallgropar och lösningar till fallgroparna för varje rapport jag läser. Detta arbete har även lett till att jag har kunnat se på området med ”andra ögon” och får ständigt nya funderingar och idéer om olika lösningar eller hur saker hänger ihop.

Tack till er som har hjälp mig att få möjligheten att göra denna resa, imorgon börjar en ny.

Modell till modell

Johan Bjurén

Sammanfattning

Detta arbete undersöker möjligheterna att ta fram riktlinjer för en transformation från modelltyperna i EKD, som är ett mindre formellt semi-formellt modelleringsspråk, till UML, som är ett mer formellt semi-formellt modelleringsspråk, i syfte att komma närmare en systemspecifikation som kan implementeras som ett IT-system. Arbetet är utfört med en designansats som grund för hur arbetsprocessen sett ut. Processen följer genom hela arbetet och kan läsas om under metodkapitlet. Arbetet tar upp frågor som vilka modelltyper som är lämpliga att transformera till, vilken information som kan transformeras, vilken information som inte kan transformeras och vad som kan göras med denna. Riktlinjerna testas i valideringssyfte även på EKD modeller som skapats i annat syfte än för denna rapport.

Nyckelord: Modeller, Transformering, Metamodell, Modell till modell, EKD, UML.

Innehållsförteckning

1 Introduktion och problemställning 1

1.1 Frågeställning 3

1.2 Förväntat resultat 3

2 Teoretisk grund 5

2.1 Konceptuell modell 5

2.2 Modelleringsspråk 6

2.3 Användning av modeller 8

2.4 Transformation till en mer formaliserad modell 10

3 Metod och metodval 13

3.1 Kort om metodansats 13

3.2 Arbetsprocess 14

3.2.1 Inledning 15

3.2.2 Teori och metod 15

3.2.3 Analys av valda modelleringsspråk 15

3.2.4 Framtagning av riktlinjer för transformering 16

3.2.5 Test och verifiering 16

3.2.6 Analys 16

3.3 Kort om valda sätt att samla information 17

3.3.1 Dokument- och litteraturstudier 17

3.3.2 Validering 17

4 Analys av valda modelleringsspråk 18

4.1 EKD 18

4.1.1 Målmodellen (Goals Model) 20

4.1.2 Processmodellen (Business Process Model) 22

4.1.3 Aktörs- och resursmodell (Actor and Resource Model) 23

4.1.4 Regelmodellen (Business Rules Model) 24

4.1.5 Begreppsmodell (Concepts model) 26

4.1.6 Kravmodellen (Component of the Technical Components and Requirements Model) 27

4.2 UML 29

4.2.1 Modelltyper i UML 29

4.2.2 Metamodeller för UML 31

4.2.3 Modelltyper att transformera till 31

4.3 Slutsats om val av modelltyper 38
5 Riktlinjer för transformering 39
5.1 Riktlinjer för mappning av EKD modeller till use case diagram. 39
5.2 Riktlinjer för mappning av EKD modeller till Aktivitetsdiagram 42
6 Alternativ att utföra mappningen på 46
7 Förlorad information och kompletteringsbehov 48
8 Test och verifiering 52
8.1 Mappning till use case 52
8.2 Mappning till aktivitetsdiagram 56
9 Analys och diskussion 60
9.1 Utvärdering av arbetet 62

Referenser

Figurförteckning.

Figur 1:1 Arbetsprocess – Inledning 1
Figur 1:2 Processen från en idé till utvecklat IT-system 3
Figur 2:1 Arbetsprocess - Teoretisk grund 5
Figur 2:2 Formalitetsnivåer 6
Figur 2:3 Exempel på verksamhetsmodeller i EKD 7
Figur 2:4 Sekvens- och klassdiagram är två exempel vanliga diagram i UML 8
Figur 2:5 Modelleringsspråk X och Y inom området för semi-formella modeller 11
Figur 2:6 Matchning mellan metamodeller i modelleringsspråken X och Y 11
Figur 2:7 Transformering av modelleringsspråk X till Y. Endast viss data överlappar mellan språken. 12
Figur 3:1 Arbetsprocess vid framtagning av metoder 13
Figur 3:2 Arbetsprocessmodell 15
Figur 4:1 Arbetsprocess under analysen av modelleringsspråken 18
Figur 4:2 Modelltyper i EKD 19
Figur 4:3 Entitettyper i metamodeller 19
Figur 4:4 Metamodell av EKD's Målmodell 21
Figur 4:5 Metamodell av EKD's Processmodell 22
Figur 4:6 Metamodell av EKD's Aktör och Resursmodell 24
Figur 4:7 Metamodell av EKD's Regelmodell 25
Figur 4:8 Metamodell av EKD's Begreppsmodell 27

Figur 4:9 Metamodell för EKD's Kravmodell 28
Figur 4:10 Funktionellt krav isolerat i Kravmodellen 33
Figur 4:11 Krav har en koppling till processmodellen 34
Figur 4:12 Krav är kopplat till Processer i Processmodellen. Process ärver från Processer och har alltid en koppling till Aktör och Resursmodellen. 34
Figur 4:13 Processmodellen har en koppling till Aktör / Resurs i Aktör och Resursmodellen. 34
Figur 4:14 Exempel av UML's Aktivitetsdiagram 35
Figur 4:15 Den del av EKD's processmodell som UML kan visa. 36
Figur 4:16 Processmodellens knytning till Aktör / Resurs 36
Figur 4:17 Exempel på hur ett sekvensdiagram kan vara utformat 37
Figur 4:18 Processmodellen med stor del av sin funktionalitet framtagen. 37
Figur 4:19 Vägen från processmodellen till Aktör /Resurs i Aktör och Resursmodellen. 38
Figur 5:1 Arbetsbeskrivning - Framtagning av transformationsriktlinjer 39
Figur 5:2 Grund för transformation mellan EKD och UML's use case diagram 40
Figur 5:3 Mappning av Funktionella krav till use case 40
Figur 5:4 Mappning av processer till use case. 41
Figur 5:5 Mappning av Aktör / Resurs till use case 42
Figur 5:6 Grund för mappning mellan EKD och UML's Aktivitetsdiagram 43
Figur 5:7 Mappning av Processer till Aktivitetsdiagram 43
Figur 5:8 Mappning av interna relationer till Aktivitetsdiagram 44
Figur 5:9 Mappning av information till Aktivitetsdiagram 44
Figur 5:10 Mappning av Aktör / Resurs till Aktivitetsdiagram 45
Figur 6:1 Arbetsprocess – Andra sätt att utföra mappningen på. 46
Figur 6:2 Överlappande information mellan semi-formella modelleringsspråk 47
Figur 6:3 Från informell modell till färdigt system. 47
Figur 7:1 Arbetsprocess – Saknad och förlorad information 48
Figur 7:2 Delar av EKD’s Aktör och resursmodell som inte transformerats till UML 49
Figur 7:3 Delar av EKD’s Kravmodell som inte transformerats till UML 49
Figur 7:4 Delar av EKD’s Processmodell som inte kunnat transformeras till UML ... 50
Figur 7:5 Målmodellen i EKD 50
Figur 7:6 Regelmodellen i EKD 50
Figur 7:7 Begreppsmodellen i EKD 51
Figur 7:8 Exempel på UML's Aktivitetsdiagram 51
Figur 8:1 Arbetsprocess för test och verifiering 52

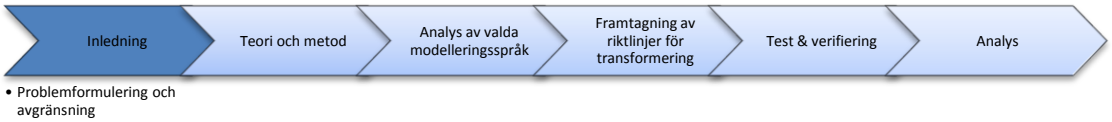
Figur 8:2 Tre utvalda modelltyper i EKD 53
Figur 8:3 Krav 53
Figur 8:4 transformerat use case 54
Figur 8:5 Process 3 54
Figur 8:6 transformerat use case och tillhörande processer 55
Figur 8:7 Aktörs- och resursmodellen 55
Figur 8:8 use case med tillhörande aktörer/resurser och en beskrivning. 55
Figur 8:9 EKD relationer 56
Figur 8:10 EKD Process 57
Figur 8:11 EKD Aktör och resurs 57
Figur 8:12 Transformerad modell från EKD till UML 58
Figur 8:13 Transformerad modell med korrekt notation för alternativ 59
Figur 9:1 Arbetsprocess i analys och diskussion 60

Tabellförteckning

Tabell 3:1 Riktlinjer för designansats och hur detta arbete följer dem 14
Tabell 4:1 Relationer i Målmodeller 21
Tabell 4:2 Relationer i processmodell 23
Tabell 4:3 Relationer i Aktör och Resursmodellen 24
Tabell 4:4 Relationer i Regelmodellen 26
Tabell 4:5 Relationer i begreppsmodellen 27
Tabell 4:6 Relationer i Kravmodellen 29

1 Introduktion och problemställning

Arbetet kommer ske utifrån en framtagen arbetsprocess som beskrivs under rubriken 3.2 Arbetsprocess. Under detta kapitel kommer inledning att tas upp.



Figur 1:1 Arbetsprocess – Inledning

Johannesson (2007) tar upp problemet med att dagens IT-system möter en mer komplex omgivning och för att kunna hitta lösningar på de problem som kan komma av detta kan organisationen använda sig av olika verktyg. Ett av dessa verktyg är konceptuella modeller. Med hjälp av dessa konceptuella modeller kan organisationen få hjälp att finna lösningar på problem, att strukturera upp, få en översikt, behålla informationen i organisationen och göra modeller som är formella nog för att bygga IT-system på skriver Bubenko, Persson & Stirna (2001).

Språk för konceptuell modellering riktar ofta in sig på en specifik fas under utvecklingen. Vissa faser kräver att modellen har relevant grad av formalitet för att modellen skall vara lätt att förstå men ändå ha en viss grundstruktur och följa ett regelverk. Krogstie, Sindre och Jorgensen (2006) tar upp olika typer av kvalité en modell kan ha och innebörden av dessa. Bland annat tar de upp syntaktisk kvalité. Med syntaktisk kvalitet menas att modellen är korrekt gentemot modelleringsspråkets syntax. De menar att för att en modell skall nå hög kvalité måste denna vara syntaktiskt korrekt. Ruiz, van Harmelen, Aben, & van de Plassche (1994) tar upp vikten av att en modell är korrekt. De förklarar att en modell inte kan bli transformerad till en modell med högre formalitet om den inte är syntaktisk och semantiskt korrekt, det vill säga att modellen hänger ihop på ett korrekt sätt, är relevant mot problemet är valid och komplett. Krogstie, et al. (2006) tar även upp, pragmatisk kvalité, vilket enligt dem innebär att alla som är med och modellerar behöver förstå och kunna tolka modellen. Modelleraren kan inte utgå från att deltagarna är införstådda med modellens syntax. För att modelleraren skall kunna arbeta med en viss grupp av människor som kanske inte är specialiserade på alla områden i organisationen utan har mer en övergripande bild av hur det ser ut krävs att modellen som modelleraren arbetar med är informell nog för att deltagarna i en modelleringssession inte ska hämmas av det uppsatta regelverk som finns. Modellen måste alltså vara lagom formell för ändamålet.

Martín-Vivaldi och Isacsson (1988) tar upp fördelarna med att börja med en mer informell modell när en verksamhet skall modelleras. De menar att det är av stor vikt att veta vilken grad av formalitet en modell skall ha för att kunna fånga det mesta möjliga utifrån den fas i skapandet av ett IT-system som organisationen befinner sig i. För att kunna komma ner på den nivå där system och data skall struktureras behöver

modelleraren ofta gå från en relativt informell modell till en mer formell modell. Bubenko, et al. (2002) tar upp att problemet med att det saknas riktlinjer för att gå från ett mindre formellt modelleringsspråk till ett mer formaliserat modelleringsspråk är ett faktum. De skriver ”We maintain that the challenging issue of how to proceed from informal, fuzzy individual statements of requirements to a formal specification that is understood and agreed by all stakeholders remains unanswered. This view is re-enforced by empirical evidence.”. Detta arbete kommer att titta på två utvalda modelleringsspråk som inte har riktlinjer för att utföra en transformation mellan dem.

Målet med mitt urval av modelleringsspråk är att kunna studera hur övergång kan hanteras från modeller som verksamhetens representanter kan förstå till modeller som lämpar sig som underlag för att bygga IT-system. Modellerna detta arbete kommer att ta upp är därför semi-formella modeller. Inom området för semi-formella modeller finns det olika formalitet. Det finns de med högre formalitet och de med mindre formalitet. Det modelleringsspråk som valts som representant för mer informella semi-formella språk är Enterprise Knowledge Development (EKD) (Bubenko, et al., 2001). Detta modelleringsspråk har valts då det är tänkt att användas för verksamhetsmodellering. Enligt Bubenko, et al. (2001) är EKD ett modelleringsspråk som är lämpligt att använda som grund för att bygga ett IT-system på. Unified Modeling Language (UML)[1] valdes som det modelleringsspråk som representerar den mer formella semi-formella typen av modelleringsspråk. UML är ett modelleringsspråk som är väl utbrett och även om inte alla använder UML för systemutveckling är notationen vanlig nog för att systemutvecklare normalt inte skall ha problem att förstå och tolka den. Eftersom det går att skapa programmeringskod direkt utifrån UML-modeller anser jag att det är ett lämpligt språk att transformera till om målet är att gå mot utvecklingen av ett IT-system eftersom det finns och pågår projekt, bland annat inom Model Driven Development, MDA, (Brown, 2004), för att omvandla denna typ av modeller till system.

Det finns flera vägar att gå för att utveckla ett IT-system. Denna rapport har tagit upp några av fördelarna med att använda sig av konceptuella modeller. I den process som visas i Figur 1:2 illustreras hur information går från en idé eller en kunskap till en mer formaliserad modell allteftersom utvecklingsprocessen fortlöper och modellen blir till sist formaliserad nog för att kunna implementeras. Idag får den mer informella modellen tolkas av den som transformerar modellen till en mer formell modell. För att kunna nå en modell med högre formalitet behövs ofta att ytterligare information tillförs av den som har kunskapen, ofta en kund. Bubenko, et al. (2002) påtalar att det finns ett tomrum mellan mindre formaliserade och mer formaliserade modelleringsspråk. Detta ger att det är av vikt att undersöka detta problem för att se vilken information som går förlorad i transformation mellan de utvalda modelleringsspråken och hur det går att säkerställa att komplettering kan ske.

[1] http://uml.org/

Figur 1:2 Processen från en idé till utvecklat IT-system

Detta examensarbete kommer alltså att undersöka olika modelleringsspråk och analysera vilka möjligheter det finns att gå från EKD, som är ett mer informellt semi-formellt modelleringsspråk, till UML, som är det mer formella semi-formella modelleringsspråket. Målet är att ta fram riktlinjer för hur en transformation mellan dess två typer av modelleringsspråk skall kunna göras och beskriva vilken information som går förlorad och vilken information som behöver kompletteras vid en transformering. Efter en utförd transformering skall den nya modellen ha den högsta möjliga detaljnivån och efter komplettering av den information som saknas, skall den mer formella semi-formella modellen vara komplett.

1.1 Frågeställning

Målet för detta arbete är alltså att ta fram riktlinjer för en transformation av en modell uttryckt i ett mer informellt semi-formellt modelleringsspråk till en modell uttryckt i ett mer formellt semi-formellt modelleringsspråk. Efter detta kommer den information som saknas för den formella modellen konkretiseras och kompletteras, detta för att de artefakter som skapas enligt de riktlinjer som tagits fram i det mer formella modelleringsspråket ska bli kompletta. Sammanfattningsvis är min frågeställning:

Vilka riktlinjer kan formuleras för att med minsta möjliga kompletteringsbehov transformera konceptuella modeller av typen EKD till modeller i UML?

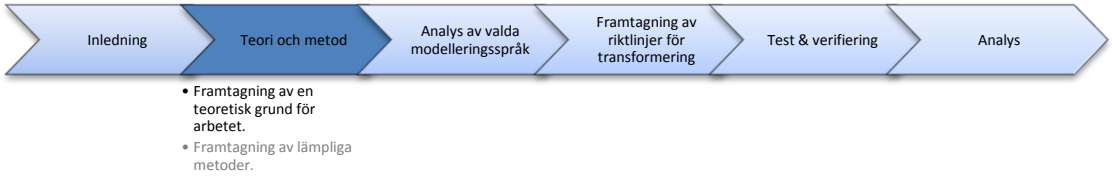
1.2 Förväntat resultat

Det förväntade resultatet bygger på en föreställning jag hade innan jag valde att utföra detta arbete. Min föreställning var att det skulle vara relativt enkelt att konvertera från EKD till UML och få nästintill helt kompletta modeller. Efter att ha gjort en del dokument- och litteraturstudier förstod jag att jag misstagit mig när det gäller hur enkelt det var och att den information som specificeras i de olika modelltyperna är till olika ändamål. Därför kommer modellerna i EKD och UML att innehålla till stor del olika information. Jag förväntar mig att kunna ta fram tillräckligt mycket information för att kunna utföra mappningen mellan modelleringsspråken och att hitta entiteter i de båda som har liknande eller samma innebörd och kan mappas direkt mellan modellerna. När de mer informella semi-formella modellerna transformeras till den mer formella semi-formella modellen kommer den formella semi-formella modellen att innehålla olika grader av syntaktisk kvalité. Jag förväntar mig att kunna

konkretisera vilken information som inte kommer att kunna transformeras och kunna konkretisera vilken information i den mer formella modellen som måste kompletteras efter en transformation för att den skall bli komplett.

2 Teoretisk grund

I detta kapitel tas en teoretisk grund fram för att belysa problemets relevans och ge en förståelse för dess domän. Denna teoretiska grund visar även att problemet har en förankring inom forskning som sker på området idag.



Figur 2:1 Arbetsprocess - Teoretisk grund

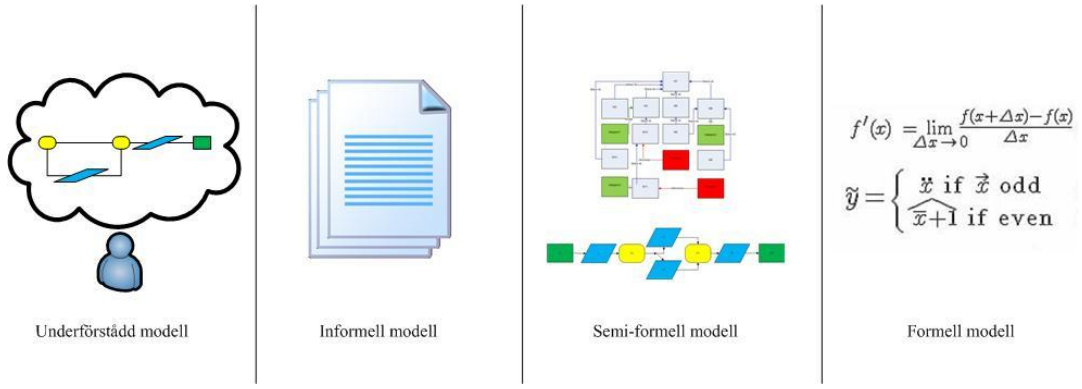
För att bemästra den komplexitet som finns i organisationer går det att använda olika hjälpmedel, exempelvis en grafisk modell. Fördelen med en sådan modell är att den ger möjligheten att beskriva ett komplext fenomen i en överskådlig modell. Denna typ av modeller kallas för konceptuella modeller.

2.1 Konceptuell modell

Brown (2004) förklarar begreppet modell som en avbild av verkligheten. En konceptuell modell är alltså en modell av en idé eller en tänkt lösning på ett problem. En konceptuell modell ger många fördelar menar Bubenko, et al. (2001). De tar bland annat upp att konceptuella modeller gör det lättare att förstå verkliga relationer och strukturer som kan vara komplexa, förutse vilken kvalité något kommer att få, få en dokumentation på hur en uppgift skall utföras och underlättar i diskussioner som går in på en specifik detalj inom det område som modelleras. Bubenko, et al. (2001) hävdar att med hjälp av konceptuella modeller kan en organisation få hjälp att finna lösningar på problem, att strukturera verksamheten, få en översikt, behålla informationen i organisationen och få stöd för att bygga IT-system. I denna rapport tas endast konceptuella modeller upp om inget annat anges.

Modellens detaljrikedom mäts i hur hög formalitet den har. Desto högre formalitet en modell har desto mer detaljerad och entydig blir den. Att ha en högre formalitet sker dock på bekostnad av att modellen blir mindre flexibel och förlorar sin skalbarhet. För att en modell skall uppnå en högre formalitet behöver modelleraren applicera ett regelverk för hur modellen skall ritas. Martín-Vivaldi och Isacsson (1988) tar upp olika grader på formalitet där de börjar med vad de kallar en ”underförstådd modell”. Denna modell är odokumenterad och finns i hjärnan på dem som har kunskapen om det område som ska modelleras. Nästa nivå på modeller är vad de kallar en ”informell modell”. Denna modell är en textbaserad modell, där kunskapen finns dokumenterad. Det är inte säkert att modellen har en fördefinierad form vilket kan leda till att viss information lätt kan missas. Detta gör att en informell modell kan skilja mycket beroende på om den som dokumenterat modellen fått med all nödvändig information. Nästa nivå på formalitet enligt Martín-Vivaldi och Isacsson (1988) är ”semi-formella modeller”. Dessa modeller har en fördefinierad struktur, checklistor och är ofta

grafiska för att uppmärksamma användaren på att få med all nödvändig information. Eftersom dessa modeller dokumenteras efter ett regelverk är kvalitén på modellerna mätbar. Dessa modeller har fortfarande god flexibilitet och ger betraktaren en lättförstådd, överskådlig bild av den uppritade verkligheten. Den sista nivån av formalitet enligt Martín-Vivaldi och Isacsson (1988) är ”formella modeller”. Detta är modeller som är detaljerade och entydiga, inga otydligheter får finnas i dessa. Det finns goda förutsättningar för att implementera programmeringskod direkt utifrån denna typ av modeller eftersom de ofta matematiskt baserade.



Figur 2:2 Formalitetsnivåer.

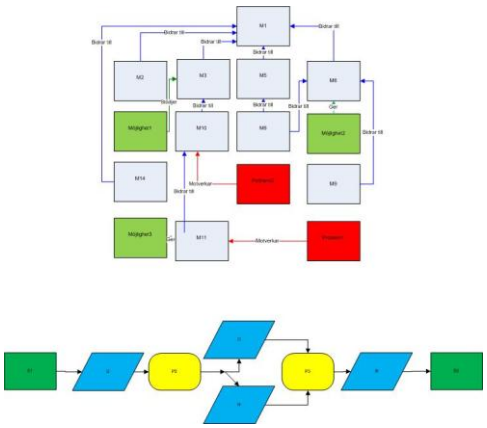
Martín-Vivaldi och Isacsson (1988) förklarar att de olika modelltyperna passar vid olika tillfällen. Exempelvis kan en ny organisation som inte ännu vet hur alla uppgifter skall utföras hämmas av att använda sig av en alltför formell modell. Det vill säga att ingen formalitet på modeller är bättre än någon annan utan att val av modell beror på situation och kontext.

2.2 Modelleringsspråk

För att definiera ett modelleringsspråk måste syntaxen beskrivas skriver He, et al. (2007). Syntaxen består av en abstrakt och en konkret syntax. Den abstrakta syntaxen, även kallad metamodell, beskriver strukturen på modelleringsspråket, som relationer och restriktioner mellan entiteter som ska modelleras. Den konkreta syntaxen är modellens utseende när språket används, entiteter och relationers utseende. Även Booch, et al. (2008) tar upp ett modelleringsspråks uppbyggnad i sin bok. De definierar de två byggstenarna i ett modelleringsspråk som notation och metamodell. Dessa begrepp har samma innebörd som definitionen i He, et. al (2007). Ett modelleringsspråk kan vara textbaserat, grafiskt eller en kombination av dessa.

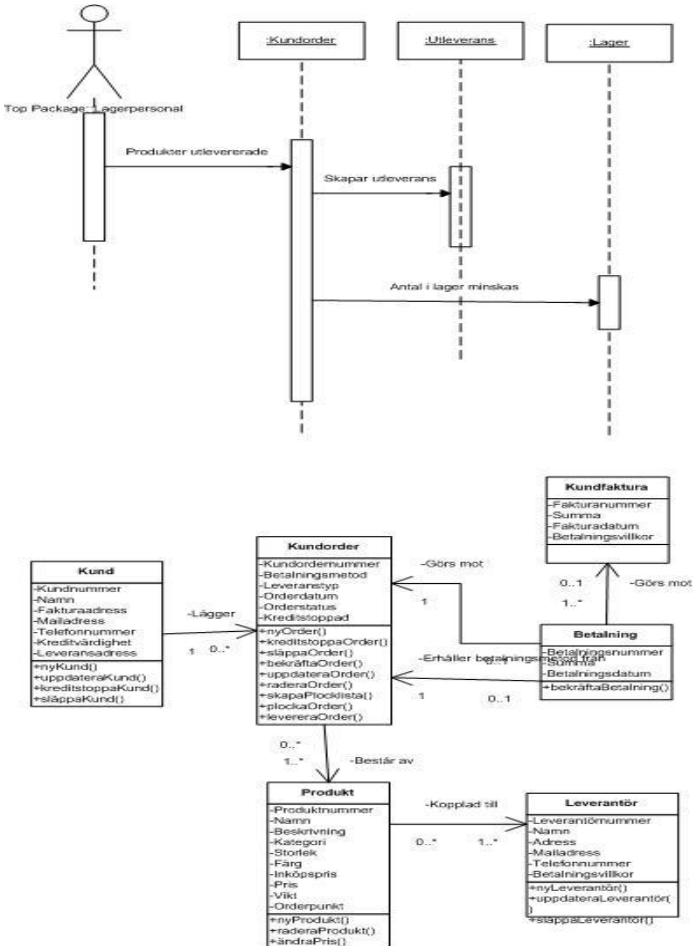
Detta examensarbete kommer att behandla semi-formella modeller. De modeller som kommer att tas upp är relativt informella semi-formella modeller, som skall transformeras till mer formella men fortfarande semi-formella modeller. De modelleringsspråk som kommer att analyseras i denna rapport har en grafisk notation med olika stor formalitet.

Det relativt informella semi-formella modelleringsspråk som kommer att tas upp i denna rapport är EKD (Bubenko, et al., 2001). Detta modelleringsspråk har regler för hur modellen skall byggas upp. Modellen har även en egen grafisk notation och exempel på dessa finns i Figur 2:3. Att EKD återfinns på den mer informella sidan för med sig att språket passar för en verksamhetsanalys genom att deltagarna inte hämmas i arbetet på grund av en för hög formalitet. Bubenko, et al. (2001) skriver att modelleringsspråket även passar bra att användas inför införande av ett IT-system.



Figur 2:3 Exempel på verksamhetsmodeller i EKD

I detta arbete kommer UML vara det modelleringsspråket som de mindre formella semi-formella modellerna kommer att transformeras till. Även UML är ett semi-formellt modelleringsspråk men det är mer formellt semi-formellt. Booch, et al. (2008) tar upp att det huvudsakliga syfte med UML är att fungera som underlag vid diskussioner angående specifika aspekter av ett system. Om hela system är uppbyggda från modeller i UML kan utvecklarna använda UML som ett eget programmeringsspråk och kan översätta modellerna direkt till programmeringskod. Exempel på modeller som är vanliga i UML är sekvensdiagram och klassdiagram som visas i Figur 2:4 Vissa modeller i UML, om de används på ett korrekt sätt, det vill säga uppnår en syntaktisk kvalité som är hög nog, kan med hjälp av olika verktyg som finns på marknaden översättas direkt till programmeringskod. Vilken syntaktisk kvalité som krävs bestämmer verktyget som skall användas. Booch, et al. (2008) skriver dock att det är svårt att förstå vilken programmeringskod som modellen kommer att generera genom att bara se på modellen men det kan däremot ge en förståelse för hur programmeringskoden kommer att se ut.



Figur 2:4 Sekvens- och klassdiagram är två exempel vanliga diagram i UML

2.3 Användning av modeller

Det finns många användningsområden när det gäller modeller. Under förgående punkt påvisades hur ett budskap kunde kommuniceras med hjälp av en modell och hur den kunde användas för olika ändamål, exempelvis att finna lösningar på problem. Det togs även upp att en modell kan användas i dokumenteringssyfte och i detaljdiskussioner för olika saker, exempelvis ett IT-systems funktionalitet eller hur något utförs i en verksamhet. Dessa sista punkter kommer ofta in när en organisation har bestämt sig för att införskaffa ett IT-system. Det är inte bara nya IT-system som behöver modelleras och byggas, Brown (2004) tar upp problemet med att många organisationer idag har gamla IT-system som har hängt med över åren och har olika nivå av kvalité. Dokumentationen för dessa finns ofta inte eller är bristfällig. För att skapa eller införskaffa ett IT-system åt en organisation behöver behov, hur processer fungerar, vilka begrepp som används, vilka aktörer och intressenter som finns bland annat kartläggas och gärna dokumenteras i form av modeller. Dessa modeller blir tillsammans en verksamhetsanalys och denna är grunden till att få IT-system anpassade till verksamheten. He, et. al (2007) menar att modeller nyligen har blivit det centrala i mjukvaruutveckling. Detta märks även av att uttrycket Model Driven Architecture (MDA) blir mer och mer förekommande inom detta område när det diskuteras användning av modeller.

Brown (2004) förklarar att modeller och modelldriven utveckling av mjukvaror är grundstenen i MDA. Model Driven Architecture (MDA) är ytterligheten inom systemutveckling när det gäller användning av modeller. Vanligtvis används modeller som ett komplement till en systemutvecklingsprocess. Inom MDA är det modellerna som driver utvecklingsprocessen framåt. MDA är ett fenomen som växer enligt Brown (2004). Brown (2004) tar även upp att Object Management Group (OMG)[2] har förändrat sitt angreppssätt och arbetar för att använda MDA i utvecklingsskeden då de anser att det är ett bättre sätt att möta kundens behov. OMG anser även att MDA ger större flexibilitet vid systemförvaltning.

Att bygga ett IT-system utifrån modeller kräver att den modell som skall användas har högsta möjliga formalitet (Martín-Vivaldi och Isacsson, 1988). De förklarar att ju fler som finns med i uppbyggnaden av systemet desto högre formalitet krävs, detta för att de som skall utveckla systemet skall ha samma bild av hur systemet skall vara uppbyggt. Martín-Vivaldi och Isacsson (1988) tar upp problemet med att gå från en underförstådd eller informell modell direkt till en formell modell. De menar att glappet är för stort och att det är värt mödan att gå via en semi-formell modell. I och med att modelleringsspråket har ett uppsatt regelverk för de semi-formella modellerna menar Martín-Vivaldi och Isacsson (1988) att modeller från detta språk får en förutbestämd konstruktion. Detta resulterar i att det blir lättare att definiera regler då den semi-formella modellen blir mer specificerad och planerad än en modell från ett modelleringsspråk med mindre formalitet. De tar även upp fördelen att det går att se om modellen inte är valid i ett tidigare skede och om det finns element i de artefakter som skapats som inte har någon koppling. Det finns även andra fördelar att gå via en semi-formell modell till en formell modell menar Bubenko, et al. (2002). De tar upp att det finns ett behov av att göra en verksamhetsanalys innan ett IT-system byggs för att få kopplingen till organisationens behov genom infångade funktionella och icke funktionella krav. När det gäller existerande IT-system och modeller hos en organisation tar de upp att kraven på förändringar i dessa IT-system idag är höga och att dessa förändringar skall ske snabbt. Ett sätt att minska tiden för förändringen kan vara att återanvända modeller med redan existerande funktionella krav.

Även om alla dessa fördelar med att använda sig av modeller finns skriver Brown (2004) att en majoritet av utvecklarna ändå ofta implementerar systemet direkt utan att använda sig av modeller. Den information som finns om projektet har de på temporära medier, exempelvis whiteboards, PowerPoints eller i utvecklarnas hjärnor. Detta leder till problem i utvecklingen, problem som exempelvis svårigheter att få en överblick av hur systemen hänger ihop och vem som utvecklar vilken del av systemen och komplexiteten ökar. Det blir ännu större problem när något skall förändras i systemen och utvecklaren som implementerade sin del av systemet inte finns tillgänglig längre för organisationen. Brown (2004) tar upp att en majoritet inte använder sig av modeller men skriver även att inom utveckling av programvara är det vanligt att använda sig av modeller och att detta började redan när utveckling av IT-system var nytt. I och med detta går det att säga att det finns vissa som använder sig av modeller och andra gör det inte men även om inte alla använder modeller för att utveckla så är det ett välkänt fenomen att göra det.

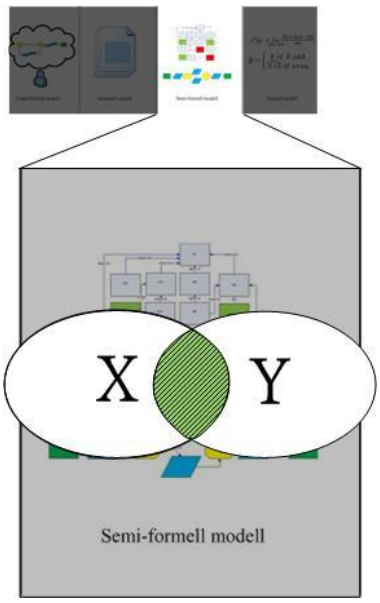
[2] http://www.omg.org

2.4 Transformation till en mer formaliserad modell

För att bygga ett IT-system utifrån modeller måste alltså modellen vara av typen formell modell enligt det resonemang som tidigare förts. För att komma till den formella modellen bör en modellerare gå igenom en semi-formell modell för att underlätta och hitta brister i ett tidigt stadium. Brown (2004) tar upp att de semi-formella modellerna måste vara korrekta. Om modellerna inte är korrekta kan detta leda till att det är omöjligt att transformera modellen till en modell med högre formalitet. Krogstie, et al. (2006) tar upp begreppet modellkvalité, och vilka aspekter det finns på detta. De tar bland annat upp syntaktisk kvalité, semantisk och pragmatisk kvalité. Dessa tre faktorer är viktiga på olika sätt vid en transformation. Att modellen är korrekt, som Brown (2004) tar upp, betyder att modellen är syntaktiskt korrekt, det vill säga att modellen följer den notation som det valda modelleringsspråket har. Med korrekt menas även att modellen är semantiskt korrekt vilket innebär att modellen är sammanhängande och alla funktioner har mappats ut korrekt i förhållande till den verklighet som beskrivs. Den sistnämnda kvalitén är pragmatisk kvalité. Pragmatisk kvalité är utbytet mellan modell och aktörernas tolkning av modellen. Målet med pragmatisk kvalité är förståelse. Om aktörerna har förstått modellen och kan interagera med den har denna kvalité uppfyllts. Som rapporten tidigare tagit upp krävs att modellen har en viss formalitet, men inte för mycket, för att de flesta verksamhetsrepresentanter skall förstå den. Blir modellen för formell krävs att aktörerna har en hög förståelse för mer formella modelleringsspråk. När en verksamhet skall modelleras behövs kompetenser från alla de olika områdena inom verksamheten som berörs (Bubenko, et al. 2001). För att uppnå en hög pragmatisk kvalité när en verksamhet modelleras krävs att alla inblandade förstår modellen och därför krävs det att en modell med något lägre formalitet används, som en semi-formell modell.

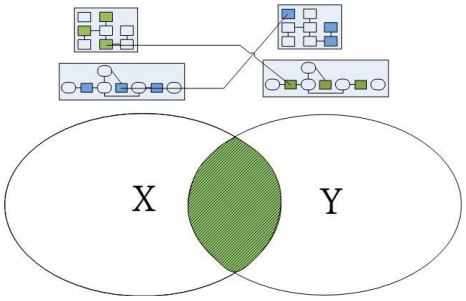
Det finns en flera studier gjorda på transformation mellan modeller. Brown (2004) hävdar att för att göra transformationer mellan modeller behövde modellerna själva formaliseras. Genom att modellera dessa till en metamodell kan modellernas entiteter mappas mot varandra mellan modeller. Object Management Group (OMG) har sett fördelen med detta och tagit fram ett standardspråk för att skapa metamodeller, Meta Object Family (MOF), som är baserad på Unified Modeling Language (UML). Även Bubenko, et al. (2002) beskriver hur de handskas med problemet att transformera en modell till en annan och även de tar upp fördelarna med användning av en metamodell.

Inom en viss grad av formalitet kan modellerna vara mer eller mindre formella. Detta arbete kommer att röra sig inom det semi-formella området, som visas i Figur 2:5. Även inom detta område finns olika modelleringsspråk som har mer eller mindre formalitet. Som arbetet tagit upp tidigare har modeller valts som sträcker sig från den mer informella delen av semi-formella modeller till den mer formella delen av semi-formella modeller. Detta, detta illustreras i Figur 2:5.



Figur 2:5 Modelleringsspråk X och Y inom området för semi-formella modeller

De språk som kommer undersökas har en spännvidd från informella semi-formella modeller till formella semi-formella modeller. Modellerna kommer även att ha viss överlappande information vilket representeras av det markerade området mellan X och Y i Figur 2:5. Modelleringsspråkens metamodeller talar om hur information kan specificeras i modelleringsspråket, vilken betydelse informationen har och hur den är sammanlänkad med andra entiteter och modeller.

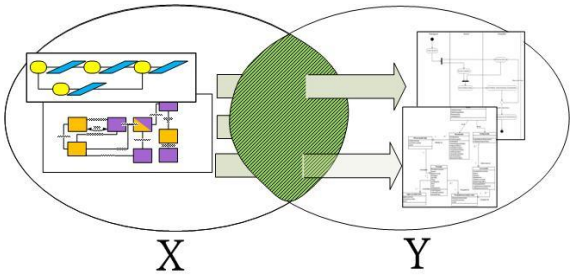


Figur 2:6 Matchning mellan metamodeller i modelleringsspråken X och Y.

Metamodellerna i det mindre formella språket kan alltså vara en bra grund för att kunna få fram vilken information som kan transformeras till samma eller liknande entiteter i det mer formella modelleringsspråket. I Figur 2:6 illustreras metamodellerna som en ovanliggande struktur för modelleringsspråken. Dessa har

kopplingar som visar på att vissa entiteter i modellerna motsvarar varandra även om de inte är uttryckta på samma sätt. Viss information inte kan mappas i figuren, dvs. det som inte är i det överlappande området av modelleringsspråk X och Y.

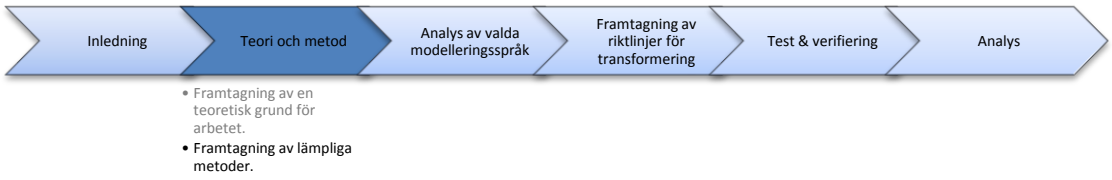
Figur 2:7 visar på hur viss information kan transformeras direkt över från modelleringsspråk X till modelleringsspråk Y. Figuren visar även hur information inte kommer fram alls och hur annan information delvis kommer fram. Att veta vilken information som kan överföras och vilken som inte kan överföras är viktigt. Den information som inte överförs kan vara av stor vikt för att systemet skall bli framgångsrikt vid en implementation. Det är även av vikt för modelleraren att veta vilken information som kan överföras. Detta för att kunna prioritera den information som krävs för att utföra transformeringen. Genom att kunna prioritera rätt information kan modelleraren minska behovet av komplettering när modellen nått en högre formalitet. Detta examensarbete kommer att undersöka denna punkt, vilken information som inte kan överföras och går förlorad om den inte tas om hand, vilken information som kan överföras och vilken information som behöver kompletteras för att åstadkomma en komplett modell.



Figur 2:7 Transformering av modelleringsspråk X till Y. Endast viss data överlappar mellan språken.

3 Metod och metodval

Detta arbete grundar sig på kända och beprövade metoder för att samla in fakta och göra studier på. Arbetet med att ta fram och redogöra för varför dessa metoder valts görs i början av processen som modellen under visar.



Figur 3:1 Arbetsprocess vid framtagning av metoder

Detta kapitel skrivs när arbetsprocessen befinner sig i inledningsfasen av arbetet. För att välja vilken metod som är lämplig att använda gås metodansatser först igenom och utifrån dessa väljs vilka metoder som passar bäst för detta arbete.

3.1 Kort om metodansats

Detta arbete är tänkt att utföras genom att studera de olika modelleringsspråken. Genom att studera, manualer, böcker och litteratur kommer att information samlas in för att utföra konstruktionen av transformeringsriktlinjerna. För att utföra informationsinsamlandet kommer dokument- och litteraturstudier utföras. dokument- och litteraturstudier kan anses vara en kvalitativ ansats. Berndtsson, et al. (2002) menar att syftet med kvalitativa metoder är att öka förståelsen istället för att hitta en förklaring. De förklarar att över åren har det utvecklats många olika ansatser för att utföra kvalitativ undersökning. En forskare som utövar denna teknik sätter ofta sig själv i nära relation till det som skall undersökas.

För att ta fram de riktlinjer som arbetet tar upp i det tidigare kapitlet måste det finnas material som ligger till grund för dessa. När materialet är framtaget måste riktlinjerna designas och konstrueras. För att utföra detta arbete en ansats som kallas för designansats att användas. Hevner, et al. (2004) förklarar att designansats är ett paradigm för problemlösning som har sitt ursprung i ingenjörstekniker. I denna ansats används kunskap och förståelse för problemdomänen för att konstruera/designa något och sedan utvärdera konstruktionen. Resultatet av en sådan approach tar fram ett ”best practise” och bidrar till en kunskapsbas som går att återanvända. I detta arbete är tanken att utifrån dokument- och litteraturstudierna skapa ett ”best practise”, riktlinjer, för hur transformation kan ske mellan de olika modelleringsspråken. Målet är att dessa riktlinjer skall vara övergripande nog för att kunna återanvändas i olika situationer. Hevner, et al. (2004) tar upp att designansats är både en process och den ger artefakter eller produkter. Denna rapport har som mål att ta fram nya artefakter i ett nytt modelleringsspråk och på så sätt passar denna ansats bra för att nå rapportens förväntade resultat. Det finns ett antal riktlinjer som skall följas för att ansatsen på det arbete som utförts skall klassas som en designansats enligt Hevner, et al. (2004). I

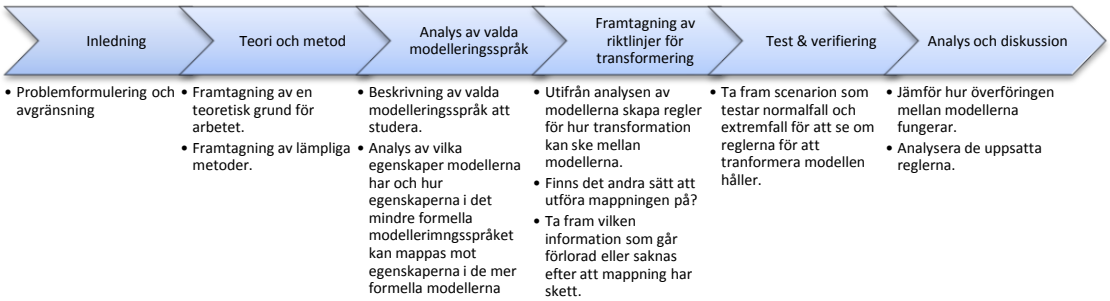
Tabell 3:1 finns dessa riktlinjer uppsatta tillsammans med hur detta arbete är tänkt att uppfylla dess mål.

Designansatsens riktlinje	Hur detta arbete är tänkt att uppfylla riktlinjen
En designansats måste skapa någon form av artefakt.	Detta arbete har som mål att skapa riktlinjer för transformation mellan två modelleringsspråk för att producera artefakter i ett nytt modelleringsspråk och kommer att uppfylla riktlinjen för att artefakt kommer att skapas.
Målet med arbetet i designansatsen är relevant för problemet.	Inledning och bakgrund för detta arbete påvisar att det finns en grund för varför detta arbete utförs. Det är genom att konstruera något, i detta fall riktlinjer, som den problemformulering som detta arbete har kan besvaras.
Artefakten, kvalitén och effektiviteten skall vara noga demonstrerade och validerade med kända metoder.	Artefakten skall demonstreras och valideras i detta arbete. Jag kommer även använda mig av dokument och litteraturstudier för att samla information för att skapa transformationsriktlinjerna. Utöver detta är arbetet ett examensarbete där det finns det krav på att resultatet skall demonstreras i form av en presentation och offentligöras. Det finns även krav på användandet av vetenskapliga metoder för att samla in och arbeta med information.
Den skall tillhandahålla klara och verifierbara bidrag inom sitt område.	Omfattningen av vad detta arbete kan bidra med inom detta område kan jag inte avgöra. Men som exempel kan detta arbete påvisa att viss överlappning finns mellan de utvalda modelleringsspråken. Resultatet av detta arbete kan göra det lättare att veta vilka delar som kan transformeras.
Arbetet, både tillämpning och utvärdering, i en designansats skall vila på beprövade metoder.	Arbetet är grundat på vetenskapliga metoder. Tillämpningen kommer att testas i steget för test och verifiering för att sedan utvärderas i diskussionen. Arbetet är tänkt att skapa ett så kallat ”best practise” för hur transformeringen kommer att gå till.
För att nå önskat resultat i sökandet efter nya effektiva artefakter krävs att tillgängliga medel brukas utan att bryta de regler som finns i problemets domän.	Arbetet kommer att använda sig av olika medel så som rapporter, litteratur och vetenskapliga metoder för att samla in information. Detta för att nå målet, att skapa riktlinjer för en överföring av modeller. Regler för problemets domän, så som modelleringsspråkens semantik och syntax, är de viktigaste reglerna som inte får brytas.
Artefakten från arbetet gjort på designansatsen skall presenteras för både ledning och tekniskt kunniga intressenter	De riktlinjer som detta arbete skapar kommer inte att presenteras för någon utanför högskolan, inte i det första skedet i alla fall utan kommer att presenteras för personer som är mer eller mindre insatta inom detta område i form av studenter och lärare på utbildningen.

Tabell 3:1 Riktlinjer för designansats och hur detta arbete är tänkt följa dem

3.2 Arbetsprocess

Arbetet är planerat att utföras enligt Figur 3:2. Även om denna modell ser ut att vara sekventiell går stegen in i varandra och visst arbete sker ändå parallellt. I arbetsprocessen kommer det även finnas med hur metoderna kommer att användas för att komma fram till ett svar på min frågeställning.



Figur 3:2 Arbetsprocessmodell

Modellens tidsflöde läses från vänster till höger. Varje steg i modellen har en överskrift och en djupare förklaring under vad som skall åstadkommas i just det steget. Denna modell kommer att förekomma på olika ställen i rapporten för att visa var i processen arbetet befinner sig. Som Figur 3:2 visar består detta arbete till största delen av en analys av modelleringsspråk och hur dessa kan mappas motvarandra.

3.2.1 Inledning

Under inledningen kommer problemformuleringen att presenteras som detta arbete ämnar besvara. Problemformuleringen skall presenteras ihop med en bakgrund för att ge en lite djupare förståelse till problemets relevans. Problemet skall även vara väl förankrat i den forskning som bedrivs idag inom området. Detta moment genomförs genom att presentera fakta och forskning inom området som tas fram med hjälp av dokument och litteraturstudier.

3.2.2 Teori och metod

Under denna del av arbetsprocessen kommer dokument- och litteraturstudier utföras för att få en bra grund till området. Tidigare forskning som gjorts kommer att användas för att få fram nya och viktiga begrepp som finns inom detta område. Att få information om vilka problem som kan uppstå kommer att hjälpa mig med utförandet av analysen och upprättandet av riktlinjerna då den som använder dem kan slippa göra misstag som andra redan gjort. Dokument- och litteraturstudier är av stor vikt för att få en god grund att stå på när arbetet utförs. I detta stycke kommer Metoder för att undersöka problemet kommer att tas fram och presenteras. Metoderna skall vara genomtänkta och relevanta för att kunna ta fram en lösning på mitt uppsatta problem. Detta sker genom att förklara hur dessa metoder skall användas.

3.2.3 Analys av valda modelleringsspråk

I detta steg kommer de utvalda modelleringsspråken att analyseras. Detta kommer att ske genom att göra dokument- och litteraturstudier på manualer, böcker och andra informationskällor för modelleringsspråken. Från dokument- och litteraturstudien kommer metamodeller tas fram eller konstrueras för att sedan undersökas. Målet är att få fram entiteter och relationer som har likheter mellan modelleringsspråken av någon

grad. Desto mer lika entiteterna och relationerna är desto troligare är det med en direktöversättning mellan språken. Modellernas egenskaper och information kommer att modelleras upp för att få en metamodell. Med hjälp av metamodellen kommer det gå att avgöra vilken av modelleringsspråkets information som kommer att kunna representeras i det andra modelleringsspråket. För att utföra detta kommer teorier framtagna i de förgående avsnitten i detta examensarbete användas.

3.2.4 Framtagning av riktlinjer för transformering

I detta steg kommer riktlinjer designas och konstrueras för att transformera mellan de valda modelleringsspråken. Det förgående steget, Analys av valda modelleringsspråk, kommer att ligga till grund för utförandet av detta steg. Riktlinjerna kommer att baseras på metamodellerna, analysen av de tidigare gjorda studierna på transformationer mellan modeller samt de dokument och litteratur studier som gjorts. Konstruktionen kommer att ske utefter designansatsen och språkens syntax och semantik kommer att studeras för att inte bryta dessa. Efter att transformering är utförd kommer andra teorier framtagna av dokument- och litteraturstudien testas för att utföra mappningen mellan de olika modelltyperna. I detta steg kommer sedan den informationen som förloras i transformationen och den information som saknas för att den nya modellen skall vara syntaktiskt korrekt konkretiseras. Denna information kommer att undersökas närmare för att se om det finns andra modeller som kan ta tillvara på denna information för att undvika att den går förlorad.

3.2.5 Test och verifiering

I detta steg kommer de framtagna riktlinjerna testas på ett fall som finns beskrivet i en metodhandbok (Bubenko, et al., 2001) för ett av de valda modelleringsspråken. Fallet innehåller ett antal modeller, vilka ska transformeras till ett annat modelleringsspråk. Detta för att se om reglerna fungerar i verkligheten och inte bara i teorin. Syftet är även att undersöka om den modell som transformerats till får en korrekt syntax.

3.2.6 Analys

Analysen kommer att bestå av en analys av hur överföringen mellan modellerna fungerade. Graden av modellernas semantiska kvalité och hur den påverkar resultatet kommer även att tas upp här. I analysen kommer de uppsatta riktlinjerna för transformationen att analyseras utifrån den informationen som insamlats från dokument- och litteraturstudien men även resultaten från arbetet i Analys av valda modeller och Test och verifiering.

I analys kommer även frågan om transformeringen kunde lösas på annat sätt och förslag till förbättringar att diskuteras. Dessa punkter kommer att finnas under Analys och diskussion.

3.3 Kort om valda sätt att samla information

Denna undersökning är tänkt att utföras med hjälp av en designansats och även ta hjälp av några kvalitativa metoder. De som passar bäst är dokument- och litteraturstudier för att få information om modellerna genom att läsa manualer, rapporter och annan litteratur om ämnet och om någon försökt att utföra transformeringar mellan olika modeller tidigare och därigenom ta reda på vilka fallgropar de funnit.

3.3.1 Dokument- och litteraturstudier

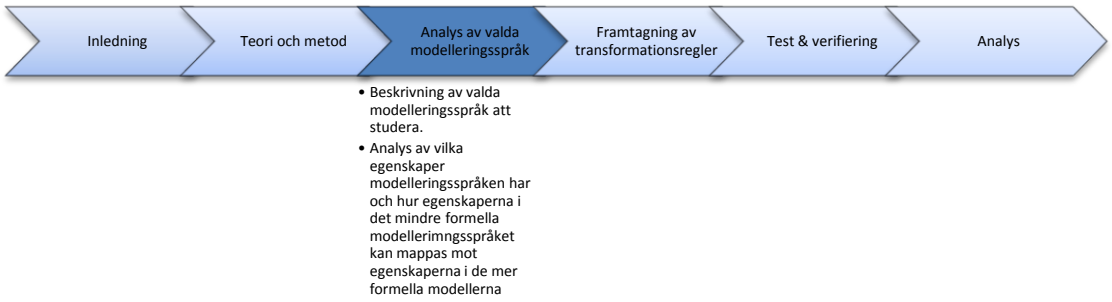
Ejvegård (2009) tar upp dokumentstudier i sin bok. Han menar på att det är av stor vikt att använda sig av bra och relevanta sökord för att få relevanta träffar i databaser. Olika databaser har använts för att söka informationen, exempelvis Google Scholar, Springerlink, ACN, ELIN och IEEE. Jag har använt sökord för att studera de olika nivåerna av formalitet som ”informal to formal models” och orden ”transform”, ”formal” och ”model” i olika kombinationer. Själva transformationerna har jag bland annat använt ”metamodel”, ”model to model”, ”model driven development”, ”MDA” och ”fuzzy to formal”. Dessutom har de arbeten som handledaren utarbetat i sin forskning inom området använts. Dokument- och litteraturstudier för att få reda på vilken information de informella och de formella modellerna innehåller kommer att utföras och försök eller teorier om hur transformering kan utföras kommer att studeras. Litteratur som tar upp problematik med transformering kommer även att studeras med intresse för att kunna utföra detta arbete på ett så bra sätt som möjligt. Utefter den information som hittas om modellernas innehåll och teorierna som finns för transformering kommer riktlinjer försöka tas fram som kan användas för att transformera de utvalda modellerna.

3.3.2 Validering

För att uppnå en hög kvalité på riktlinjerna för transformationen kommer validering av riktlinjerna som arbetet tar upp i steget Test och verifiering i min Arbetsprocessmodell att utföras. Bubenko, et al. (2001) skriver om validering i deras manual och att detta är en viktig del för att uppnå en hög kvalité på en modell. Lindland, Sindre och Solvberg (1994) tar upp vikten av validering för att uppnå hög kvalité för att se om modellen kan tillfredställa de mål som har satts upp. Validering är även ett viktigt moment i designansatsen som det går att se i Tabell 3:1.

4 Analys av valda modelleringsspråk

I detta kapitel, som är en del av empirin i denna rapport, kommer valda modelleringsspråk analyseras och hur innehållet i dessa kan länkas mellan varandra beskrivas. Detta steg i arbetet visualiseras i Figur 4:1.

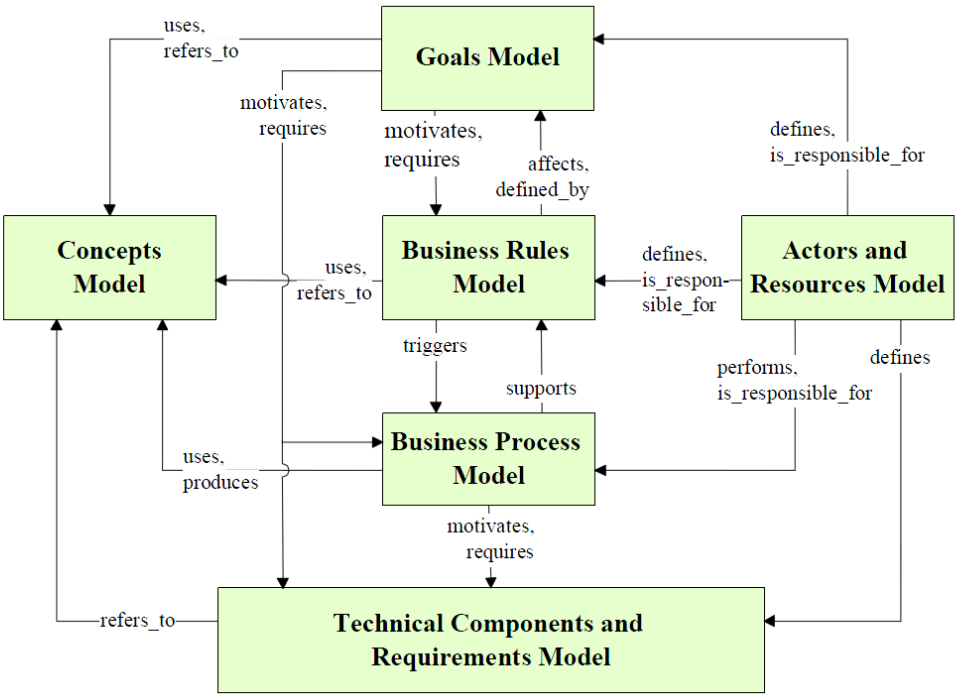


Figur 4:1 Arbetsprocess under analysen av modelleringsspråken

Tanken med att välja EKD och UML är att kunna arbeta med utveckling av ett IT-system från början till slut inom ramen för semi-formella modeller. Modelleringsspråken EKD och UML valdes för att de tar oss genom hela steget för semi-formella modeller. Under denna punkt kommer arbetet först att gå igenom EKD och ta fram metamodeller för detta modelleringsspråk. Efter det kommer UML gås igenom för att se vilka delmodeller inom UML som är lämpliga att transformera till för att sedan välja modeller som passar för denna transformation.

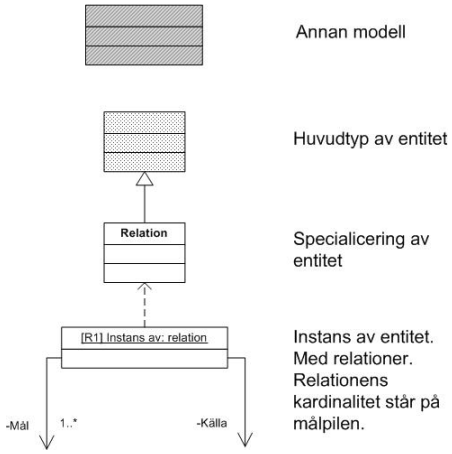
4.1 EKD

EKD har ett sex modelltyper för att representera olika aspekter av en verksamhet. De modeller som EKD tar upp i sin manual är bland annat målmodellen, aktörs- och resursmodellen och processmodellen. Figur 4:2 är tagen från Bubenko, et al. (2001) och visar på hur modellerna hänger hop i detta modelleringsspråk. Jag kommer att skapa metamodeller för EKD med utgångspunkt från manualen för EKD (Bubenko, et al., 2001). Här tas även upp att andra modelltyper kan användas vid behov men detta examensarbete kommer inte ta upp användning av andra modelltyper.



Figur 4:2 Modelltyper i EKD[3]

I de metamodeller jag har skapat för EKD finns olika typer av entiteter. I Figur 4:3 visas vilka specialtyper av entiteter som använts i mina metamodeller. Den första, av mörkare grå färg visar en annan modell. Den används för att visa relationer mellan modelltyper. Nästa komponent, som är ljusare grå, är huvudtyp av entitet. Denna är en huvudtyp som ofta har specialiseringar av typen i modellen. Relationer i modellen symboliseras av en klass. Denna klass har instanser av klassen som symboliseras med streckade pilar. Instanserna av relationen har en källa och ett mål. Kardinaliteten mellan entiteterna sätts på målrelationen från instansen av relationen.



Figur 4:3 Entitettyper i metamodeller

[3] Modellen används med tillstånd från Bubenko, et al. (2001)

I metamodellerna är uttrycken Aggregering (PartOf), Generalisering (ISA) och Binära relationer vanliga. Med aggregering menas att en entitet är en del av en annan entitet. Generalisering är när en entitet är samma som en annan entitet men med vissa egenskaper som gör den till en specialisering av förälderklassen.

4.1.1 Målmodellen (Goals Model)

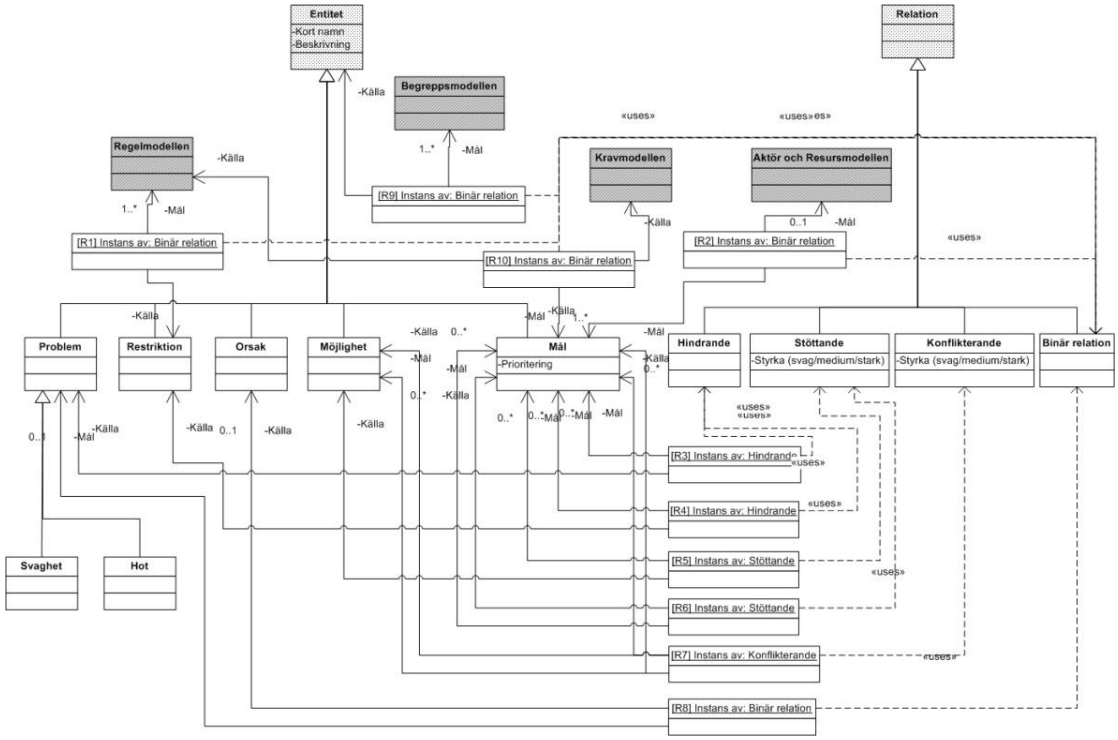
Denna modelltyp beskriver målen för organisationen samt vilka hinder och restrektioner som finns för att uppnå målen samt vilka möjligheter som målen för med sig. Mål i sig är ett önskat tillstånd som verksamheten skall uppnå. Mål skall vara både kort- och långsiktiga mål och dessa bör även prioriteras.

De entiteter som finns i en målmodell är alltså Mål, Problem, Restriktion, Orsak och Möjlighet. Alla entiteter skall ha ett unikt namn eller id. Mål skall även ha en text som förklarar målet och gärna en prioriteringsgrad. Det finns två typer av problem: hot och svagheter, vilka båda kan bidra till att ett mål inte kan uppfyllas med skillnaden att en svaghet har organisationen kunskap om hur de skulle kunna avvärja medan ett hot har inte organisationen kunskap om hur de skulle kunna motstå. En restriktion är något som förhindrar möjligheten för att ett mål skall kunna uppfyllas. Exempel på en sådan restriktion kan vara policys inom företaget, regler, externa policys eller andra faktorer. Orsak används för att förklara skälet till varför ett problem uppstår och är oftast något utanför organisationen som inte kan påverkas men är bra att ha med för att bland annat skapa förståelse. En möjlighet används för att visa om ett mål underlättar för ett annat mål att uppnås. Vissa möjligheter är sådana som organisationen kan ”få på köpet” när de uppfyller ett annat mål.

Inom målmodellen finns fyra olika relationer: binär relation, stöttande, hindrande och konflikterande. Den stöttande relationen visar på att ett mål stödjer ett annat mål. Den hindrande relationen visar på att ett mål hindrar ett annat mål att uppfyllas. Konflikterande mål är mål som inte båda kan uppfyllas. Relationerna kan användas i ”och” och ”eller villkor. Målmodellen har även relationer till andra modeller och relationer inom modellen av olika slag som är av typen binära relationer, dessa tas även upp i metamodellen för målmodellen, Figur 4:4.

I målmodellen finns alltså fem entiteter definierade, varav en har två stycken subentiteter och fyra olika relationer.

Figur 4:4 är en metamodell av EKD’s målmodell med de entiteter, relationer, instanser av relationer och kopplingar till andra modeller som har utlästs ifrån Bubenko, et al. (2001).



Figur 4:4 Metamodell av EKD's Målmodell

I modellen finns en toppentitet kallad ”Entitet”. Den har två attribut som är kortnamn och beskrivning. Det korta namnet är unikt och kan identifiera en specifik entitet. Mål är den enda av subtyperna som har ytterligare attribut utöver toppentiteten kopplat till sig, Prioritering. Denna modell har även kopplingar till Aktör och Resursmodellen, Kravmodellen och Regelmodellen. Relationerna för metamodellen i Figur 4:4 är förklarade här under i Tabell 4:1.

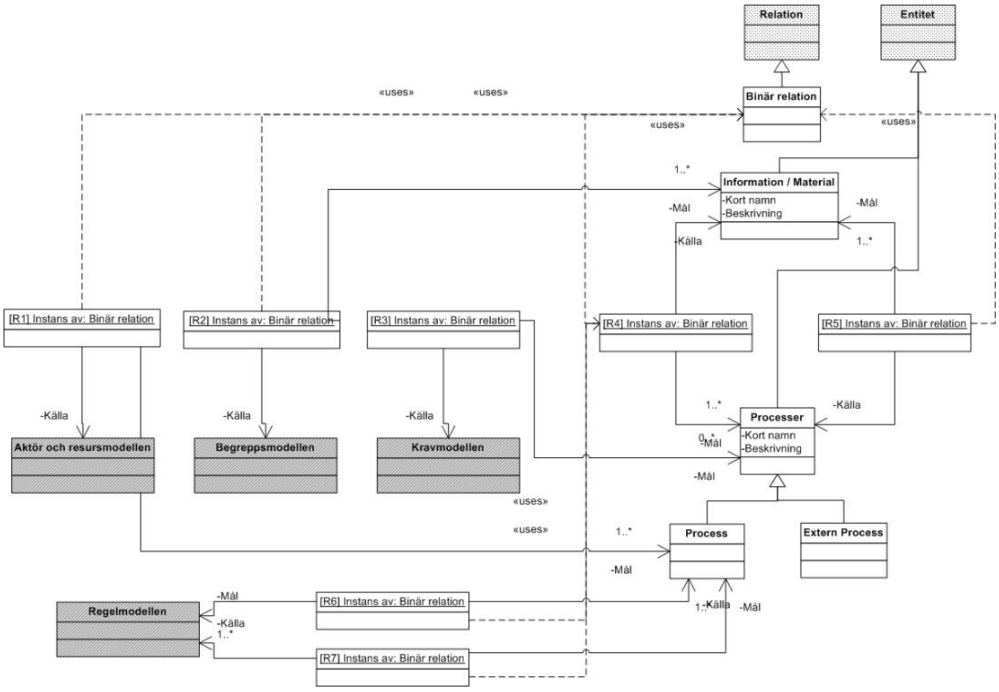
Relationsnamn	Källa	Mål	Kommentar
[R1] Instans av: Binär relation	Restriktion	Regelmodellen	Restriktion från påverkas och definieras utifrån regelmodellen.
[R2] Instans av: Binär relation	Mål	Aktör och resursmodellen	Vilken aktör och/eller resurs som ansvarar för målen och att dessa uppfylls visas med relationen mellan mål och aktör och resursmodellen.
[R3] Instans av: Hindrande	Problem	Mål	Ett problem kan hindra ett eller flera mål.
[R4] Instans av: Hindrande	Restriktion	Mål	En restriktion kan hindra ett eller flera mål.
[R5] Instans av: Stöttande	Möjlighet	Mål	En möjlighet kan stötta ett eller flera mål.
[R6] Instans av: Stöttande	Mål	Mål	Ett mål kan stötta ett eller flera mål.
[R7] Instans av: Konflikterande	Mål	Möjlighet / Mål	Ett mål kan konfliktera ett eller flera möjligheter eller mål.
[R7] Instans av: Konflikterande	Möjlighet	Möjlighet / Mål	En möjlighet kan konfliktera ett eller flera möjligheter eller mål.
[R8] Instans av: Binär relation	Orsak	Problem	Ett problem kan komma ur en orsak.
[R9] Instans av: Binär relation	Entitet	Begreppsmodellen	Entiteter använder och refererar till olika begrepp i begreppsmodellen
[R10] Instans av: Binär relation	Regelmodellen	Mål	Regler i regelmodellen påverkar hur mål sätts upp.
[R10] Instans av: Binär relation	Kravmodellen	Mål	Olika krav på IT-systemet bestäms av vilka mål som finns.

Tabell 4:1 Relationer i Målmodeller

4.1.2 Processmodellen (Business Process Model)

Denna modelltyp beskriver de olika funktioner och processer som finns i en organisation. Modellen kan ha flera nivåer och kan gå på djupet med att beskriva de olika processer, informationssamlingar och materialflödet som finns i en organisation. Dessa processer är drivna av målmodellen, det vill säga målet för en process är att uppfylla ett mål.

Processmodellen har olika entiteter. Den har två olika typer av processer, process och extern process. En process konsumerar en information eller ett material och producerar en ny värdeförändrad information eller ett material. Processen har relationer till aktör och resursmodellen för att veta vem som är ansvarig och utför processen. En extern process är ett antal aktiviteter som sker utanför den del av verksamheten som modelleras. Information/Material är en annan typ av entitet. Denna kommer från en process eller extern process och konsumeras av en process eller en extern process.
De relationer som finns är en pil som går mellan entiteterna för att visa vilket håll flödet går i, denna relation kan ha olika kardinalitet och kallas för binär relation. Dessa relationer kan användas som hopslagning för ”och” och ”eller” villkor, det vill säga för ett ”ochvillkor” skall uppfyllas och en process skall kunna köras krävs information1 och information2. Ett ”ellervillkor” gör att en process kan startas om information1 eller information2 finns tillgänglig. Figur 4:5 är en metamodell över EKD’s Processmodell utifrån den beskrivning som finns för processmodellen i Bubenko, et al. (2001).



Figur 4:5 Metamodell av EKD's Processmodell

I processmodellen finns två entiteter som behöver kunna identifieras unikt. Dessa är Information/Material och Processer. Det korta namnet i dessa fungerar som den unika identifieraren. Denna modell har även relationer med tre andra modeller, Aktör och

Resursmodellen, Begreppsmodellen och Kravmodellen. Relationerna i modellen är beskrivna i Tabell 4:2.

Relationsnamn	Källa	Mål	Kommentar
[R1] Instans av: Binär relation	Aktör och resursmodellen	Process	Vilken aktör och/eller resurs som ansvarar för målen och att dessa uppfylls visas med relationen mellan mål och aktör och resursmodellen.
[R2] Instans av: Binär relation	Begreppsmodellen	Information / Material	Informationen eller materialet definieras i begreppsmodellen.
[R3] Instans av: Binär relation	Kravmodellen	Processer	De olika krav på IT-systemet motiveras utifrån de processer som behövs.
[R4] Instans av: Binär relation	Information / Material	Processer	Informationen eller materialet konsumeras av en eller flera processer
[R5] Instans av: Binär relation	Processer	Information / Material	Processerna skapar en eller flera informationer eller material.
[R6] Instans av: Binär relation	Process	Regelmodellen	Processer stödjer regler i regelmodellen.
[R7] Instans av: Binär relation	Regelmodellen	Process	Kan starta vissa processer.

Tabell 4:2 Relationer i processmodell

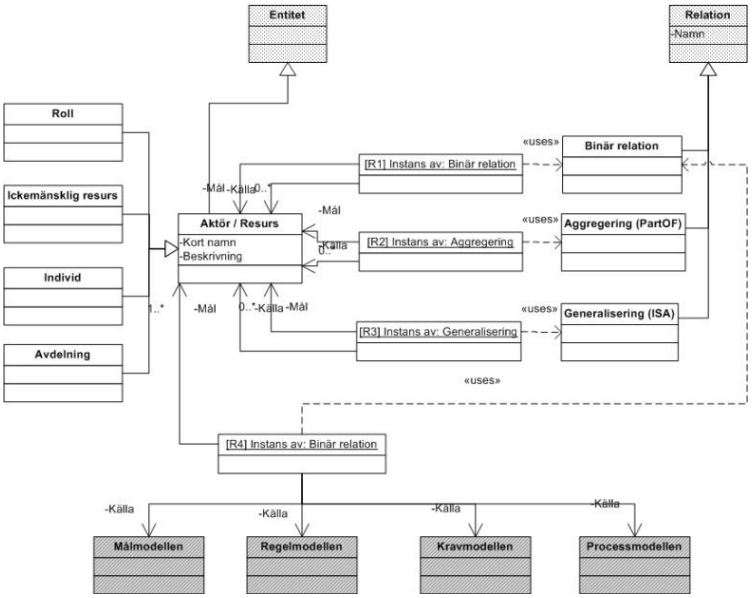
4.1.3 Aktörs- och resursmodell (Actor and Resource Model)

I denna modell definieras vilka aktörer och resurser organisationen har att tillgå.

I denna modelltyp finns en entitet som är delad i fyra delar, individ, avdelning, icke mänsklig resurs och en roll. En individ är en person som identifieras av sitt namn. En avdelning är en viss del av organisationen, en grupp som har ett visst samband. En icke mänsklig resurs kan vara en maskin eller utrustning av något slag. Dessa ickemänskliga resurser kan vara knutna till processer som utförare. Den sista delen i denna modell är roll. En roll är en resurs av något slag där den som modellerar inte vill specificera lika precist som exempelvis individ. Exempel på en roll kan vara avdelningschef och administratör. Det kan vara en individ eller en avdelning som representerar denna roll. Delarna definieras genom att vilken del entiteten representerar står med på entiteten.

Relationerna i denna modell talar om hur aktörer och resurser hänger ihop, vilken koppling de har mellan varandra. De typerna av relationer som finns inom modellen är binära, ISA och PartOF relationer.

I Figur 4:6 visas en metamodell utifrån beskrivningen av Aktör och resursmodellen i EKD från Bubenko, et al. (2001). I denna finns entiteten Aktör/Resurs, denna har en unik identifierare som är kortnamn. Modellen har en sorts entitet och tre olika sorters relationer inom modellen.



Figur 4:6 Metamodell av EKD's Aktör och Resursmodell

Denna Modell har relationer till Regelmodellen, Kravmodellen, Målmodellen och Processmodellen. Relationerna inom modellen är Binär relation, Aggregering och Generalisering. Relationerna inom modellen specificeras tydligare i Tabell 4:3.

Relationsnamn	Källa	Mål	Kommentar
[R1] Instans av: Binär relation	Aktör / Resurs	Aktör / Resurs	En relation från en aktör eller resurs till en eller flera andra av samma entitet. Relationen har ett namn.
[R2] Instans av: Aggregering (PartOf)	Aktör / Resurs	Aktör / Resurs	Relation mellan aktör eller resurs till en eller flera av samma entitet. Denna relation talar om att en entitet består av andra entiteter.
[R3] Instans av: Generalisering (ISA)	Aktör / Resurs	Aktör / Resurs	Relation mellan aktör eller resurs till en eller flera av samma entitet. Denna relation talar om att denna typ av entitet är en specialisering av en annan typ av entitet.
[R4] Instans av: Binär relation	Processmodellen	Aktör / Resurs	En relation från processmodellen som talar om vilken aktör eller resurs som ansvarar för eller utför en viss process och vem som är ansvarig för att definiera den.
[R4] Instans av: Binär relation	Regelmodellen	Aktör / Resurs	Regelmodellen har relationer till Aktör och Resursmodellen för att visa vem som är ansvarig för regeln och vem som är ansvarig för att definiera regler.
[R4] Instans av: Binär relation	Målmodellen	Aktör / Resurs	Målmodellen kan ha en aktör eller en resurs som ansvarig för målet eller definiera olika mål.
[R4] Instans av: Binär relation	Kravmodellen	Aktör / Resurs	Kravmodellen har en aktör eller resurs som ansvarig för att definiera eller ansvara för olika krav.

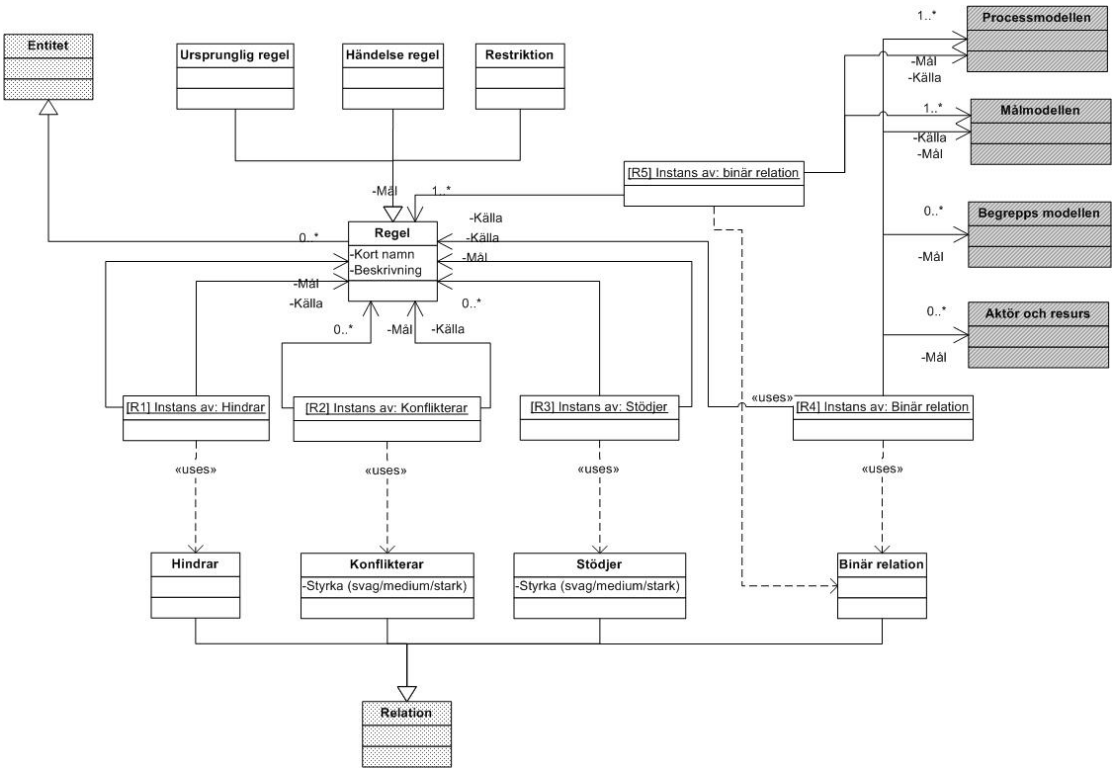
Tabell 4:3 Relationer i Aktör och Resursmodellen

4.1.4 Regelmodellen (Business Rules Model)

I denna modelltyp definieras affärsregler. Dessa skall definieras för att de som arbetar med att ta fram modeller, exempelvis målmodellen, skall veta vilka restriktioner som måste hållas. Reglerna är formade i en hierarki.

Reglerna är delade i tre delar; ursprungliga regler, händelsebaserade regler och restriktioner. De ursprungliga reglerna innehåller redan existerande regler och policys för organisationen. Händelsebaserade regler är regler eller policys som inträffar när vissa villkor inträffar. Restriktioner är regler för att inte kunna göra vissa fel. Dessa delas in i ytterligare två kategorier; fasta restriktioner, restriktioner som inte förändras över tid och övergångsrestriktioner som innebär en restriktion som kan inträffa vid en händelse. Reglerna använder samma notation i modellen för att rita ut denna entitet. Vilken typ av regel det är markeras på entiteten.

I affärsregelsmodellen finns tre sorters relationer. Den första stödjer, den andra konflikterar den entitet som den pekar på. Dessa två relationer har även en skala på hur starkt den stödjer eller konflikterar, dessa nivåer är svag, medium och stark. Den tredje hindrar den komponenten som relationen pekar på. Dessa kan kombineras i ”och” och ”eller” villkor. Figur 4:7 visar en av mig framtagen metamodell av EKD’s regelmodell utifrån Bubenko, et al. (2001). I denna modell måste reglerna kunna identifieras unikt och det korta namnet på entiteten Regel skall alltid vara unik.



Figur 4:7 Metamodell av EKD's Regelmodell

Modellen har fyra olika relationer som används inom modellen och en binär relation för att knyta till andra modeller. De andra modeller som denna modell har kopplingar till är Aktör och Resursmodellen, Begreppsmodellen och målmodellen. Relationerna och kopplingarna till de andra modellerna förklaras tydligare i Tabell 4:4.

Relationsnamn	Källa	Mål	Kommentar
[R1] Instans av: Hindrar	Regel	Regel	En regel kan hindra en eller flera andra regler.
[R2] Instans av: Konflikterar	Regel	Regel	En regel kan vara i konflikt med en eller flera andra regler.
[R3] Instans av: Stödjer	Regel	Regel	En regel kan stödja en eller flera andra regler.
[R4] Instans av: Binär relation	Regel	Målmodellen	Regler definieras utifrån målen. Regler har därför kopplingar till ett eller flera mål.
[R4] Instans av: Binär relation	Regel	Begreppsmodellen	Regler kan trigga processer genom att påverka olika saker. Dessa saker definieras i begreppsmodellen och därför kan det finnas en koppling mellan en regel och ett eller flera begrepp.
[R4] Instans av: Binär relation	Regel	Aktör och resursmodellen	För att utse ansvarig för en regel eller utse någon som definierar regler finns även en koppling till Aktör och Resursmodellen.
[R4] Instans av: Binär relation	Regel	Processmodellen	Regler kan trigga processer att utföras
[R5] Instans av: Binär relation	Processmodellen	Regel	Processer kan stödjas av uppsatta regler.
[R5] Instans av: Binär relation	Målmodellen	Regel	Mål motiveras och kräver kunskap om vilka regler som är uppsatta

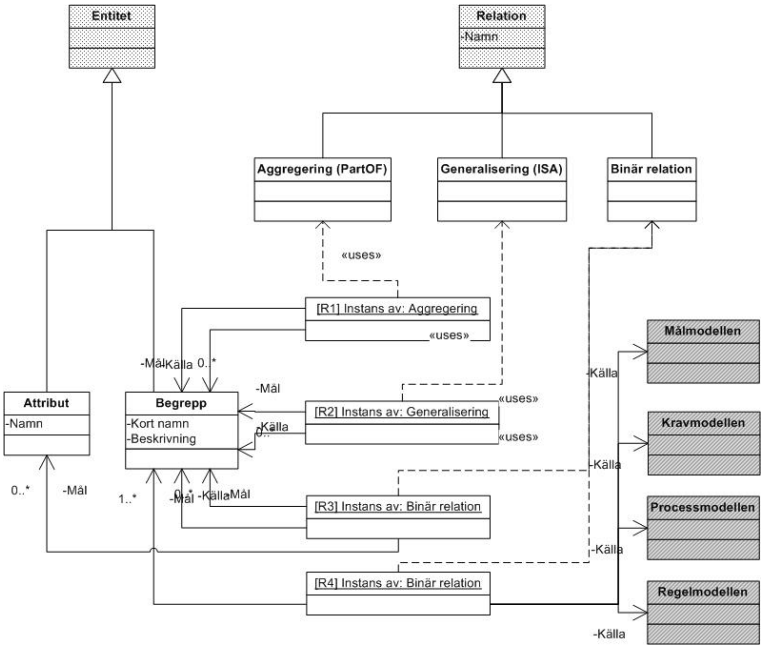
Tabell 4:4 Relationer i Regelmodellen

4.1.5 Begreppsmodell (Concepts model)

En begreppsmodell är en modell över de begrepp och koncept som organisationen använder. Den förklarar dessa begrepp och skiljer på de som liknar varandra för att minska risken för missförstånd genom att alla använder samma termer när de diskuterar domänen. Denna modelltyp är viktig då mycket kan gå fel om samma begrepp används till olika saker alternativt att olika begrepp används för ett visst fenomen. Willars (1999) tar upp vikten att ha olika begrepp för olika fenomen, och om inte det finns något bra ord för ett specifikt fenomen kan det vara bättre att ta fram ett nytt ord för fenomenet. Detta gör att just detta fenomen kan specificeras unikt med hjälp av det ordet.

Begreppsmodellen har två entiteter; begrepp och attribut. Begrepp är något i domänen som är av intresse att förklara eller mappa upp för att visa en relation. Attribut är en egenskap för begreppet.

Relationerna i denna modell har olika typer. En av dessa är av binär typ, dessa relationer är riktade åt båda hållen och har ett namn och en kardinalitet. Begreppsmodellen har även en så kallad ISA relation vilket innebär att entiteten är en variant av en annan entitet. Exempelvis kan en kund vara ett företag eller en person. Efter som de är en annan sak används en ISA relation för att visa detta. Sista typen av relationer i denna modell är PartOF relationer. Denna typ av relationer visar att en entitet är en del av en annan entitet. Ett exempel på detta kan vara saft där vatten är en del av ingredienserna. Manualen skiljer även på notationen i ISA och PartOF relationer om alla entiteter som kan ingå i den är definierade eller inte. Metamodellen för EKD's begreppsmodell har modellerats upp i Figur 4:8 utifrån Bubenko, et al. (2001). I denna identifieras entiteten Begrepp unikt via sitt korta namn.



Figur 4:8 Metamodell av EKD's Begreppsmodell

Inom modellen i Figur 4:8 används tre relationer, Generalisering, Aggregering och Binär relation. Modellen använder även den binära relationen för att koppla till andra modeller och för att knyta attribut till Begreppet. Modellens interna och externa relationer specificeras i Tabell 4:5 här under.

Relationsnamn	Källa	Mål	Kommentar
[R1] Instans av: Aggregering (PartOf)	Begrepp	Begrepp	Ett Begrepp kan vara en del av ett annat Begrepp.
[R2] Instans av: Generalisering (ISA)	Begrepp	Begrepp	Ett Begrepp kan vara en specialisering från ett annat Begrepp.
[R3] Instans av: Binär relation	Begrepp	Begrepp	Ett Begrepp kan ha relationer till ett eller flera andra Begrepp.
[R3] Instans av: Binär relation	Begrepp	Attribut	Ett Begrepp kan ha ett eller flera begrepp kopplade till sig,
[R4] Instans av: Binär relation	Kravmodellen	Begrepp	Alla funktionella krav i kravmodellen skall ha begreppen förklarade i Begreppsmodellen
[R4] Instans av: Binär relation	Processmodellen	Begrepp	Alla informationer eller material skall vara förklarade i Begreppsmodellen.
[R4] Instans av: Binär relation	Regelmodellen	Begrepp	Begrepp kan triggas genom processer. Dessa Begrepp definieras i denna modell och därför kan det finnas en koppling mellan en regel och ett eller flera begrepp.
[R4] Instans av: Binär relation	Målmodellen	Begrepp	Begreppen som finns i målmodellen definieras och förklaras i begreppsmodellen och dessa modeller är därmed sammanknutna där.

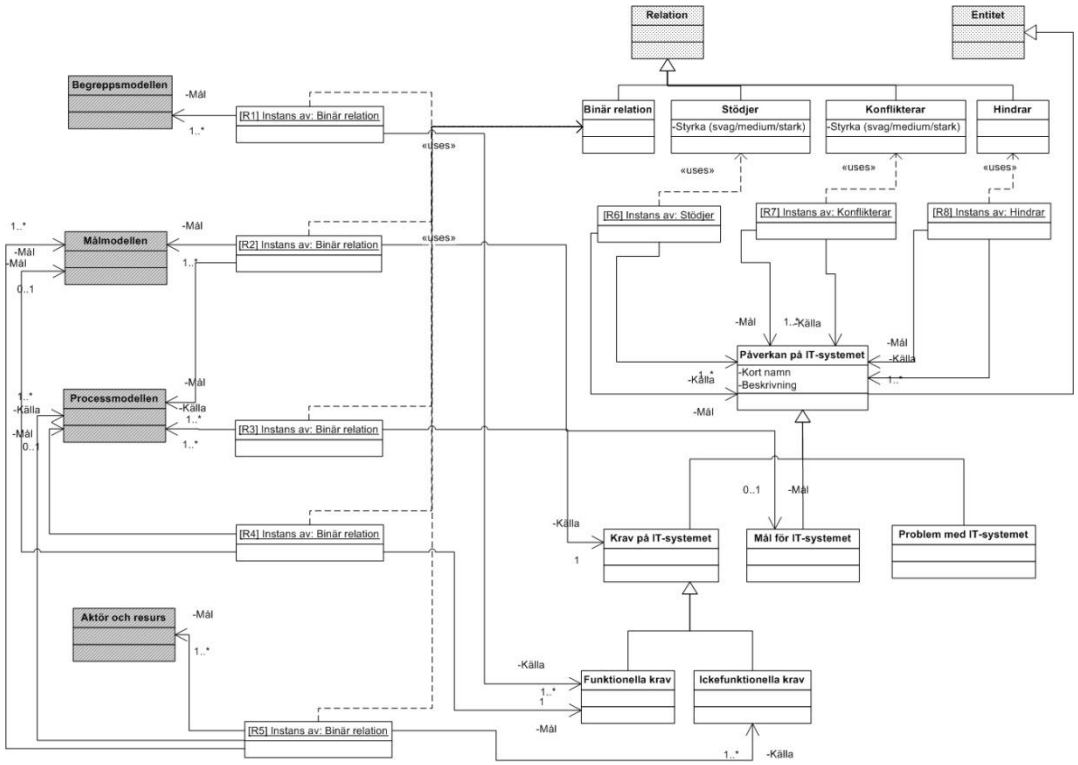
Tabell 4:5 Relationer i begreppsmodellen

4.1.6 Kravmodellen (Component of the Technical Components and Requirements Model)

Denna modelltyp tar upp vilka krav som finns på det tänkta IT-systemet. Dessa krav mappas sedan mot alla de andra modellerna där det visas att kraven uppfylls.

Modeller har en typ av entitet. Denna entitet kan vara ”Mål för IT-systemet”, Problem med IT-systemet” och ”krav på IT-systemet”. Mål för IT-systemet används för att uttrycka mål, visioner eller riktningar för IT-systemet och dess delar. Dessa mål bygger ofta på målen i målmodellen. Problem med IT-systemet visar vilka tillstånd och situationer organisationen inte vill hamna i eller uppnå. Denna typ av entiteter är oftast något som hindrar ett mål för IT-systemet. Den sista, krav på IT-systemet, tar upp specifika krav som skall vara med. Dessa krav är av typerna funktionella och icke funktionella krav.

Det finns tre relationer i denna modell. Relationerna ritas ut på samma sätt men benämns olika beroende på vad de gör. De benämningar de kan ha är Stöttande, hindrande och konflikterande. Dessa relationer kan användas som hopslagning för ”och” och ”eller” villkor. EKD’s kravmodell har en specialisering av entitet, Påverkan på IT-systemet, som identifieras unikt. Denna entitet har tre specialiseringar varav en har ytterligare två. Dessa kopplas sedan internt i modellen och till andra modeller via fyra olika sorters relationer.



Figur 4:9 Metamodell för EKD's Kravmodell

De relationer som används inom kravmodellen är Stödjer, Konflikterar och Hindrar medans Binär relation ändvänds för att koppla ihop denna modell med andra modeller i modelleringsspråket. Utförligare specifikation hur relationerna inom denna modell fungerar finns i Tabell 4:6.

Relationsnamn	Källa	Mål	Kommentar
[R1] Instans av: Binär relation	Funktionella krav	Begreppsmodellen	Funktionella krav skall vara uppsatta med en tydlig förbindelse till de begrepp som den involverar.
[R2] Instans av: Binär relation	Mål för IT-systemet	Målmodellen	Mål för IT-systemet används för just att uttrycka dessa mål som finns med i målmodellen och har därmed en tydlig koppling till dessa.
[R2] Instans av: Binär relation	Mål för IT-systemet	Processmodellen	IT-systemets mål sätts ofta med anledning av processer. Därför har Mål för IT-systemet en tydlig koppling till processmodellen.
[R3] Instans av: Binär relation	Krav på IT-systemet	Processmodellen	Ett krav på IT-systemet är alltid kopplad till en process då den är krav att utföra.
[R4] Instans av: Binär relation	Funktionella krav	Processmodellen	Funktionella krav måste vara kopplade till en process.
[R4] Instans av: Binär relation	Funktionella krav	Målmodellen	Målen i Målmodellen kan stötta funktionella krav.
[R5] Instans av: Binär relation	Ickefunktionella krav	Processmodellen	Uppsatta mål i målmodellen kan vara motiverande för ickefunktionella krav.
[R5] Instans av: Binär relation	Ickefunktionella krav	Målmodellen	Processer i processmodellen kan vara motiverande för ickefunktionella krav.
[R5] Instans av: Binär relation	Ickefunktionella krav	Aktör och resursmodellen	Ickefunktionella krav kan inkludera en medverkande från aktör och resursmodellen.
[R6] Instans av: Stödjer	Påverkan på IT-systemet	Påverkan på IT-systemet	En påverkan på IT-systemet kan stödja en eller flera andra entiteter av samma sort.
[R7] Instans av: Konflikterar	Påverkan på IT-systemet	Påverkan på IT-systemet	En påverkan på IT-systemet kan vara i konflikt med en eller flera andra entiteter av samma sort.
[R8] Instans av: Hindrar	Påverkan på IT-systemet	Påverkan på IT-systemet	En påverkan på IT-systemet kan hindra en eller flera andra entiteter av samma sort.

Tabell 4:6 Relationer i Kravmodellen

4.2 UML

Inom UML finns många olika modellertyper och vissa utav dessa modeller kan representera samma saker men inom olika områden. Enligt Booch, et al. (2008) har UML (version 2) 13 olika modelltyper. Dessa modellertyper kommer att förklaras översiktligt i avsnitt 4.2.1. I avsnitt 4.2.2 förklaras vilka modelltyper som det inom ramen för detta arbeta passar att transformera till.

4.2.1 Modelltyper i UML

De modelltyper som Booch, et al. (2008) tar upp är:

- Use case (Use case)

  Use case används för att fånga funktionella krav mellan olika aktörer och resurser. Ett use case diagram tar upp vilka aktörer som utför ett visst use case och vilka use cases som inkluderar andra use cases. Förutom en grafisk modell innefattar use case även en textbaserad beskrivning. Booch, et al. (2008) anser att use case skall användas i ett inledningsskede av en utvecklingsprocess för att fånga krav.

- Klassdiagram (Class diagram)

  Klassdiagrammen är det mest använda diagrammet av alla UML's diagram. Ett klassdiagram är tänkt att användas för att beskriva olika typer av objekt i ett system och relationer mellan dessa. Dessa diagram används inte alltid på det sätt som det är tänkt att användas, utan har egenskapen att kunna passa många ändamål. Booch, et al. (2008) menar på att klassdiagram är ryggraden i

UML och dessa används hela tiden och att de är bra underlag vid utveckling av ett system.

- Aktivitetsdiagram (Activity diagram)

  Dessa diagram visar processer och arbetsflöden. Aktivitetsdiagram kan göras på flera olika sätt och innehålla olika mycket information. De visar hur en process skall utföras och används ibland för verksamhetsmodellering enligt Booch, et al. (2008). En stor fördel med dessa är att det går att visa att vissa processer sker samtidigt och med hjälp av ”simbanor” visa vem eller vad som utför aktiviteten. Denna typ av modell används på olika ställen under utvecklingsfasen, allt från att få fram en förståelse för processerna i verksamheten till att visa processer och flöden i ett system.

- Sekvensdiagram (Sequence diagram)

  Detta diagram beskriver kommunikationsförlopp mellan olika objekt vd en viss händelse. Denna modell kan även uttrycka loopar, selektionsvillkor och liknande företeelser. Sekvensdiagram är nära hopknutet med att tänka i det objekt orienterade paradigmet. Booch, et al. (2008) tar upp två områden där de ser att sekvensdiagram kan användas med fördel; det första är för att visa hur objekt samarbetar. Det andra området är när ett use case behöver undersökas djupare, där det går med hjälp av ett sekvensdiagram, ta fram ett flöde eller ett scenario hur det kommer gå till.

- Komponentdiagram (Component diagram)

  Komponentdiagram visar hur olika komponenter i systemet är kopplade tillvarandra. Detta diagram används sällan men kan vara till stor nytta när ett system skall delas upp i komponenter.

- Installationsdiagram (Deployment diagram)

  Detta diagram visar vilken mjukvara som skall köras på vilka maskiner och hur dessa mjukvaror kommunicerar med varandra. Dessa diagram har en god funktion för en översikt när mjukvara utvecklas.

- Kommunikationsdiagram (Communication Diagrams)

  Visar hur olika komponenter interagerar med varandra genom numrerade relationer där funktionen som anropar specificeras. Dessa används sällan och när de används ersätter de oftast bara ett sekvensdiagram.

- Objektdiagram (Object Diagram)

  Dessa diagram visar en ögonblicksbild på ett antal utvalda objekt vid ett visst tillstånd. Dessa används för att kunna kommunicera och lättare skapa förståelse för vad som skall hända.

- Tillståndsdiagram (State Machine Diagram)

  Detta diagram visar ett objekts olika tillstånd. Förändring av tillståndet ritas ut med en relation till ett nytt tillstånd för objektet med en förklaring vad och hur en funktion förändrar tillståndet. Tillståndsdiagrammen kan användas för att visa hur objektets tillstånd är i systemet vid exempelvis ett use case eller ett aktivitetsdiagram.

- Tidsdiagram (Timing Diagram)

En modell som visar interaktioner med fokus på tidsbegränsningar. Dessa är bra för att kunna modellera upp system som är beroende av svarstider som är mer vanligt när mjukvara utvecklas till en hårdvara.

- Komposit strukturer (Composite Structures)

  Dessa diagram används för att visa hierarkier i en klass och kan bryta ner ett objekt av klassen till olika delar.

- Interaktionsöversikt (Interaction Overview Diagram)

  Dessa diagram är en blandning av aktivitetsdiagram och sekvensdiagram. Diagrammet visar ett aktivitetsdiagram där aktiviteterna ersätts av sekvensdiagram.

- Paketdiagram (Package Diagram)

  Detta är en modell som tar upp vilka paket, en redan utvecklad del som kan återanvändas, som finns med i systemet och vilka paket som använder sig av andra paket. Dessa kan med fördel användas vid utvecklandet av större IT-system.

4.2.2 Metamodeller för UML

Jag har sökt efter UML's metamodeller och funnit olika sätt att presentera dessa. Via OMG och IBM finner jag modeller som är delade i flera beståndsdelar. Därför är inte dessa modeller möjliga att jämföra med de metamodeller som skapats för EKD. För att göra en bra jämförelse mellan metamodellerna behöver UML's metamodeller modelleras upp på nytt utifrån den struktur EKD's metamodeller har. Begränsningen i tiden för detta examensarbete hindrar mig att utföra detta utan jag kommer att förklara och visa i EKD's modeller vilken data som den utvalda UML modellen kan representera.

4.2.3 Modelltyper att transformera till

Under detta stycke kommer arbetet att fokusera mer på vilka modeller i modelleringsspråket UML som kan uttrycka EKD's framtagna information. När dessa alternativ tagits fram kommer frågan om vilka modeller som troligast kan representera EKD's modeller med minsta möjliga informationsförlust att tas upp. När detta är gjort kommer en djupare analys av de metamodeller som tagits fram för EKD och jämföras mot metamodeller för de modelltyper i UML som valts att göras.

Här under kommer arbetet att redogöra för vilka modeller som kan passa för att fånga de uttryck som EKD har i UML:

- Målmodellen

  I UML finns ingen modelltyp som är specifikt utformad för att representera mål. Beroende på vilken sorts mål det finns det möjlighet att de kan uttryckas som ett krav i ett use case diagram eftersom dessa uttrycker de funktionella krav som finns. Dock finns det många andra typer av mål i EKD som inte kan uttryckas som ett use case.

- Aktörer och resursmodellen

Aktörer och resurser har ingen egen modell i UML men är ändå ofta representerade i flera olika modeller. Exempel på dessa är use case, där olika aktörer och resurser visas i samband med vilka uppgifter de skall kunna utföra. I aktivitetsdiagram går det att använda sig av såkallade simbanor för att koppla en process till en aktör eller resurs. I UML's sekvensdiagram tas även aktörer och resurser upp för att visa vem som utför en viss process och i vilken ordning. Utöver dessa diagram som har aktörer och resurser inbakade i sig finns inget diagram som förklarar vilka aktörer och resurser som finns att tillgå.

- Processmodellen

UML har olika modeller för att representera processer. Vid första anblicken ser aktivitetsdiagram ut som en modell som skulle passa bra då den tar upp processer på ett liknande sätt som processmodellen. Den ger även en överblick vilka aktörer organisationen har som krävs för att utföra en process. Sekvensdiagram är ett av UML's diagram. Sekvensdiagram kan användas för att visa aktörer och processer och hur de anropar varandra. Den nya modelltypen som heter Interaktionsöversikt är även intressant då den kan specificera lite utav de båda modellerna.

- Regelmodellen

UML har inga modeller som är tänkta att användas för att dokumentera och använda sig av regler av denna typ eftersom UML är inriktat på systemspecifika detaljer.

- Begreppsmodellen

UML har inga modeller för att modellera och förklara begrepp i domänen i sitt orginalutförande.

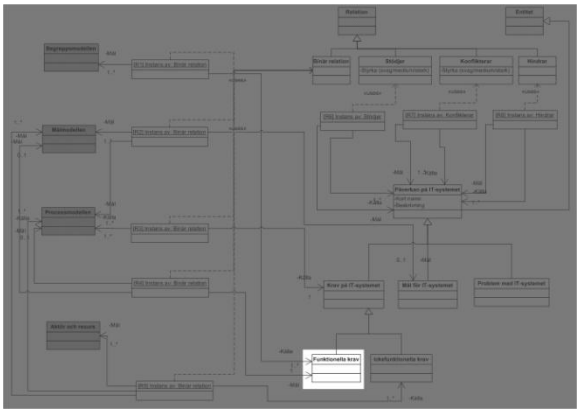
- Kravmodellen

För att fånga krav har UML få diagram. Use case diagrammen är de som tydligast visar på att de skall användas för att arbeta med krav. Use case används för att fånga funktionella krav, hur systemet skall fungera. För icke funktionella krav har UML inte något passande diagram. Det finns vissa diagram som kan fånga vissa krav. Ett exempel på detta är Tidsdiagrammet som kan användas för att dokumentera ickefunktionella tidskrav.

Utifrån den diskussion ovan finner jag att UML har tre modelltyper som kan användas för att få in stora delar av EKD. Dessa modelltyper är use case, Aktivitetsdiagram och Sekvensdiagram.

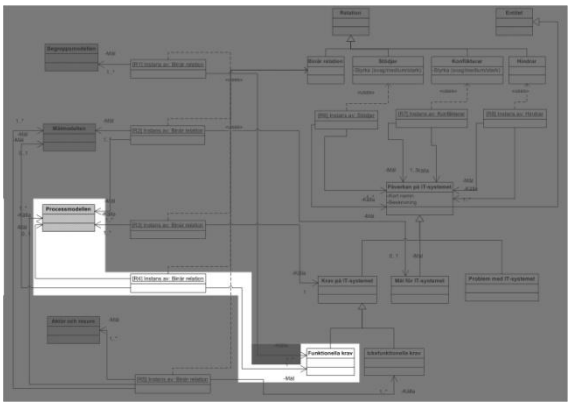
4.2.3.1 Use case

UML har bara ett diagram för att fånga krav och detta är use case diagrammet. Den tar bara upp funktionella krav för att alltid veta att utvecklingen riktar in sig på nödvändiga bitar Booch, et al. (2007). I use case diagrammet visas alltid vilka aktörer och resurser som utför det use caset, alltså det funktionella kravet. I EKD's kravmodell har inte det funktionella kravet någon direkt koppling till aktörer och resurser och blir därför en isolerad entitet som Figur 4:10 visar.

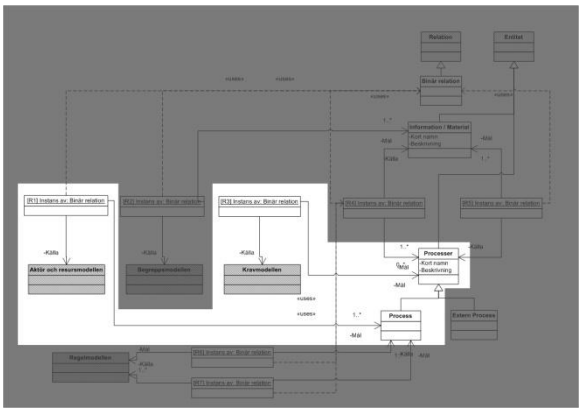


Figur 4:10 Funktionellt krav isolerat i Kravmodellen

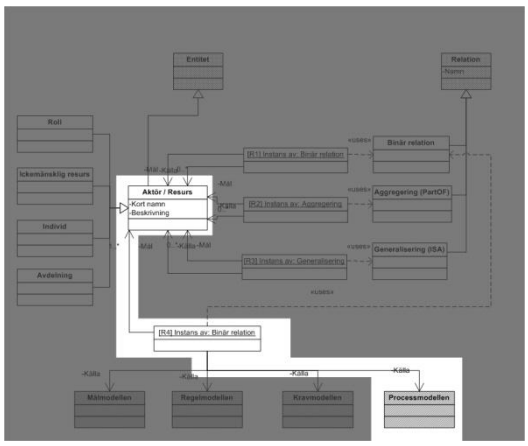
I förklaringen till EKD's kravmodell står det att alla krav är knutna till en process i och med det får kraven en koppling till processmodellen som visas i Figur 4:11. Denna process i sin tur är knuten till aktör och resurs modellen (Figur 4:12) där den har en koppling till just aktör eller resurs (Figur 4:13).



Figur 4:11 Krav har en koppling till processmodellen



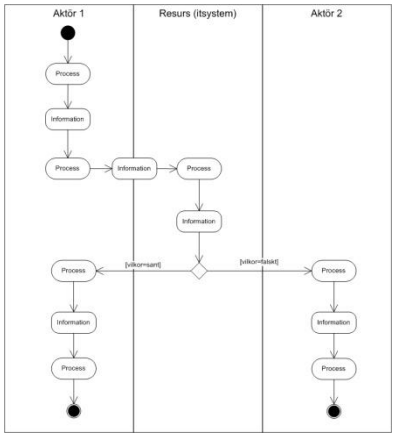
Figur 4:12 Krav är kopplat till Processer i Processmodellen. Process ärver från Processer och har alltid en koppling till Aktör och Resursmodellen.



Figur 4:13 Processmodellen har en koppling till Aktör / Resurs i Aktör och Resursmodellen.

4.2.3.2 Aktivitetsdiagram

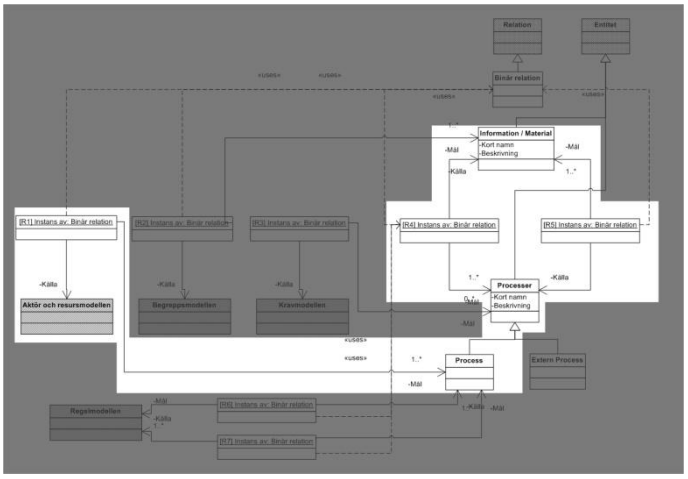
Aktivitetsdiagrammet kan utifrån EKD's modeller visa processer, både interna och externa, och vilken ordning dessa utförs. De kan även visa om modelleraren väljer att ha med "simbanorna", som de kallas, i modellen, vilken aktör eller resurs som utför processen i modellen visualiseras i Figur 4:14.



Figur 4:14 Exempel av UML's Aktivitetsdiagram

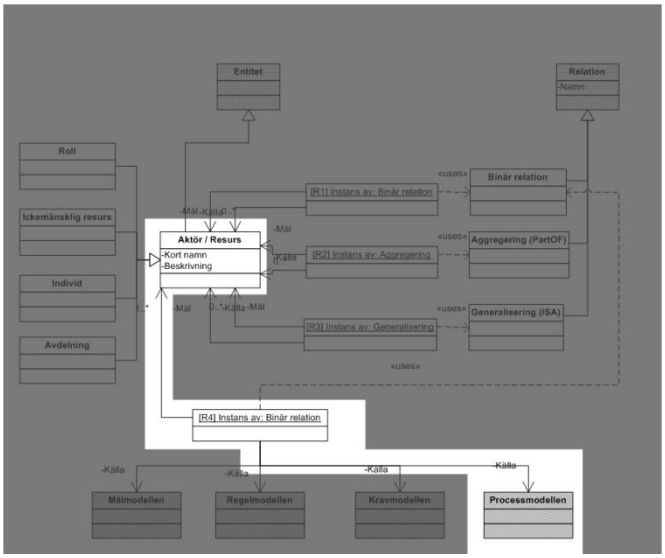
En processmodell i EKD har alltid en Externprocess som startar en processmodell. EKD's processmodell skiljer på dessa olika typer av processer och visar detta med olika utseende på entiteterna för dessa i modellen. UML's Aktivitetsdiagram startar oftast med en ifylld punkt som pekar på nästa process som illustreras i Figur 4:14. UML har även en annan metod för att starta en process och det är genom startsignaler, även kallad ”accept signal”, som inväntar en startsignal för att kunna köra igång processen eller en ”timesignal” där startsignalen utgörs av exempelvis en timer för att starta processen. Att det inte finns ett enhetligt sätt gör att det ibland är otydligt hur processen startar och hur den avslutas. Den som tolkar modellen kan inte tydligt se om eller hur processen fortsätter. Ett sådant exempel finns i Figur 4:14, där det inte går att se var processen kommer ifrån eller var den tar vägen i och med att modellen både startar och slutar i ringar.

EKD definierar även mellan varje process vilken typ av material eller information som skapas och konsumeras. I Aktivitetsdiagram används normalt en valig enkel pil mellan de olika processerna för att visa hur flödet rör sig. UML har från grunden några andra kombinationer på relationerna mellan processerna i aktivitetsdiagrammet men enligt Booch, et al. (2008) är det inte lika vanligt att använda dessa som vanliga pilar. Detta lilla tillägg i UML gör att det går att ha informationer mellan processerna även i UML. Detta kallas objektflöden (Object Flow) och är en del av UML's sekvensdiagram. I Figur 4:14 visas ett objekts flöde med hjälp av aktivitetsdiagram. Denna visar på hur det går att ha en information/förändrad information mellan varje process som objektet passerar. Med detta tillägg som ändå är med i UML's uppsatta standard kan vi, som Figur 4:15 visar, få med större delen av funktionaliteten som EKD's processmodeller har inom modellen.



Figur 4:15 Den del av EKD's processmodell som UML kan visa.

I UML's Aktivitetsdiagram finns även möjlighet att använda sig av simbanor, detta är dock inget krav men för att få ut en bra övergång till EKD passar det bra att ha med det, dessa simbanor visas i exemplet i Figur 4:14. Detta ger i sin tur att modellen i Figur 4:15 har en koppling från process till aktör och resursmodellen och i den modellen, som Figur 4:16 visar, har en koppling till aktör / resurs.

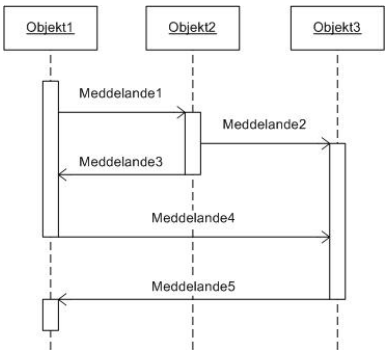


Figur 4:16 Processmodellens knytning till Aktör / Resurs

Både EKD och UML tillåter användning av submodeller och dessa används av båda på ett liknande sätt men UML har ett sätt att definiera en selektion, som Figur 4:14 visar, där givna val ges. Detta saknar notationen för EKD.

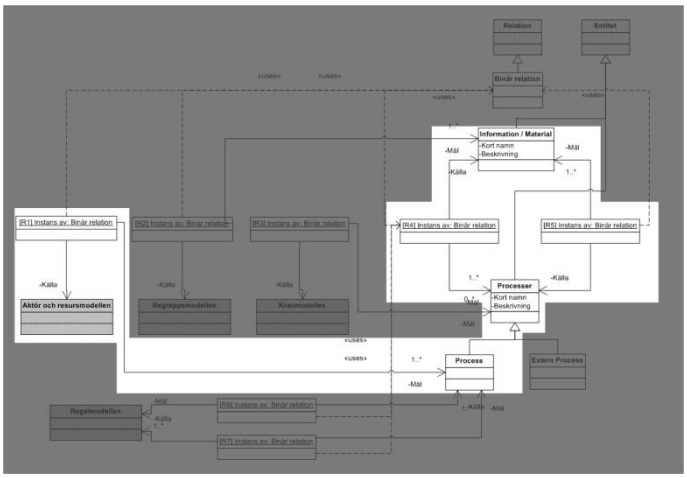
4.2.3.3 Sekvensdiagram

Sekvensdiagram används normalt för att beskriva en enskild händelse eller process och om hur data som skickas fram och tillbaka mellan objekt i ett system. I Figur 4:17 går det att se ett exempel på hur ett sekvensdiagram kan se ut. I detta exempel finns tre objekt med olika livslängd som kommunicerar med varandra vid en specifik uppgift.



Figur 4:17 Exempel på hur ett sekvensdiagram kan vara utformat

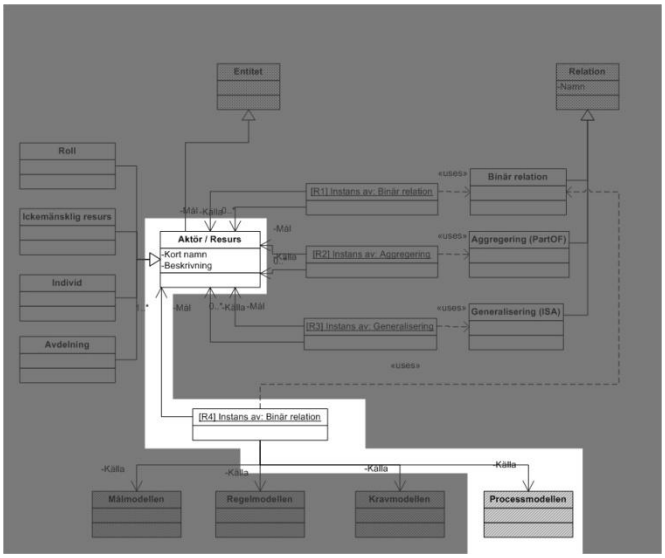
Med denna modelltyp går det att ha med vilka aktörer och resurser som är inblandade och hur de kommunicerar och under vilken del av förloppet detta sker med dessa objekt. Sekvensdiagrammet kan vara ett alternativ för att presentera EKD's processmodeller eller bitar utav den och få med mer informations som skickas mellan olika aktörer och i systemet för att slutföra processen men inte i samma mån som EKD's processmodell. Eftersom det i viss mån går att specificera information som skickas mellan objekten går det, om modelleraren väljer att använda modellen på det sättet, visualisera samma saker som UML's aktivitetsdiagram som Figur 4:18 visualiserar.



Figur 4:18 Processmodellen med stor del av sin funktionalitet framtagen.

Eftersom sekvensdiagrammet även kan specificera aktörer och resurser utöver de objekt som kommunicerar inom IT-systemet finns det en naturlig koppling till EKD's

aktör och resursmodell på det sätt som Figur 4:19 visar. Sekvensdiagram har precis som aktivitetsdiagram en notation för alternativ i modellen. Upprepning och loopar går att uttrycka i aktivitetsdiagram men eftersom detta är väldigt vanligt inom IT-system har sekvensdiagram en egen notation för att lösa detta.



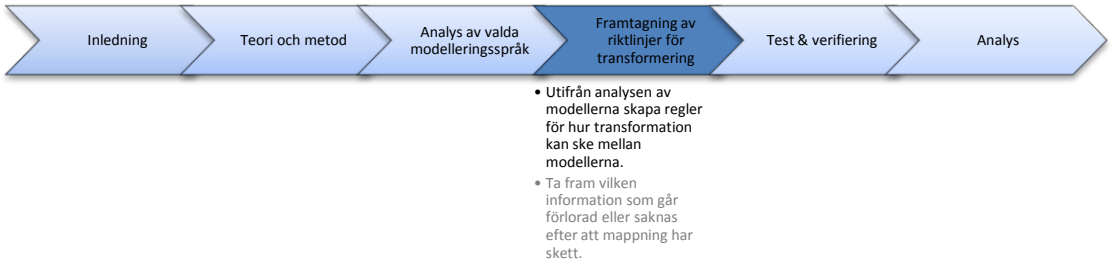
Figur 4:19 Vägen från processmodellen till Aktör /Resurs i Aktör och Resursmodellen.

4.3 Slutsats om val av modelltyper

Det finns många olika modelltyper i UML. Flera av dessa är mycket systemfokuserade och flera av modellerna kan tyckas beskriva samma sak, exempelvis beskrivning av ett flöde, dessa flödesbeskrivande modeller är dock anpassade för att beskriva flöden i olika situationer. De som fungerade enligt den diskussion som fördes i 4.2.3 var use case, aktivitetsdiagram och sekvensdiagram. Aktivitetsdiagram har dock en större potential för att uppnå önskat resultat vid en transformation av processmodellen då den bland annat har stöd för submodeller och en egen notation för att visa detta. Normalt används vanliga pilar mellan olika processer för att visa flödet. UML har en notation för att ha en information mellan processerna och även om den inte är vanlig när modellering i UML utförs används den ibland och passar vårt behov bra. Därför kommer detta arbete fortsättningsvis att titta endast på aktivitetsdiagram när det gäller att transformera EKD’s processmodell och use case när det gäller att transformera funktionella krav.

5 Riktlinjer för transformering

Nästa steg enlig den processmodell i Figur 5:1 som detta arbete utgår ifrån kommer, utifrån den analys av modeller som gjordes i kapitel 4. Analys av valda modelleringsspråk görs och riktlinjer transformation mellan de valda modellerna i EKD och UML tas fram.

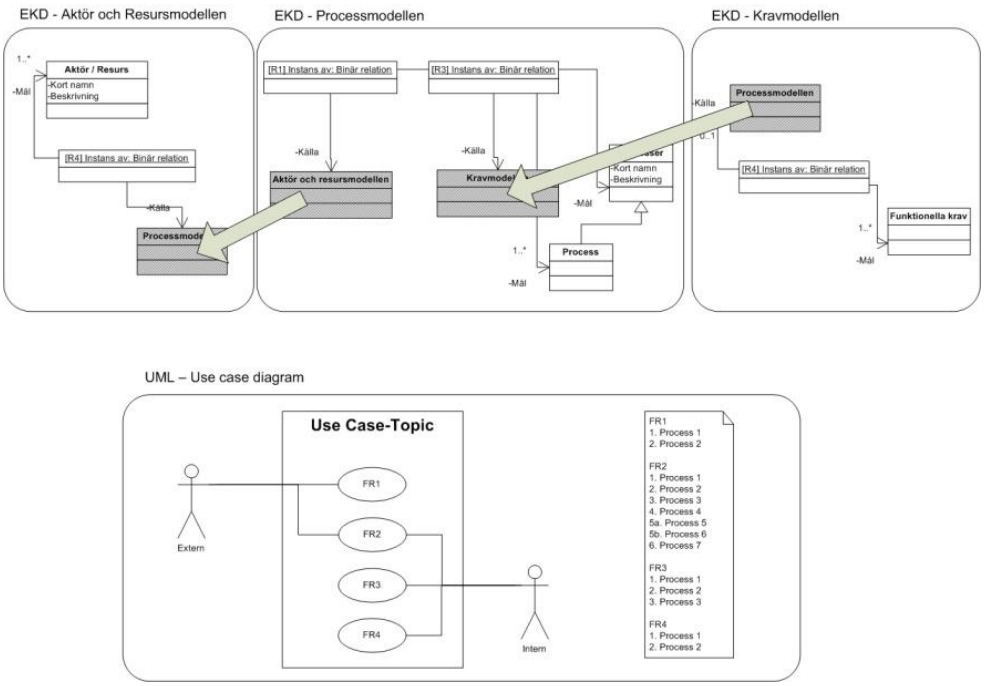


Figur 5:1 Arbetsbeskrivning - Framtagning av transformationsriktlinjer

De modeller som det passade bäst att transformera till var use case diagram och aktivitetsdiagram. Transformeringen kommer först att förklaras för att sedan avslutas med riktlinjerna för överföringen till just den modellen. Dessa transformationer kommer att ritas upp och illustreras avsnitten 5.1 och 5.2.

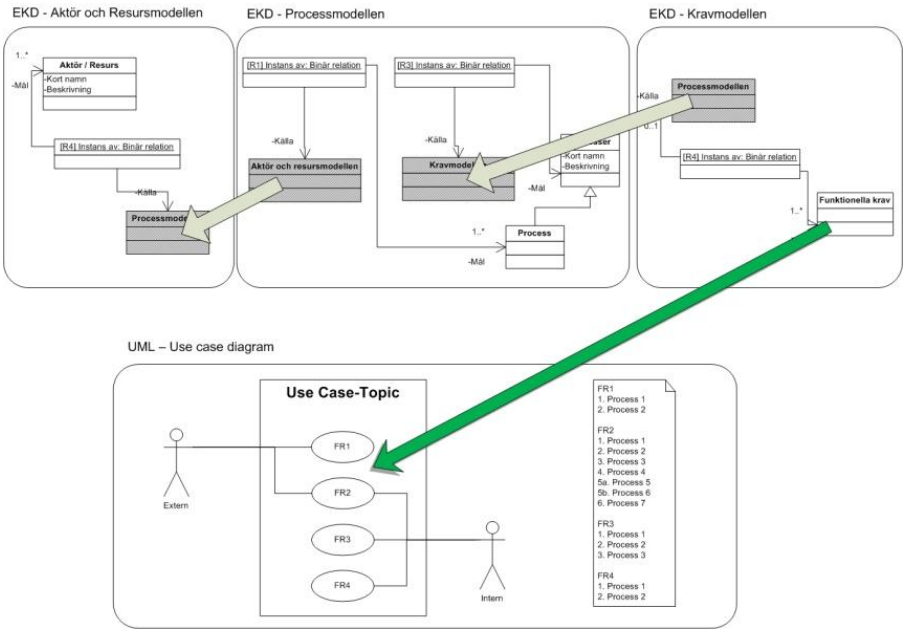
5.1 Riktlinjer för mappning av EKD modeller till use case diagram.

I detta avsnitt kommer transformationen från EKD modeller till use case att beskrivas. I Figur 5:2 finns den grund som denna transformationsprocess kommer illustreras utifrån. De tre översta modellerna är, från vänster till höger, EKD's Aktörs- och resursmodell, Processmodell och Kravmodell. Dessa är utdrag från de tidigare skapade metamodellerna: Aktör och resursmodellen - Figur 4:6, Processmodellen - Figur 4:5 och Kravmodellen - Figur 4:9. Dessa hänger ihop via de mörkare entiteterna med namnet på den mottagande modellen. Övergången visas i denna modell med hjälp av pilar mellan dessa entiteter. Under dessa tre modeller finns ett fiktivt use case diagram med en förklaring till de olika use case som är uppsatta.



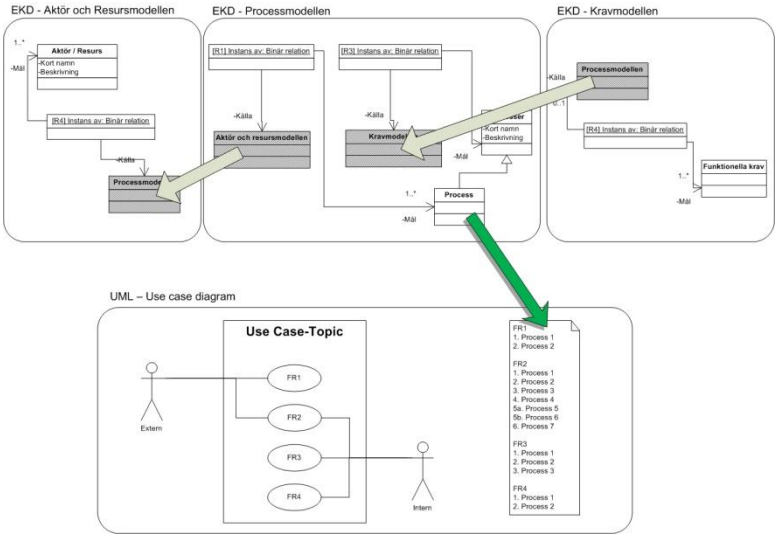
Figur 5:2 Grund för transformation mellan EKD och UML's use case diagram

I Figur 5:2 är de uppsatta use casen i detta exempel namngivna till FR1 till FR4. Detta står för Functional Requirement 1 – 4. Dessa namn har de fått på grund av att enligt Booch, et al. (2008) är ett use case ett funktionellt krav. I EKD finns även funktionella krav i kravmodellen. I Figur 5:3 visas att dessa funktionella krav kan alltså översättas från funktionella krav i EKD’s kravmodell till use case i UML’s use case diagram eftersom båda dessa uttrycker samma sak enligt specifikationerna som denna rapport tidigare tagit upp i avsnitt 4.2.3.1.



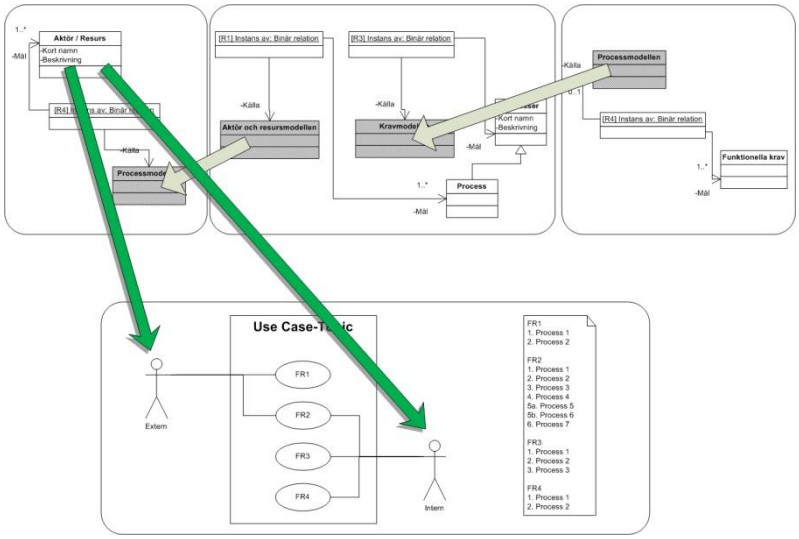
Figur 5:3 Mappning av Funktionella krav till use case.

I Figur 5:3 går det att se att det finns en eller flera aktörer eller resurser kopplade till de olika use casen. Dessa har översatts till Funktionella krav i EKD's kravmodell. Vägen mellan Funktionella krav och Aktör / Resurs i EKD går genom Processmodellen. Ett use case kan brytas ner och består ofta av flera olika processer som skall utföras för att uppfyllas. I avsnitt 4.2.1 står det om den textbaserade beskrivningen av use caset. Denna beskrivning kan se olika ut men den innehåller en beskrivning på vilka processer som behöver utföras. Beskrivningen på processerna är ofta ytliga och går inte på djupet. Dessa processer går att använda för att sedan göra submodeller som innehåller processen i mer detaljerade form. De Funktionella kraven i EKD's Kravmodell är alltså kopplade till en eller flera processer. Även i avsnitt 4.2.3.1 förklarar detta arbete hur denna har en koppling mot UML's use case. Dessa processer kan överföras till den textbaserade beskrivningen för use caset som Figur 5:4 visar.



Figur 5:4 Mappning av processer till use case.

Dessa processer som visas i Figur 5:4 ska alltid ha en aktör som utför och en aktör som ansvarar för denna process. Detta går att se i Figur 5:4 där det går en relation mellan process och aktör och resursmodellen. Denna relation fortsätter i Aktörs- och Resursmodellen och går till entiteten Aktör / Resurs. Den aktör och/eller resurs som är inblandad i att utföra processen är även en del i att uppfylla det funktionella kravet. Därför kan de aktörer och resurser som är ansvariga för att utföra processerna i EKD mappas till UML's use case diagram som aktörer som interagerar med de processerna. Detta beskrivs även i avsnitt 4.2.3.1 och illustreras här under i Figur 5:5.



Figur 5:5 Mappning av Aktör / Resurs till use case.

Ett tips innan denna mappning utförs är att gruppera funktionella krav i EKD's kravmodell och utföra denna mappning gruppvis på flera olika use case diagram för att fortsätta kunna ha en bra överblick på dessa vilket är ett av kraven som går under pragmatisk kvalité, för att en modell skall hålla hög kvalité enligt Krogstie, et al. (2006).

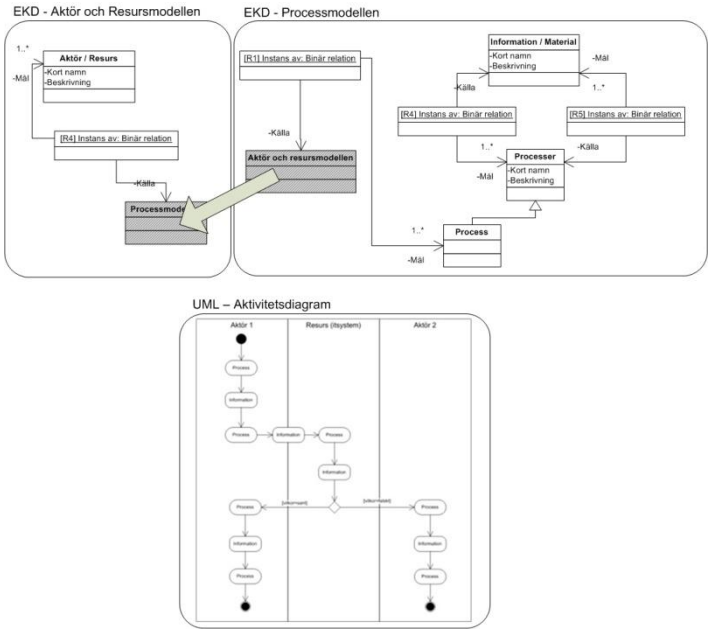
Som en följd av ovanstående redogörelse följer följande riktlinjer för mappning till use case diagram:

1 Gruppera och välj ut funktionella krav enligt tipset ovan.
2 Funktionella krav i EKD's Kravmodell mappas mot use case i UML's use case diagram (Figur 5:3).
3 De processer i EKD's processmodell som har kopplingar till det funktionella krav som analyseras tas fram och förklaras i textbeskrivningen för use caseet (Figur 5:4).
4 De aktörer och/eller resurser som utför den processen knyts till det specifika use caset (Figur 5:5).

Denna process görs för alla funktionella krav.

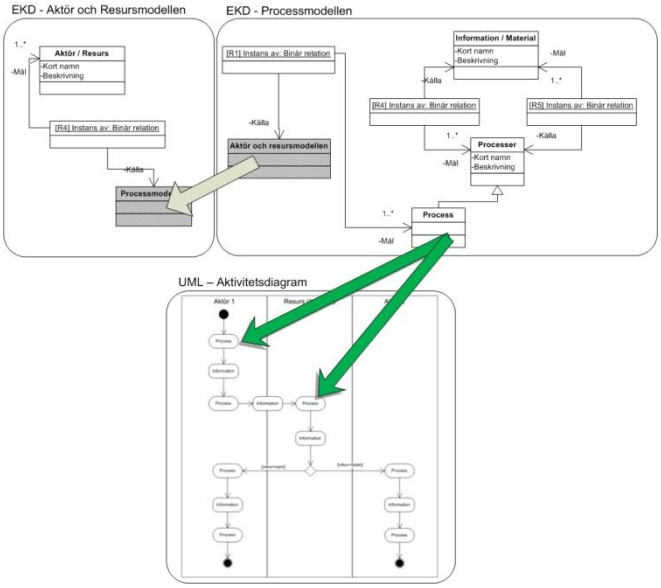
5.2 Riktlinjer för mappning av EKD modeller till Aktivitetsdiagram

Arbetet har tagit upp två olika modeller som ansågs lämpliga att föra över processer till men kom fram till att aktivitetsdiagram passade mer utav dessa två. I Figur 5:6 visas den grund som kommer att användas för att utföra en transformation mellan EKD och UML's aktivitetsdiagram. De EKD modeller som transformationen kommer använda sig av är Aktörs- och Resursmodellen och Processmodellen, dessa är presenterade högst upp i figurerna. De modellerna bygger på metamodellerna som skapats tidigare: Aktörs- och resursmodellen - Figur 4:6 och Processmodellen - Figur 4:5. Modellerna hänger ihop via den mörkare entiteten med modellens namn som det finns en koppling till. Själva kopplingen visualiseras i denna modell med en pil som binder ihop dem.



Figur 5:6 Grund för mappning mellan EKD och UML's Aktivitetsdiagram

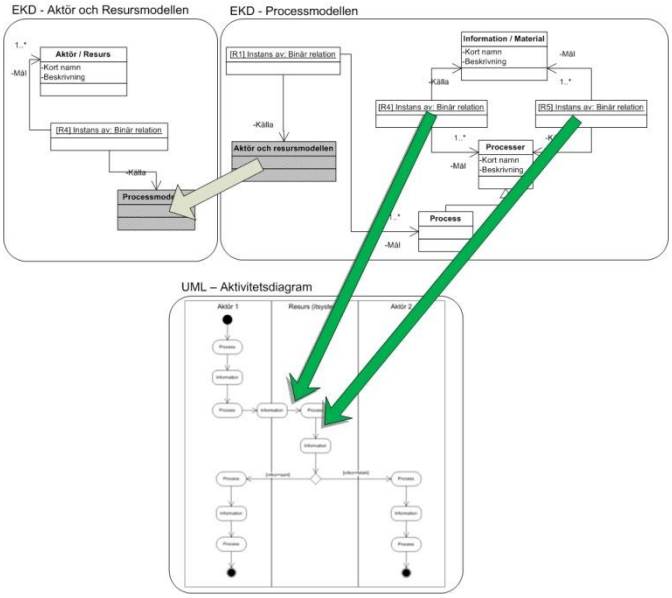
UML’s Aktivitetsdiagram startar med en process efter punkten som går att se i Figur 5:6. Processer hittas i EKD’s processdiagram och denna kan mappas rakt över till process, eller aktivitet som den även kallas, i UML’s aktivitetsdiagram. Figur 5:7 visar två exempel på hur EKD’s processer kan mappas mot UML’s processer / aktiviteter.



Figur 5:7 Mappning av Processer till Aktivitetsdiagram

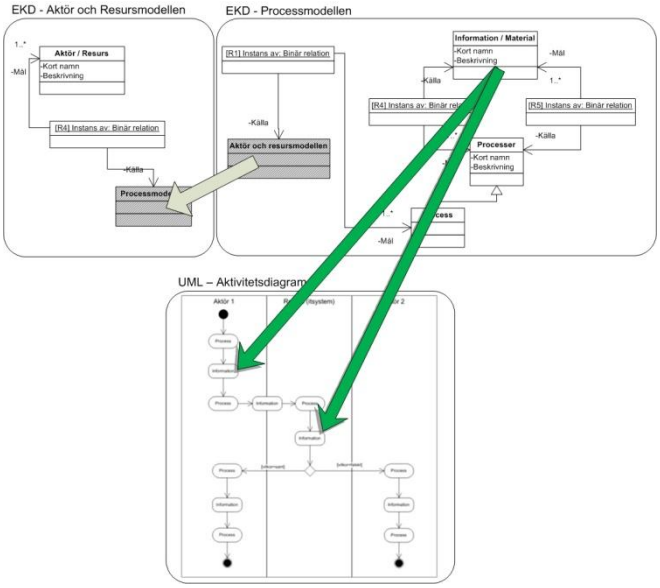
Processen har kopplingar till information, en process är alltid kopplad till en eller flera informationer och informationen i sin tur är alltid kopplad till en eller flera processer. Relationen till en information / material från en process och relationen från en information / material till en process översätts enligt som Figur 5:8 en relation från en

process till en information och från en information till en process i EKD's Processmodell till UML's Aktivitetsdiagram.



Figur 5:8 Mappning av interna relationer till Aktivitetsdiagram

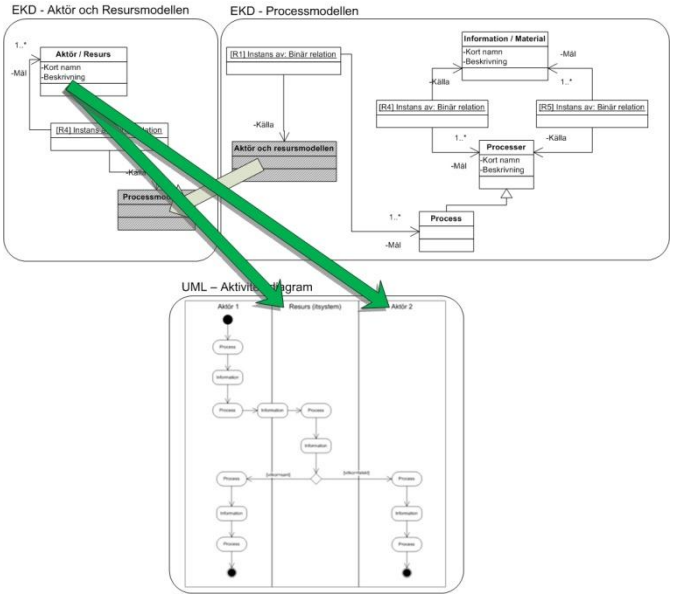
Processen och relationerna till informationen har nu mappats från EKD modellen till UML's sekvensdiagram enligt Figur 5:7 och Figur 5:8. Information mappas till informationsentiteten i UML diagrammet enligt Figur 5:9. Denna entitet är valbar och det är inte vanligt att den används. Att ha med den ger dock en bättre översättning av EKD mot UML och finns med i standarden för UML-notationen.



Figur 5:9 Mappning av information till Aktivitetsdiagram

Den sista mappningen som kan utföras är att få med vilken aktör eller resurs som utför uppgiften. Processmodellen har en övergång från Processmodellen till Aktör och Resursmodellen. Denna relation säger att varje process skall ha utförare och ansvarig kopplad till varje process. Den aktör eller resurs som utför processen mappas enligt

Figur 5:10 från Aktör / Resurs i EKD's Aktörs- och Resursmodell till simbanorna i UML's Aktivitetsdiagram.



Figur 5:10 Mappning av Aktör / Resurs till Aktivitetsdiagram

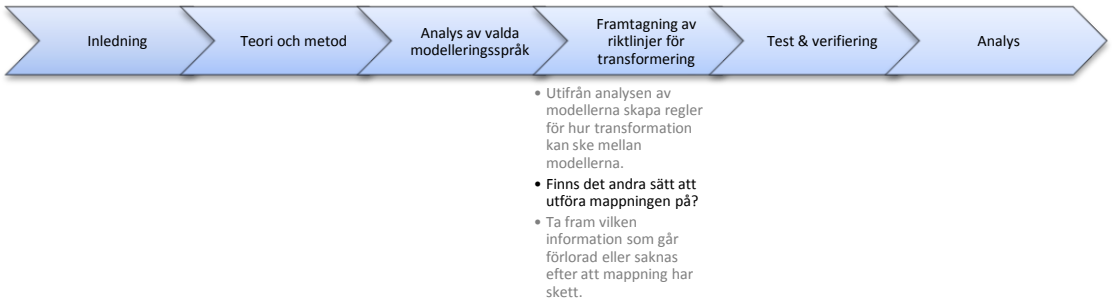
Ett tips innan denna mappning utförs är att gruppera sammanhängande processer för att antalet inte ska påverka överskådligheten negativt. Blir det för många processer går det att slå ihop processerna och göra en submodell av dessa. Är det många olika utförare på processerna kan detta leda till att överskådligheten på modellerna minskar och detta resulterar i att den pragmatiska kvalitén för modellen sänks (Krogstie, et al. 2006). För att förebygga detta bör därför processerna spridas ut över flera modeller.

Riktlinjerna för mappning till Aktivitetsdiagram som denna diskussion resulterar i är:

1 Gruppera sammanhängande processer.
2 Se om det är många olika utförare på de olika processerna. Om så är fallet, försök att dela isär gruppen ytterligare för att få ett mer lagom antal.
3 Granska om antalet processer är lagom för att ge en överskådlig modell för den modell som skall ritas upp. Om inte, försök att lämpliga grupper av processer att slå ihop till en egen submodell.
4 Sätt ut simbanor för de olika Aktörer och resurser som definierats (Figur 5:10).
5 Sätt ut en startpunkt i den simbanan där processen skall starta.
6 Sätt ut kommande information mellan startpunkt och kommande process (Figur 5:9).
7 Sätt ut kommande process i simbanan som processen tillhör (Figur 5:7).
8 Dra relationen mellan informationen och processen.
9 Dra relationen mellan processen och informationen.
10 Sätt ut kommande information mellan förgående och kommande process (Figur 5:9).
11 Upprepa steg 7 till 10 tills processer och informationer är slut.
12 Avsluta med att sätta ut en avslutningspunkt i aktivitetsdiagrammet för varje process som inte har ett slut och dra en relation mellan avslutningspunkten och processen.

6 Alternativ att utföra mappningen på

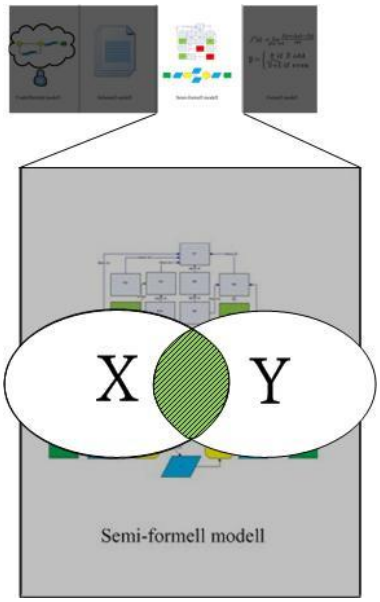
UML's modeller kan ändvändas på flera sätt. Detta skriver både Booch, et al. (2008) och Eriksson & Penker (2000) i sina böcker. I detta kapitel kommer detta arbete, som Figur 6:1 visar, ta upp andra tänkbara sätt att utföra mappningen.



Figur 6:1 Arbetsprocess – Andra sätt att utföra mappningen på.

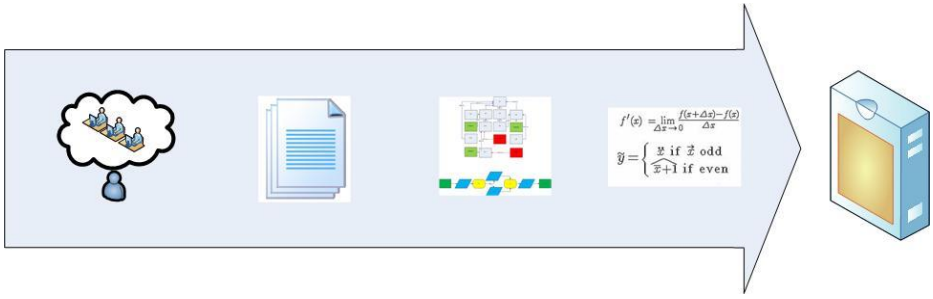
Att använda ett modelleringsspråk som en grund och komplettera eller förändra det utefter de behov är något som är vanligt. Detta är inget som är specifikt för UML, Bubenko, et al. (2001) tar upp att andra modeller kan användas vid behov för att komplettera EKD för att kunna passa specifika behov. Även Norman, P. (2010) förklarar att Astrakans modeller kan kompletteras och utökas med hjälp av andra modeller för att hjälpa till att fylla syftet. I modelleringsspråket UML är det vanligt med utökningar. Ett exempel är rapporten Baumeister, Koch & Mandel. (1999), där det tas upp hur det går att utöka UML, som de säger är en accepterad standard för att designa objektorienterade system. I Eriksson & Penker (2000) tar även de upp hur UML kan användas på andra sätt än vad det är tänkt för, de använder UML för att modellera en organisation.

I ett arbete som detta finns det andra tänkbara sätt att göra en mer komplett transformering med hjälp av en sådan utökning eller förändring av språket. Ett exempel skulle kunna vara som Booch, et al. (2008) tar upp, att klassdiagram ofta används till andra syften än vad de är menade för. De kan användas för att bland annat fånga upp uppsatta mål. Att använda UML som Eriksson & Penker (2000) skriver om hade kunnat leda till att en stor del, om inte allt, hade kunnat överföras. Att använda sig av en transformation från EKD till en omgjord modell i UML som visar samma sak skapar bara redundans och är inte alls vad detta arbete är ute eftersom arbetet fokuserar på en övergång från det mindre formella semi-formella modelleringsspråket EKD till det mer formella semi-formella modelleringsspråket UML som Figur 6:2 visar.



Figur 6:2 Överlappande information mellan semi-formella modelleringsspråk.

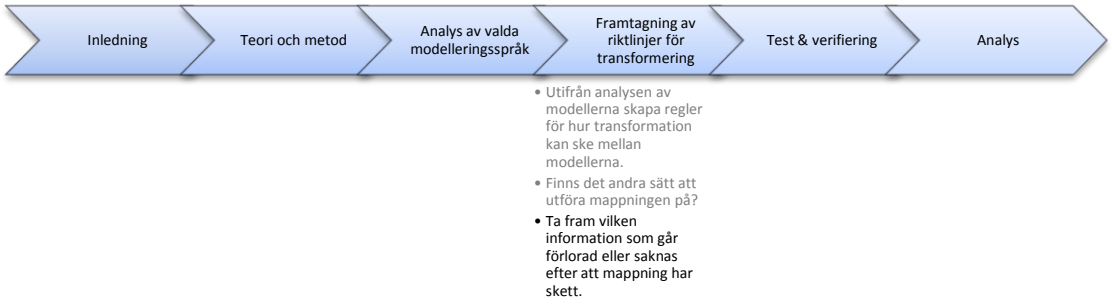
Med hjälp av det studerade överlappet av information som finns hos de båda modelleringsspråken anser jag att det blir lättare kunna nå en mer systemnära modell som kan användas som ett underlag för att bygga ett IT-system på, Figur 6:3.



Figur 6:3 Från informell modell till färdigt system.

7 Förlorad information och kompletteringsbehov

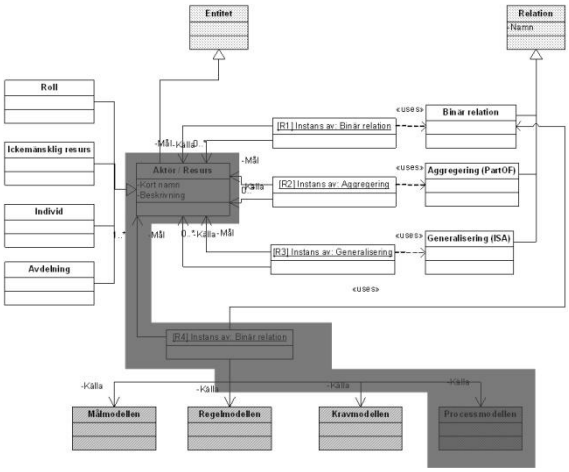
Detta kapitel kommer att ta upp vilken information som går förlorad utifrån den transformation som har beskrivits. Som Figur 7:1 visar, kommer kapitlet även att ta upp vad som behöver kompletteras i det mer formella semi-formella modelleringsspråket UML för att de modeller som transformeras till skall bli kompletta.



Figur 7:1 Arbetsprocess – Saknad och förlorad information

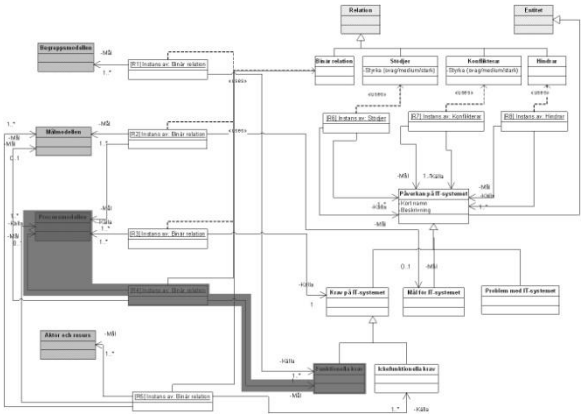
I avsnitt 4.2.3 tog detta arbete upp vilka modelltyper som var lämpliga att transformera till och vilken information i EKD som överensstämmer med informationen i UML och kan transformeras mellan språken. Detta utfördes genom att modellerna i EKD mörkades ned och endast de entiteter och relationer som kunde överföras var ljust. För att få fram vilken information som inte kunnat översättas från EKD till UML bör vi således kunna ta dessa modeller och vända på dem och mörka den delen som kunnat översättas.

I både aktivitetsdiagrammet och use case diagrammet fanns kopplingar till Aktörs- och resursmodellen. Utifrån dessa modeller täcks en stor del av EKD's Aktörs- och resursmodell in i den transformering som detta arbete visar på kan göras. Figur 7:2 illustrerar den bit av EKD's Aktörs- och resursmodell som transformationen täcker in. Som vi kan se är det aktörerna och resursernas relationer mellan varandra och relationer till andra modelltyper som saknas. Detta beror på att UML inte har i sitt grundutförande någon central punkt där modelleraren kan samla alla aktörer eller resurser. Det finns exempelvis ingen koppling mellan de aktörer och resurser som specificeras i use case diagrammet och sekvensdiagrammet. Detta gör att misstolkningar kan ske om inte begreppen är utredda på förhand Willars (1999).



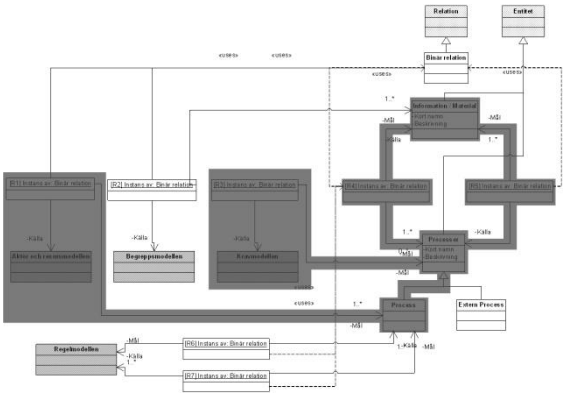
Figur 7:2 Delar av EKD's Aktör och resursmodell som inte transformerats till UML

I Figur 7:3 illustreras de delar utifrån metamodellen för kravmodellen i EKD de delar som inte kunnat transformeras utifrån den transformering detta arbete tar upp. I den modellen täcks funktionella krav och dess koppling till processmodellen eftersom den funktionaliteten kunde transformeras. De interna relationerna mellan kraven och till de övriga modellerna har inte kunnat transformeras till UML. De övriga kraven har heller inte kunnat transformeras i denna transformering. Detta beror på att inte något lämpligt diagram i UML har hittats för att ta hand om dessa krav.



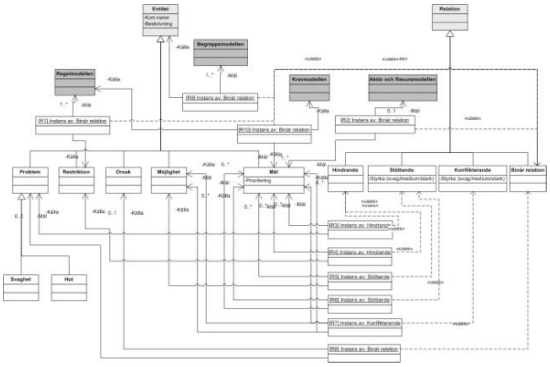
Figur 7:3 Delar av EKD's Kravmodell som inte transformerats till UML

Figur 7:4 illustrerar de delar som inte har kunnat transformeras i detta arbete till UML. Som figuren visar har en stor del av funktionaliteten kunnat fångas med hjälp av UML enligt den transformation som detta arbete föreslår. Figur 7:4 är en kombination av Figur 4:12 och Figur 4:15, den visar alltså både vilka delar use case och aktivitetsdiagram tar upp. Även i denna modell, som i Figur 7:2 och Figur 7:3, saknas relationer till andra modeller.

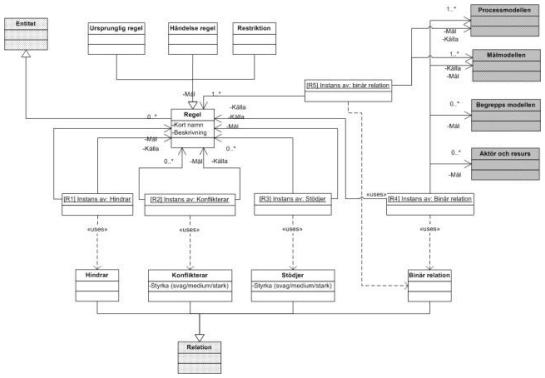


Figur 7:4 Delar av EKD's Processmodell som inte kunnat transformeras till UML

EKD's Målmodell och Regelmodell har inte kunnat mappas alls till UML enligt den transformation som detta arbete utfört. Detta illustreras med att det inte finns några mörkade ytor i Figur 7:5 och Figur 7:6. Det går att peka på flera orsaker men i grund och botten beror detta på att modellernas syfte i grund och botten är att modellera olika saker.

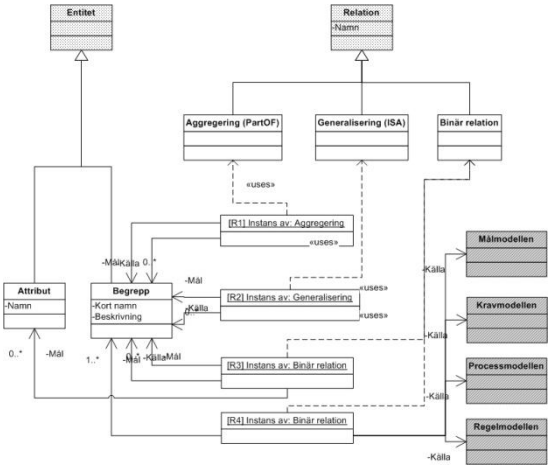


Figur 7:5 Målmodellen i EKD



Figur 7:6 Regelmodellen i EKD

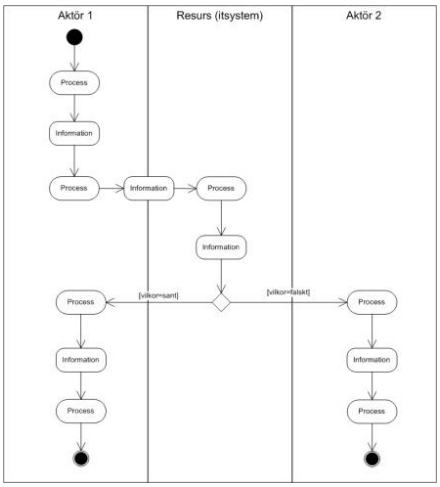
Figur 7:7 illustrerar EKD's Begreppsmodell. Denna modelltyp är en väldigt central modelltyp i EKD som beskriver olika begrepp i de olika modellerna, detta för att alla uttryck ska ha samma betydelse för alla parter som använder modellen. Denna modelltyp ihop med aktör och resursmodellen utgör en grund för begrepp som används inom organisationen. Dessa båda leder till att alla skall kunna förstå vad som menas med ett visst uttryck eller att samma person/aktör/resurs menas när en viss aktör eller resurs diskuteras.



Figur 7:7 Begreppsmodellen i EKD

Har transformationen utförts på det sätt som specificerats i detta arbete är de modellerna som transformerats till i stort sett kompletta. Det som kan behövas ändras eller läggas till är om det finns mer information som inte tagits upp i EKD modellen, alternativt om mer funktionalitet som inte kan visas i EKD skall tillföras.

En sådan funktionalitet finns i UML's Aktivitetsdiagram. Diagrammet har en notation för att visa selektion, detta illustreras i Figur 7:8 där det sker ett val i den mittersta simbanan, Resurs (itsystem), där figuren "◇" visar på en selektion. I detta fall finns en pil åt vänster som uppfylls när villkor är lika med sant och en pil åt höger som uppfylls när villkor är lika med falskt. EKD har ingen notation för att visa på en selektion men tar upp att funktionalitet får tillföras till modellerna vid behov.

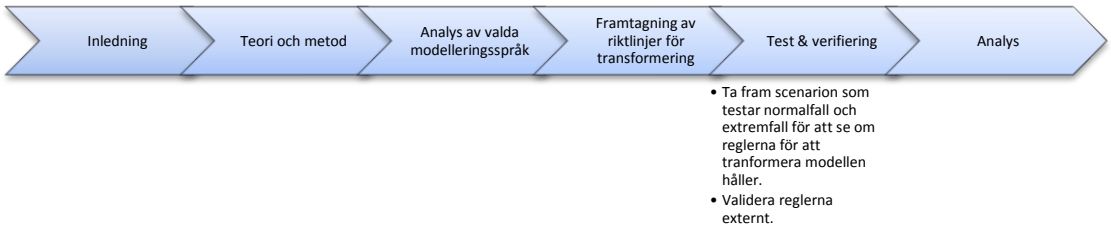


Figur 7:8 Exempel på UML's Aktivitetsdiagram

Även use case har funktionalitet som inte detta arbete hittar en direkt motsvarighet för i EKD. I UML kan ett use case utöka och inkludera andra use cases. Exempelvis kan utökning innebära att ett use case vara en specialisering av ett annat use case. Ett use case kan även inkludera ett annat use case för att var villkor måste uppfyllas för att use caset skall bli sant. Ett exempel på detta kan vara att en användare måste autentisera sig på nytt för att utföra en ändring på ett konto. Autentiseringen kan då ligga som ett use case för att logga in i systemet.

8 Test och verifiering

I detta kapitel skall arbetet testas och verifieras enligt arbetsprocessen i Figur 8:1. En riktig verksamhetsmodellering kommer att tas fram för att verifiera att transformationen fungerar på ett korrekt sätt även mot ett framtaget fall.



Figur 8:1 Arbetsprocess för test och verifiering

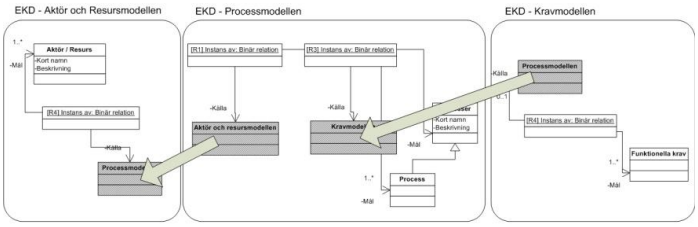
I detta kapitel, test och verifiering, skall de uppsatta riktlinjer som detta arbete utarbetat skall verifieras och testas för att se om de håller. I EKD manualen Bubenko, et al. (2001) tas en verksamhetsmodellering upp. Den utförda modelleringen som är beskrivet i källan bygger på en verklig organisation men fallet är fiktivt. Modelleringen är utförd mot Kungliga Tekniska Högskolans (KTH) huvudbibliotek och går ut på att de har en förgrening vid Electrum i Kista där de behöver anpassa sig efter en ny budget. Denna rapport kommer inte gå in mer på just detta arbete och hur det är utfört utan kommer att titta på modellerna som skapades utifrån det.

De modelltyper som går att utföra transformeringen ifrån är Aktörs- och resursmodellen, Processmodellen och kravmodellen enligt den argumentation som fördes i 4.2.3.

8.1 Mappning till use case

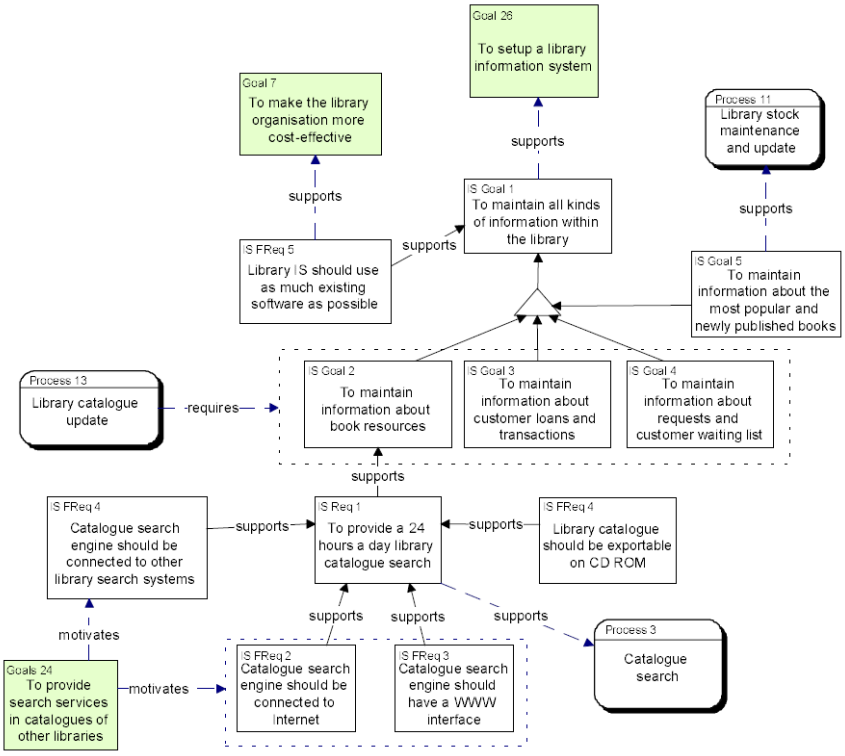
Under kapitel 5.1 går denna rapport igenom hur en mappning till use case diagram från EKD skall utföras. Här under kommer uppsatta process att utföras för att genomföra denna transformering på fallet som beskrivs i stycke 8, ett bibliotek under KTH.

I kapitel 5.1 beskrivs att de modelltyper som används från EKD är Kravmodellen, Processmodellen och Aktörs- och resursmodellen. Dessa modelltyper relaterar till varandra enligt Figur 8:2.



Figur 8:2 Tre utvalda modelltyper i EKD

I Figur 8:3 visas den kravmodell som gjordes för arbetet i EKD modellen. I denna ser vi ett antal funktionella krav, som FReq2 med texten, fritt översatt från engelska, ”Katalogens sökmotor skall vara kopplad till internet” som jag kommer ta som exempel för att transformera till UML’s use case.



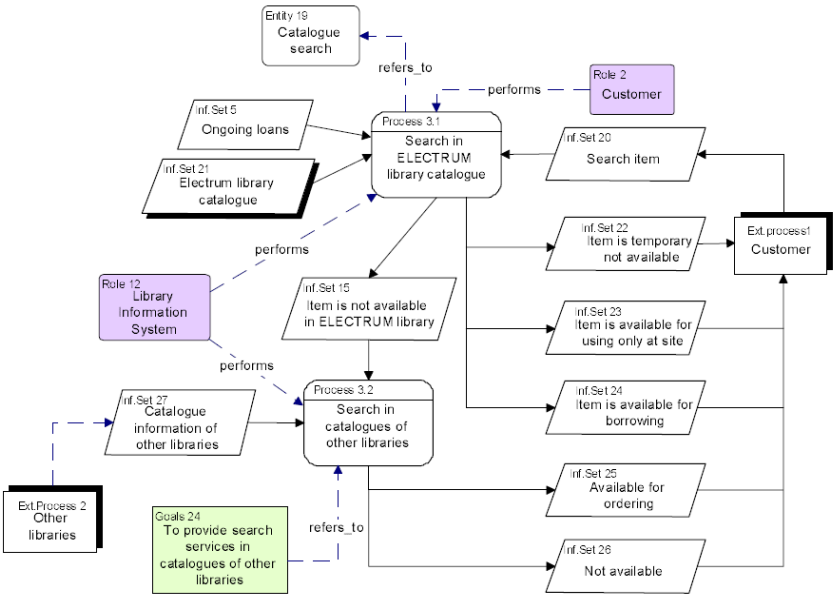
Figur 8:3 Krav[4]

Arbetet tar i kapitel 5.1 upp hur dessa modelltyper skall transformeras. Första punkten är att gruppera funktionella krav. Figur 8:3 har inte många funktionella krav och de hänger ihop på ett tydligt sätt då alla handlar om IT-systemet. Nästa punkt är själva mappningen av det funktionella kravet. Dessa skall mappas mot use case i diagrammet, som Figur 5:3 illustrerar. Efter att denna mappning är utförd ska vi ha ett use case som liknar Figur 8:4.

[4] Modellen används med tillstånd från Bubenko, et al. (2001)

Figur 8:4 transformerat use case

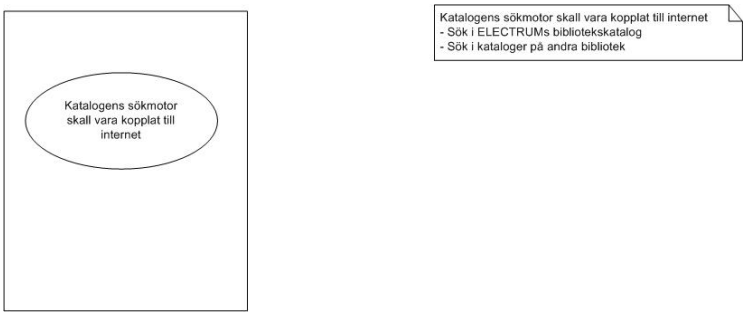
I Figur 8:3 går det att se att FReq2 stödjer ett krav som är kopplad till Process 3 som är en process. Modellen för Process 3 visas i Figur 8:5.



Figur 8:5 Process 3[5]

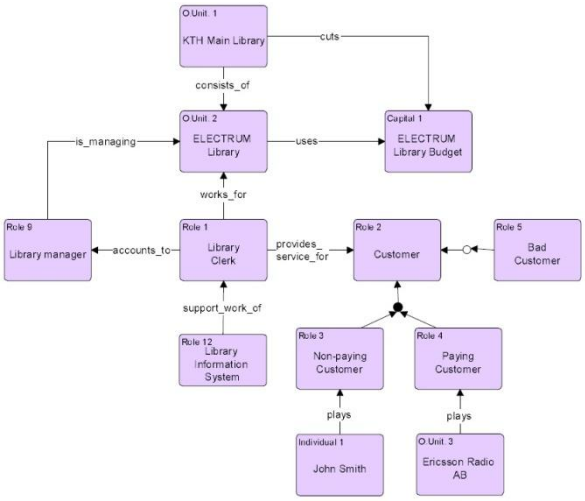
Nästa punkt för transformationen som tas upp i kapitel 5.1 är att mappa processerna som tillhör detta funktionella krav till beskrivningen, vilket illustreras i Figur 5:4. Resultatet av detta i det här specifika testfallet blir som i Figur 8:6.

[5] Modellen används med tillstånd från Bubenko, et al. (2001)



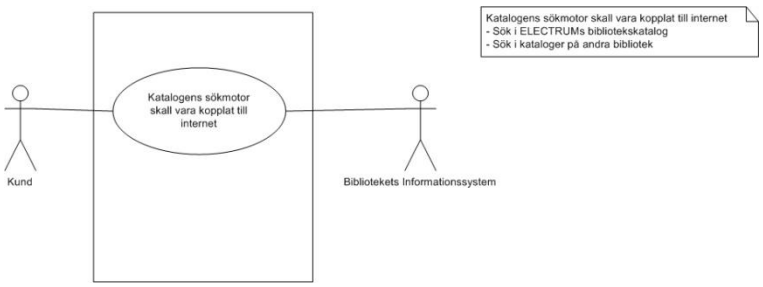
Figur 8:6 transformerat use case och tillhörande processer

Efter mappningen skett till process kan en eller flera aktörer som ansvarar för processen i EKD's processmodell knytas till detta use case. I just denna modell har aktörer satts ut på processmodellen, vilket underlättar denna transformering. Om dessa inte skulle vara utsatta på detta sätt behöver modelleraren gå till EKD's processmodell, som i detta fall illustreras i Figur 8:7, för att ta reda på aktör eller resurs. Dessa aktörer och resurser sätts ut på det sätt som Figur 5:5 illustrerar.



Figur 8:7 Aktörs- och resursmodellen[6]

När mappningen av aktörer och resurser är utsatta för detta funktionella krav skall modellen likna Figur 8:8. I den figuren finns use case, aktörer som är kopplade till det och en förklaring för det specifika use caset.



Figur 8:8 use case med tillhörande aktörer/resurser och en beskrivning.

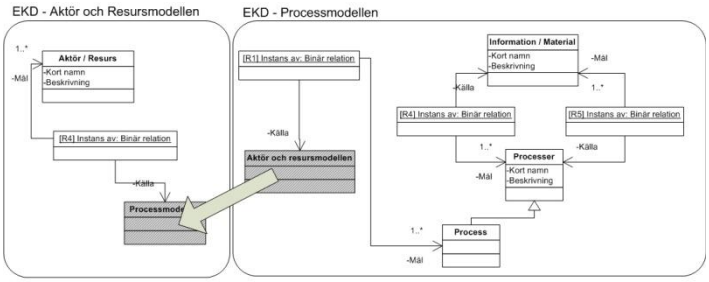
[6] Modellen används med tillstånd från Bubenko, et al. (2001)

När detta är gjort börjar den som transformerar med nästa funktionella krav. Detta krav kan inkludera samma aktörer och peka på samma process. I dessa fall lägger den som transformerar till use caset, skriver till detta use case på beskrivningen ihop med den process som den uppfyller tillsammans med det andra use caset och drar streck till de aktörer som är involverade.

8.2 Mappning till aktivitetsdiagram

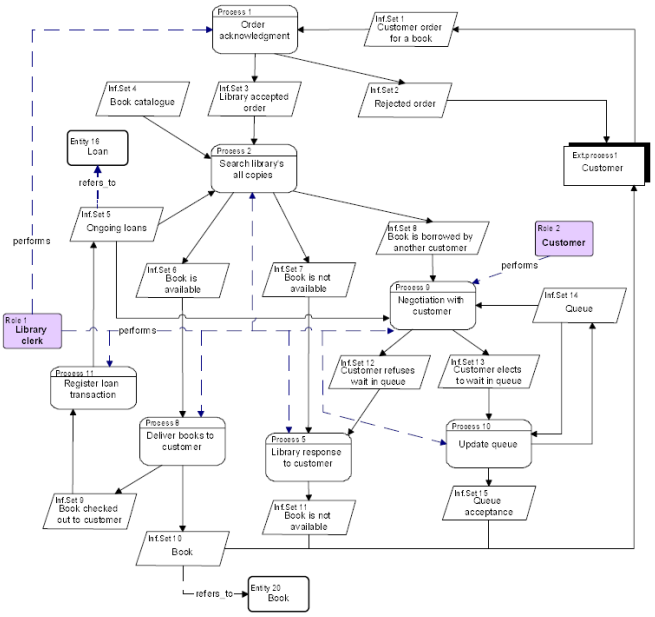
I kapitel 5.2 beskrivs den mappning som görs från EKD till aktivitetsdiagram. Under detta kapitel kommer den process för att överföra från EKD modellerna till aktivitetsdiagram att testas på fallet som tas upp i stycke 8, biblioteket under KTH.

Som kapitel 5.2 förklarar används EKD's aktör och resursmodell och Processmodell för att utföra denna transformation. Dessa modelltyper relaterar till varandra enligt Figur 8:9 där en process har en aktör eller resurs som utför processen.



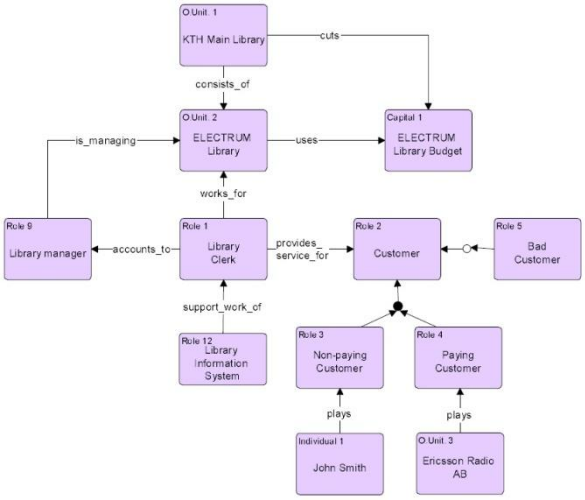
Figur 8:9 EKD relationer

Figur 8:10 visar den processmodell skapad i EKD som används för att testa att den transformation som tagits fram i detta arbete. Det fösta som görs enligt transformationsriktlinjerna i kapitel 5.2 är att gruppera de sammanhängande processerna. I detta fall är det ett bra sammanhängande fall och punkt 2 uppfylls även den då det bara är två aktörer inblandade i detta fall. Antalet processer är högt men eftersom det är få utförare går det att använda utan att det blir svårt att tyda. Sätt sedan ut startpunkt och kommande information och fortsätt med process och upprepa som transformationsriktlinjerna i kapitel 5.2 tar upp. Relationerna mappas som Figur 5:8 visar och informationen som Figur 5:9.



Figur 8:10 EKD Process[7]

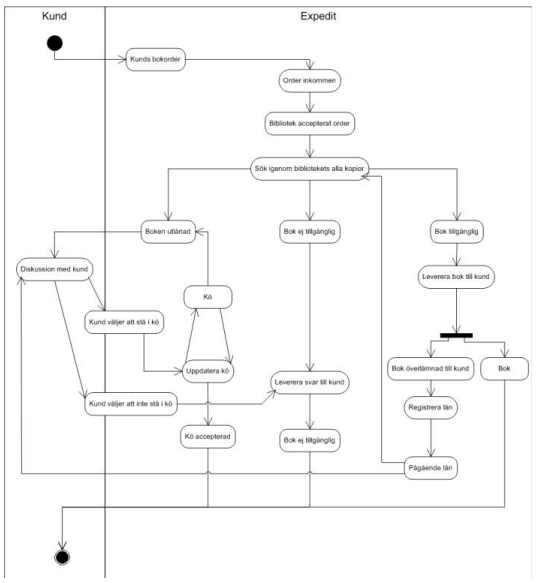
Processmodellen i Figur 8:10 har för enkelhetens skull satt ut vilka aktörer som utför vilka processer vilket underlättar denna transformation. Hur dessa skall relateras mellan varandra är inte sagt utan ofta används listor på de unika namnen på entiteterna i de olika modellerna för att visa på relationen. Om detta hade varit fallet hade vi behövt Aktörs- och resurs modellen för att jämföra nyckeln mot. Denna visas i Figur 8:11. Hur denna mappning går till förklaras i Figur 5:10.



Figur 8:11 EKD Aktör och resurs[8]

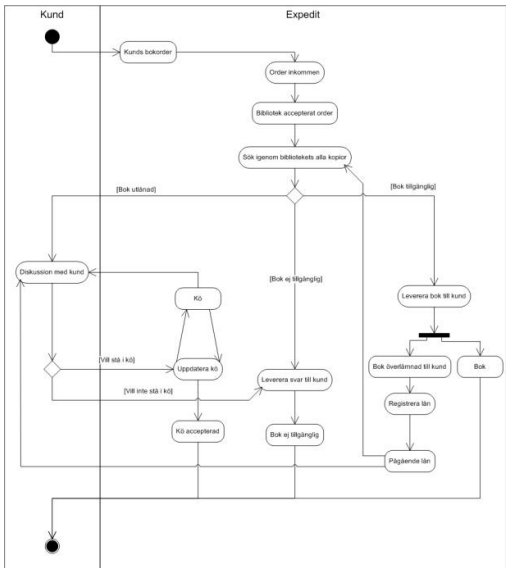
[7] Modellen används med tillstånd från Bubenko, et al. (2001)
[8] Modellen används med tillstånd från Bubenko, et al. (2001)

När tranformation skett utifrån dessa riktlinjer har vi en modell liknande den i Figur 8:12. Vi kan då se att processen börjar med att gå från kund med en information om ordern och följer samma förlopp som EKD modellen. Det finns dock en notation i Figur 8:12 som inte är helt korrekt enligt UML's standard. I figuren, vid ”Sök igenom bibliotekets alla kopior” och ”Diskussion med kund” finns flera alternativa vägar. I UML finns en notation för dessa val.



Figur 8:12 Transformerad modell från EKD till UML

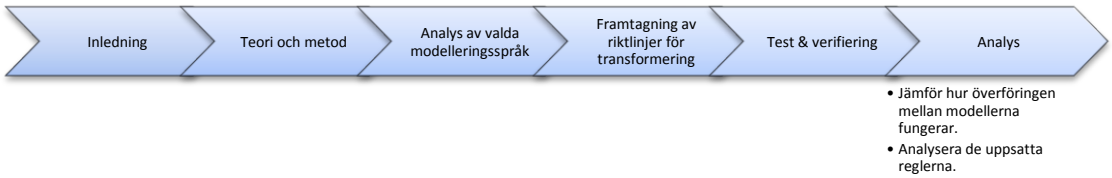
Notationen för alternativ i UML är symbolen ”◇”. På förhand går det inte att se om det är ett alternativ eller olika informationer som skapas av en process i EKD utan detta är något som får utläsas av modellen. De alternativen översätts från de informationerna som var alternativen i EKD modellen eftersom dessa är val av vilken information som skapas. För att upprätthålla standarden för alternativ i UML får vi alltså ändra dessa alternativ som finns i Figur 8:12 enligt Figur 8:13.



Figur 8:13 Transformerad modell med korrekt notation för alternativ

9 Analys och diskussion

Under detta kapitel kommer en analys och diskussion ske omkring hur riktlinjerna och överföringen mellan modelleringsspråken fungerat. I Figur 9:1 visas vilken fas arbetsprocessen i denna rapport befinner sig.



Figur 9:1 Arbetsprocess i analys och diskussion

Detta arbete visar på att det finns möjliga sätt att transformera från det mindre semi-formella språket EKD till två modelltyper av det mer formella semi-formella modelleringsspråket UML. Arbetet talar även om att det går att utföra mer omfattande transformationer om modellerna skulle tillåtas att användas för annat bruk. För att göra detta finns det litteratur, men detta var inte arbetets mål. Målet för detta arbete var att undersöka hur det går att gå över från ett mindre formellt semi-formellt modelleringsspråk till ett mer formellt semi-formellt modelleringsspråk i syfte av att komma närmare en specifikation av ett IT-system. Det vill säga att överföra alla modelltyper med all funktionalitet var inte syftet med denna rapport.

Transformationen som utförts i detta arbete är till aktivitetsdiagram och use case, två av UML's modelltyper. Alla modelltyper kan inte användas, och varför är uppenbart. De olika modelleringsspråken har olika syften. Det ena, EKD, har som syfte att modellera en verksamhet medan det andra, UML, har skapats för att modellera objektorienterade IT-system.

Transformationsriktlinjerna som togs fram i kapitel 5 är skapade utifrån de metamodeller för EKD som skapades i kapitel 4.1. Därför bör dessa riktlinjer fungera på i stort sett alla typer av EKD's modeller så länge de är semantiskt och syntaktiskt korrekta. Dock finns det brister i översättningen som arbetet kunde påvisa vid transformering till aktivitetsdiagram i kapitel 8.2. Där visade processmodellen i EKD ett alternativ vilket inte direkt kunde översättas utan fick antas när modellen transformerats. Detta anser jag vara en allvarlig brist i de riktlinjer som satts upp, men det är svårt att hitta en bra lösning att åtgärda problemet om inte en förändring sker i modelleringsspråket EKD, som är det språk som transformeringen utgår ifrån.

Det finns en annan viktig skillnad på de båda modelleringsspråken. EKD har modelltyper där den som modellerar samlar alla aktörer och resurser och en modelltyp där begrepp definieras. I EKD används dessa modeller flitigt och refereras till när en viss aktör, resurs eller begrepp skall användas. Genom att alla modeller refererar till

samma modell när en av dessa skall användas minskar risken för att samma begrepp används för olika ting eller aktörer. Det finns alternativ för att kunna lösa detta med enkla medel i UML om modelleraren väljer att utöka funktionaliteten. Detta kan göras med exempelvis en så kallad ”Data dictionary”, där modelleraren kan specificera aktörer, resurser och begrepp med en tillhörande förklaring och sedan relatera till denna för att peka ut ett specifikt ting. Detta är dock inget som ingår i standard UML och har därför inte tagits upp i detta arbete.

I detta arbete nämns överföring och fångande av krav i UML. I arbetet framgår det att det är funktionella krav som kan fångas. Det går att ställa sig frågan om det inte är på det sättet att alla modeller i ett ”nyläge” egentligen motsvarar någon sorts krav? Även om dessa inte specificeras som krav så är det ändå en funktionalitet som har arbetats fram och skall implementeras. Detta är något som kan forskas vidare på och då kan man ta fram vilka ytterligare krav som då kan fångas upp från verksamhetsmodelleringen.

Det finns olika fördelar med att börja med en verksamhetsmodell, som EKD, när ett IT-system skall byggas. En av dessa fördelar är att modelleraren och de som implementerar IT-systemet får en spårbarhet och kan ta tillvara på de fördelar som finns med att ha en verksamhetsmodell som grund. I fallet EKD till UML, som detta arbete tar upp, kan modelleraren som arbetar med transformeringen och de kommande UML modellerna få en spårbarhet genom att transformera på det sätt som detta arbete tar upp. Modelleraren kan även i kommande modeller använda sig av de befintliga modelltyper som finns i verksamhetsmodellen, exempelvis: Begreppsmodellen och Aktörs- och resursmodellen. Dessa kan användas för att peka ut ett visst ting för att alla ska få samma tolkning på detta ting.

Rational Unified Process är en utvecklingsprocess som är vanlig inom systemutveckling och är starkt sammanknuten med UML enligt Booch, et al. (2007). Booch, et al. (2007) tar upp att utvecklingen av artefakter startar med use case. Dessa används för att bygga vidare systemet på. Booch, et al. (2008) skriver om aktivitetsdiagram och att dessa oftast används för att illustrera och utveckla use casen ytterligare. Vi kan se i den transformation som gjorts i detta arbete att detta stämmer bra. Detta för att use casen som skapas använder sig av EKD’s kravmodell som i sin tur går genom Processmodellen till Aktörs- och resursmodellen. När Aktivitetsdiagrammet skulle skapas använde oss av Processmodellen och Aktörs- och resursmodellen. Eftersom de båda använder sig av processmodellen går det att säga att de möter varandra i processmodellen. Aktivitetsdiagrammet kunde utveckla och visa mer av hur processen flödade, i vilken ordning, vem som utförde och vilken information som konsumerades och skapades. Eftersom det faller sig på detta sätt vill jag påstå att dessa artefakter är en god inledning på att bygga ett nytt IT-system på och min slutsats blir: Att transformera från EKD till UML genom att använda dessa transformationsriktlinjer har utvecklaren en bra grund att stå på för att bygga systemet om detta görs med en metod liknande Rational Unified Process som är use case drivet.

Studiens frågeställning, ”Vilka riktlinjer kan formuleras för att med minsta möjliga kompletteringsbehov transformera konceptuella modeller av typen EKD till modeller i UML?” anser jag vara besvarad genom att transformeringsriktlinjer har designats och konstruerats som arbetet tar upp i kapitel 5. Detta för att gå mellan modelleringsspråken UML och EKD. De kompletteringsbehov som efterfrågas har analyserats och dessa behov tas upp i kapitel 7.

9.1 Utvärdering av arbetet

Arbetsprocessen för att undersöka detta problem anser jag har fungerat bra. Arbetsprocessen har haft en grund i metodansatserna för att utföra arbetet och vilka metoder som används i vilken del har förklarats i metodavsnittet. Denna process har följts genom arbetets gång och även metoderna för att utföra de olika delarna i arbetsprocessen. Att arbeta utefter de utvalda metoderna gjorde att jag på ett strukturerat sätt kunde uppnå mitt mål, att undersöka transformation mellan två modelleringsspråk av olika formalitet och skapa transformeringsriktlinjerna. Arbetet har drivits av både designansatsen och dokument- och litteraturstudier. Det var mer av dokument- och litteraturstudier i början av arbetet för att få en bakgrund och underlag till modelleringsspråken. Senare i arbetet blev det mer konstruktion när metamodellerna och framförallt transformeringsriktlinjerna designades och konstruerades.

I metoden i kapitel 3 tog detta arbete upp att det finns ett antal riktlinjer för designansatsen Hevner, et al. (2004) och hur dessa var tänkt att uppfyllas. I Tabell 9:1 har detta arbete på nytt tagit upp dessa riktlinjer och utvärderat hur riktlinjerna för designansatsen verkligen har uppfyllts. Arbetet i sig hade kunnat få en högre precision genom att ha arbetat med UML’s metamodeller och få möjligheten att mappa ihop dessa med EKD’s metamodeller för att få en tydlig och precis mappning. För att jag skulle kunna åstadkomma detta, som arbetet tar upp i kapitel 4.2.2 hade dessa metamodeller behövt omarbetats till en modell att utföra mappningen mot.

Dokument- och litteraturstudien har varit en stor del av arbetet. Den har, som jag tog upp tidigare, legat till grund för studierna av de olika modelleringsspråken. I metod togs även upp sökord upp. Det är svårt att titta just kritiskt på denna punkt. Jag anser att de källor jag har är relevanta även om vissa av dem är lite äldre. Området att transformera från ett modelleringsspråk till ett annat modelleringsspråk anser jag ändå vara ett aktuellt ämne. Exempelvis har Eclipse[9] olika projekt inom området. Något som det hade gått att leta ytterligare på är inom området MDA som skulle kunna vara fortsättningen på detta arbete. I ett sådant arbete hade det varit intressant att undersöka vilka möjligheter det finns att driva ett MDA projekt utifrån de förutsättningar modelleraren har fått utifrån denna rapport. Jag anser dock att jag fått nog med information för att utföra denna typ av arbete som gav svar på min frågeställning. Det finns ingen annan forskning som har gjorts mellan dessa modelleringsspråk som jag har kunnat hitta.

[9] http://www.eclipse.org/modeling/

Validering är en viktig del i detta arbete och i designansatsen. Arbetet är validerat mot ett uppsatt fall från EKD manualen. Fördelen med att använda sig av detta fall var att jag med säkerhet kunde få en modell som var syntaktiskt och semantiskt korrekt. Detta gjorde att jag kunde se om det fanns några och hur stora felmarginaler det fanns vid ett korrekt framställt fall. Det hade varit intressant att testa dessa riktlinjer på fler fall tagna från verkligheten men på grund av tidsbrist för detta arbete har inte detta hunnits med. Därför går det att förbättra detta arbete mot de uppsatta riktlinjerna som finns genom att göra en större validering av dessa. Att göra det hade även varit i linje med designansatsens riktlinjer.

Designansatsens riktlinje	**Hur detta arbete uppfyller riktlinjen**
En designansats måste skapa någon form av artefakt.	Detta arbete skapar riktlinjer för transformation mellan två EKD och UML för att producera artefakter det mer formella modelleringsspråket.
Målet med arbetet i designansatsen är relevant för problemet.	Genom att använda mig av en designansats i detta arbete och skapa riktlinjer för transformationen kunde frågeställningen besvaras. Designansatsen var en väg att nå målet.
Artefakten, kvalitén och effektiviteten skall vara noga demonstrerade och validerade med kända metoder.	Artefakten har både demonstreras och valideras i detta arbete. Både genom att visa hur modeller fungerar, innehåller för information och hur de kan mappas mot varandra. Arbetet har även använt sig av dokument och litteraturstudier för att samla information för att skapa transformationsriktlinjerna.
Den skall tillhandahålla klara och verifierbara bidrag inom sitt område.	Detta arbete kan bidra med information vilka delar som kan transformeras mellan språken. Det kan även hjälpa en modellerare som modellerar med EKD att ta ställning till vilken typ av systemutvecklingsprocess som skall ta vid efter att verksamhetsmodelleringen är gjord. Det kan även vara ett underlag vid ett eventuellt skapande av en automatiserad transformation mellan språken. Arbetet är verifierbart. Verifiering kan exempelvis ske genom att utöva de framtagna transformeringsriktlinjerna på framtagna verksamhetsmodelleringar i EKD.
Arbetet, både tillämpning och utvärdering, i en designansats skall vila på beprövade metoder.	Arbetet är grundat på vetenskapliga metoder. Arbetet har skapat ett ”best practise” för transformering kan utföras mellan de olika modelleringsspråken. Tillämpningen testas i steget för test och verifiering mot ett fördefenierat fall som inte är anpassat för just detta arbete. Utvärderingen av tillämpningen görs i detta kapitel, diskussion.
För att nå önskat resultat i sökandet efter nya effektiva artefakter krävs att tillgängliga medel brukas utan att bryta de regler som finns i problemets domän.	Arbetet har använt alla tillgängliga medel som funnits och varit inom ramen för att kunna hinna med att analyseras. Medel så som rapporter, litteratur och vetenskapliga metoder för att samla in informationen har använts som utlovat i Metod.
Artefakten från arbetet gjort på designansatsen skall presenteras för både ledning och tekniskt kunniga intressenter.	De riktlinjer som har skapats har presenterats under en opponering. När materialet är godkänt av examinator kommer materialet även att läggas ut på Biblioteket för Högskolan i Skövde. På så sätt är det öppet för alla att titta på. Som rapporten tog upp under metod finns ingen annan högre ledning eller personal att presentera detta arbete för.

Tabell 9:1Uppfyllande av designansatsens riktlinjer.

Det finns vissa samhälleliga aspekter på detta arbete. Det finns olika tänkbara scenarior där dessa uppsatta transformeringsriktlinjer kan användas. En modellerare kan exempelvis använda sig dessa regler för att veta att han eller hon får en översättning mellan de olika språken som är framtagen på ett korrekt sätt. Om ett projekt är tänkt att skapa ett IT-system kan modelleraren välja en utvecklingscykel utefter den information som denna har fått genom att studera detta arbete. Genom att i förväg veta detta kan modelleraren med fördel titta närmare på en utvecklingscykel som är use casebaserad då dessa transformeringsriktlinjer skapar use case och aktivitetsdiagram, som vi har fått förklarat kan ses som ett förtydligat use case.

EKD har sina grunder i en metod som kallas för ABC (Bubenko, et al., 2001). Det finns andra metoder på marknaden så som Astrakan som även dessa utgår från denna metod Willars (2008). Dessa uppsatta regler kan vara intressant att se hur de passar ihop med dessa modelleringsspråk. Det kan tänkas vara så att det går att översätta dessa till dessa språk med vissa mindre justeringar och få en översättning mellan dessa modeller med för att kunna lättare övergå till en systemutveckling när verksamhetsmodelleringen är färdigställd. Det skulle även gå att tänka sig att transformeringsriktlinjerna kommer att användas som grund för att utforma en automatiserad översättning mellan dessa två modelleringsspråk. Detta kan dock leda till att felaktiga modeller skapas på grund av helt syntaktiskt korrekta modeller inte skapas om det finns alternativ med i EKD modellen.

Det finns inga direkta etiska aspekter med själva utförandet av detta arbete. Däremot om dessa riktlinjer används i ett verkligt projekt, går det att säkerställa att de aktörer som varit med och utfört verksamhetsmodelleringens åsikter och önskemål följer med till utvecklingsfasen för IT-systemet. Genom att ha möjligheten att kunna påvisa att dessa transformeringsriktlinjer använts kan modelleraren skapa en spårbarhet mellan de olika modelleringsspråken. Aktörerna som var med och skapade verksamhetsmodelleringen kan då se att deras åsikt har tagits i beaktande under utvecklingen. Detta kan i sin tur ge en högre acceptans mot IT-systemet än ett IT-system som är framtaget utan att tagit deras åsikter i beaktning. Aktörerna som var med och tog fram grunden till IT-systemet genom verksamhetsmodellringen får sina åsikter respekterade.

Jag anser att under detta arbete har jag lärt mig mycket och fått en större inblick i modelleringsspråkens användningsområden.

Baumeister, H., Koch, N. & Mandel, L. (1999) Towards a UML extension for hypermedia design. *UML'99 Proceedings of the 2nd international conference on The unified modeling language: beyond the standard* (s 1 – 16), 28-30 Oktober, 1999, Fort Collins, USA

Booch, G., Jacobson, I. & Rumbaugh, J. (2007) *The Unified Software Development Process* Boston: Pearson Education

Booch, G., Jacobson, I. & Rumbaugh, J. (2008) *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)* Boston: Pearson Education

Brown, A.W. (2004) Model driven architecture: Principles and practice. *Softw Syst Model (2004)*, 3, 314–327

Bubenko, J. A., Persson, A. & Stirna, J. (2001) *EKD User Guide*. Stockholm: Dept. of Computer and Systems Science. KTH.

Bubenko, J., Rolland, C., Loucopoulos, P. & DeAntonellis, V. (2002) Facilitating "fuzzy to formal" requirements modeling. *Proceedings of the First International Conference on Requirements Engineering* (s 154 -157), 18-22 April, 1994, Colorado Springs, USA

Eriksson, H-E. & Penker, M. (2000) *Business Modeling with UML: Business Patterns at Work.* New York: John Wiley & Sons, Inc.

He, x., Ma, Z. Shao, W. & Li, G. (2007) A metamodel for the notation of graphical modeling languages. *Computer Software and Applications Conference* (s 219 - 224), 24-27 Juli, 2007, Beijing, Kina

Hevner, A R., March, S T., Park, J & Ram, S. (2004) Design Science in Information Systems Research. *MIS Quarterly Vol. 28, No. 1*, 75-105

Johannesson, P. (2007) The Role of Business Models in Enterprise Modelling. *CONCEPTUAL MODELLING IN INFORMATION SYSTEMS ENGINEERING*, 123-140.

Krogstie, J., Sindre G. & Jørgensen H. (2006) Process models representing knowledge for action: a revised quality framework. *European Journal of Information Systems, 15*, 91–102

Lindland, O.I., Sindre, G. & Sølvberg, A. (1994) Understanding Quality in Conceptual Modeling. *IEEE Software*, 11(2), 42-49.

Martín-Vivaldi, N. & Isacsson, P. (1988) Controlling Your Design through Your Software Process. *COMPUTER SAFETY, RELIABILITY AND SECURITY Lecture Notes in Computer Science, Volume 1516/1998*, 77-88

Norman, P. (2010) *Astrakanmetoden på nolltid.*

Tillgänglig på Internet: http://www.astrakan.se/PageFiles/1683/Astrakanmetoden_PN.pdf [Hämtad: 2011-01-22]

Ruiz, F., van Harmelen, F., Aben, M. & van de Plassche, J. (1994) Evaluating a formal modelling language. *A FUTURE FOR KNOWLEDGE ACQUISITION*, Volume 867/1994, 26-45

Willars, H. (1999) Business Modeller's Checklist: "Dos" and "Don'ts" in Hands-on Practice. I: A G. Nilsson, C. Tolis & C. Nellborn (Red), *Perspectives on Business Modelling. Understanding and Changing Organisations.* (s. 305-323) Berlin: Springer-Verlag.

Willars, H. (2008) Tillgänglig på Internet:

http://www.astrakan.se/Ovrigt/Lararsidor/Hans-Willars/ [Hämtad 2011-06-04]