REGELMODELLERING I PRAKTIKEN

– erfarenheter och resultat från Tempora och modellbaserad kunskapsinhämtning

Sten-Erik Öhlund
Rolf Wohed

TRIAD-PROJEKTET DRIVS GEMENSAMT AV TELIA, POSTEN, STATSKONTORET, ERICSSON OCH SISU, SVENSKA INSTITUTET FÖR SYSTEMUTVECKLING. FÖR VIDARE INFORMATION KONTAKTA SISU, ELECTRUM 212, 164 40 KISTA, TEL 08-752 16 00, FAX 08-752 68 00.

Kort om Modelleringshandboken

Inom TRIAD-projektets ram har parterna, dvs Ericsson, Telia, Posten, Statskontoret och SISU, beslutat sig för att satsa på ett generellt modellspråk för att analysera och beskriva verksamheter i generella konceptuella modeller. Resultatet av denna satsning utgörs av Modelleringshandboken.

Följande personer har deltagit i arbetet:

Agneta Hagberg, Posten GK-Data	Ann Rehbinder, Posten GK-Data
Malte Nordström, Telia Data	Margareta Pettersson, L M Ericsson Data
Claes-Göran Lindström, IT Plan	Hans Willars, SISU

Parterna bidrar successivt till Modelleringshandboken genom att producera separat utgivna avsnitt som ingår i en överordnad gemensam handboksstruktur. Som framgår av nedan är handboken indelad i ett antal block med delvis olika syften och målgrupper. Material som finns framme är markerat med *, övrigt är under arbete eller planerat.
Referenser inne i en text till andra handboksdelar markeras med titel i fet kursiv stil. Referenser till avsnitt i den här handboken markeras med med fet stil.

Handboksstrukturen

Block A: Översikter

Målgrupp: Ni som vill veta vad modellering är för att kunna var med.

N10:1 Modelleringshandboken - översikt *
Grundkunskap för modelleringsdeltagare

Block B: Handledningar

Målgrupp: Ni som har kommit i kontakt med modellering och vill kunna arbeta på egen hand eller leda ett modelleringsarbete.

N10:2 Modelleringsledarens bashandledning *
Modelleringsteknik, fördjupningar
Referensramar, angreppssätt *
Modermodeller *
Informatikövergång
N10:3 Modellering i grupp *
N10:4 Kommunikation*
N10:5 Arbetsgångar (F n begränsad till Verksamhetsanalys för informatikutveckling *)
N10:6 Modelleringsväskan *
N10:9 Regelmodellering i praktiken *

Block C: Teorier, bakgrunder, fördjupningar

Målgrupp: Ni som vill ha djupare kunskap i modellering.

N10:7 Objektorienterad verksamhetsanalys *
N10:8 Basmodeller - introduktion *
N10:10 Business Process Reengineering *
N10:11 Namnsättning i modelleringssammanhang.
N10:12 Tolkning av grafiska modeller *
(Fler teoriavsnitt efter behov och intresse)

Block D: Hjälpmedel för kunskapsspridning

Målgrupp: Ni som vill visa, lära ut och sprida information om modellering.

Informationsmaterial *
Kursmaterial *
Lärarhandledning
Praktikfall *

Rapporterna beställs från
SISU, Electrum 212, 164 40 Kista, Fax 08-752 68 00
Rapporterna är endast tillgängliga för Triad-parterna och är avgiftsfri

Innehåll

Förord

Denna rapport syftar till att ge handledning i regelmodellering i samband med systemutveckling i de tidiga faserna.

Rapporten vänder sig till systemutvecklare samt verksamhetsanalytiker som har erfarenhet av modellering och som är intresserade av hur man ska hantera regler i de tidiga faserna av systemutvecklingen.

Vi kommer att rapportera erfarenheter från två forsknings- och utvecklingsprojekt som SISU deltagit i. Det ena projektet heter Tempora och är ett ESPRIT-projekt som pågått i 5 år och som nyligen avslutats. Projektet har behandlat tidsaspekter vid utvecklingen av informationssystem. Regler har även varit en grundläggande komponent i projektet. Under arbetets gång har vi utvecklat och vunnit erfarenheter om regelmodellering som vi vill dela med oss av i den här rapporten. Vi hänvisar i övrigt till den metodhandbok som Tempora-projektet utarbetat samt till tidigare TRIAD-rapporter om Tempora.

Vi kommer också att rapportera från ett annat projekt, Modellbaserad Kunskapsinhämtning, MBKI. I detta projekt som bedrivits tillsammans med Infologics AB, har man studerat hur modellering för kunskapsinhämtning kan användas när man konstruerar expertsystem.

Vi tycker att de erfarenheter vi vunnit i ett praktikfall även är intressanta för dem som ska formulera och analysera verksamhetsregler. I det senare projektet handlar det främst om regler som är relaterade till en viss fackkunskap, vilket ställer speciella krav på analysarbetet. Dessa erfarenheter redovisas i avsnittet *Sårbarhetskarta vid* SGU – *ett praktikfall i regelmodellering.*

Vi vill betona, precis som undertiteln till denna rapport antyder, att vi först och främst redovisar erfarenheter och resultat som kan vara användbara för de som vill skaffa sig kunskaper i hur man praktiskt kan bedriva regelmodellering. Rapporten utgör alltså ingen sammanhängande metodhandbok. Vi har ändå strävat efter att försöka täcka de viktigaste frågorna i samband med regelmodellering.

Rapporten börjar med en beskrivning av den problembakgrund som vi anser motiverar en utveckling av regelmodellering inom system- och verksamhetsutveckling. Regelmodellering är ett viktigt men eftersatt område!

Därefter gör vi ett försök att definiera vad vi menar med regler och verksamhetsregler. Begrepp som regel eller verksamhetsregel är inte entydiga och ofta används de på ett ogenomtänkt sätt. Ofta behandlas inte ens frågan om hur verksamhetsreglerna ska tillämpas och i vilket förhållande de står till reglerna i informationssystemet. Vi förordar en distinktion mellan verksamhetsregler och regler i informationssystemet.

För att bedriva någon form av beskrivning måste vi ha ett språk. Vi går i efterföljande kapitel igenom Temporas modelleringsspråk och speciellt de olika regelspråken och visar hur man på olika sätt kan representera regler i dessa språk. För att tillgodogöra sig merparten av de erfarenheter som vi senare ska beskriva krävs en generell förståelse av dessa modelleringsspråk. Om du inte är van vid modelleringsspråk och syntax, råder vi dig att hoppa över de tekniska delarna och skaffa dig en ungefärlig förståelse av de olika språken.

Efter detta kapitel följer i resten av rapporten en redovisning av olika angreppssätt vid regelmodellering och de olika erfarenheter som vi samlat på oss.

Det första kapitlet i denna del beskriver vi hur vi använt beslutstabeller i Tempora och hur de ska kopplas till Temporas formella regelspråk ERL (External Rule Language).

Efter erfarenheterna från Tempora redovisar vi även ett praktikfall från projektet MBKI som beskriver hur beslutstabellerna kan användas för att formulera ett antal regler inom ett visst fackområde. I detta fall handlar det om geologisk kunskap och hur man framställer tematiska kartor från olika grundkartor.

Till slut har vi tagit med ett kapitel som är hämtat från Temporas metodhandbok och som ger praktiska råd vid formaliserandet av regler i ERL. I bilagan finns även ERL:s syntax beskriven.

1 Modellering av verksamhetsregler

I det här kapitlet ger vi en kort introduktion till varför vi anser att regelmodellering är ett viktigt område. Vi gör även ett försök att definiera vad vi menar med verksamhetsregler. Verksamhetsregler uppträder i olika sammanhang och vi har gjort en enkel indelning av olika ursprung för regler. Tillämpningen av verksamhetsregler är en komplicerad fråga, speciellt i samband med systemutveckling. Vi har kortfattat beskrivit varför vi anser att det är viktigt att göra en klar distinktion mellan verksamhetsregler och regler i informationssystemet. Vi tar upp några allmänna riktlinjer för hur man ska hantera formaliseringen av verksamhetsregler i ett informationssystem.

1.1 Varför är regelmodellering viktigt?

Ofta innehåller inte metoder för systemutveckling ett systematiskt sätt att hantera verksamhetsregler. Verksamhetens olika regler institutionaliseras och begravs i den stora mängden applikationskod. Detta innebär stora svårigheter att senare underhålla systemen men framför allt svårigheter att snabbt anpassa informationssystemen efter ändrade förutsättningar för verksamheten. Vi vet att dessa förutsättningar ständigt ändras och att det sker i ett allt snabbare tempo. Verksamhetsregler som varit oförändrade sedan årtionden utsätts idag för starka revideringar från omvärlden eller som ett resultat av en inre revideringsprocess. Informationssystemen och verksamheten glider alltmer isär. De gamla systemen med sina inbyggda regler blir till begränsningar.

Den kunskapsmängd som finns i systemen blir svår att utnyttja då kunskapen finns begravd under lager av kod. Oftast består lösningen av att skrota de gamla systemen för att det blir för kostsamt och mödosamt att rekonstruera och underhålla dem. Men samma problem uppstår vid utvecklingen av nya system om man inte ser till att separera viktiga verksamhetsregler från massan av applikationskod.

Ofta byggs också verksamhetens regler aningslöst in i systemen. Reglerna tenderar då att institutionaliseras och bli en del av en trögstyrd byråkrati. Gapet mellan önskat läge och faktiskt läge, mellan formell organisation och informell organisation, ignoreras ofta. Det blir viktigt att skapa en strategi för hur informationssystemen ska samverka med verksamheten. I det sammanhanget uppstår behovet av analys, granskning, formalisering samt tillämpning[1] av verksamhetsregler.

---

[1] Vi kan även tänka oss att vissa verksamhetsregler inte längre ska tillämpas i verksamheten. Sådana verksamhetsregler kan ligga kvar i de gamla systemen.

Inom forskningsvärlden finns ett förnyat intresse för tillämpning av regler. Tempora är ett exempel. Men även erfarenheter från det juridiska fältet och AI, när man använder s k deontisk logik[1] för att hantera och beskriva gapet mellan önskat läge (lagarna följs och tillämpas på ett visst sätt) och faktiskt läge (lagarna följs i vissa fall inte), kan ge vissa bidrag till analysen av verksamhetsregler[2]. Vi kommer att ge exempel på detta i inledningen. Klart är att dessa frågor måste behandlas i informationssystemen på något sätt.

Vi saknar stöd för formulering, formalisering och problemlösning i samband med vad vi vill kalla regelmodellering. Vi tror att ett sätt att hantera problemet med de begravda verksamhetsreglerna är att hantera regler som distinkta objekt i beskrivningen av informationssystemen. Vi tror också att man kan bidra till att hantera arvsproblematiken bättre genom att se till att viktiga verksamhetsregler finns representerade på ett enhetligt och åtkomligt sätt. Vi ser även möjligheter att förbättra återanvändning av olika slags verksamhetsregler, verksamhetsbegrepp, och verksamhetsfunktioner.

I gängse metoder behandlas inte några regler i de tidiga faserna, vilket leder till att de byggs in osynligt i koden. Genom att behandla reglerna tidigt kan man spara mycket arbete och tid genom att rätta fel på ett tidigt stadium. Ofta förlorar man möjligheten att spåra felen bland krav, mål och regler i det slutliga systemet.

Man ignorerar regler på verksamhetsnivån där man borde vara som mest intresserad av att hantera dem. I stället överlåter man åt programmerarna att hitta på lämpliga regler.

Detta är något om de bakgrundsfaktorer som talar för att regelmodellering är ett viktigt område.

Vad är då regelmodellering i detta sammanhang?

Ett viktigt instrument för att komma tillrätta med de oregerliga regelmängderna är att beskriva dem på ett begripligt och överskådligt sätt. Förutom överskådliga och begripliga formuleringar av verksamhetsreglerna kan också en striktare formulering av regler med ett formellt språk bidra till att klargöra oklarheter, reda ut motstridigheter och få en uppfattning om täckningsgrad och om hur felhantering ska gå till.

Själva processen att formulera verksamhetsregler kan bedrivas som en samförståndsprocess där man strävar efter att skapa en överenskommelse och förståelse för valda regler.

---

[1] Teorin om de logiska relationerna mellan satser som uttrycker förpliktelser, påbud, förebild, tillåtelser och liknande. *Deon* kommer från grekiskan och betyder det som är nödvändigt, plikt.

[2] Se t ex Deontic Logic in Computer Science 1993, R.J. Wieringa och J.-J.CH. Meyer (Eds) [J-J Ch. Meyer, 1993 (2)]

En del verksamhetsregler kan relateras till verksamhetens mål. Det kan vara önskvärt för att skapa en relation mellan önskade lägen och regler (motivation) men också för att kritiskt granska reglernas relevans för verksamhetens mål. Kanske står de i motsättning till målen, kanske ska de revideras eller förkastas, eller kanske ska de tillämpas på ett annat sätt som *rekommendationer* och inte som *obligatorium*.

Det är viktigt att utreda dessa förhållanden innan man bygger informationssystem som har inbyggda regler som i många fall kommer att *tvingas på* verksamheten. Det kan inte överlåtas åt någon applikationsbyggare som har liten kunskap om verksamheten i stort och dess förutsättningar och omgivning. Detta är strategiska beslut för företagsledningen. Vi måste därför utveckla metoder att granska, formulera och översiktligt beskriva de av verksamhetens regler som är av vitalt intresse för verksamheten.

1.2 Vad menar vi med verksamhetsregler?

Avsikten är inte att behandla begreppet regler i allmänhet. Vi kommer i kontakt med verksamhetsregler i samband med systemutveckling. Det kan vara regler som ska formaliseras i ett system, eller påverka dess utformning, användning etc. Det är den speciella problematik som uppstår vid utveckling av informationssystem som intresserar oss i denna rapport – inte i första hand analys av verksamhetsregler i allmänhet och dess koppling till verksamhetens idé, mål eller etik m m.

Dock kan vi inte undvika denna problematik helt och hållet och det kan därför vara bra att utgå från ett mer allmänt synsätt på regler.

Det verkar emellertid vara svårt att ge en entydig definition av verksamhetsregler, eftersom termen ofta används ogenomtänkt och i en mängd olika sammanhang. Vi har valt att utgå från en allmän definition i ett lexikon, *Filosofi Lexikonet*.

Regel kommer av latinet *regula* som betyder rättesnöre. En regel kan betyda åtminstone fyra olika saker:

1 En handlingsföreskrift, en norm.
2 En form av standard inom en verksamhet eller disciplin som i ett visst avseende är korrekt eller inkorrekt tillvägagångssätt eller förhållande, t ex grammatiska regler, slutledningsregler, tidsordning, rätta proportioner m m. Regler som beskriver mer eller mindre nödvändiga samband inom ett fackområde, en viss typ av verksamhet (konstruktion av integrerade kretsar), eller en vetenskap.
3 En instruktion eller ett led i en metod som har för avsikt att uppnå ett bestämt syfte.
4 En speciellt lättfattlig generalisering som inte nödvändigtvis saknar undantag, en s k tumregel.

Vi har inte begränsat oss till någon betydelse utan kan tänka oss att alla dessa typer av regler kan förekomma i samband med utveckling av informationssystem. Vi kan gruppera betydelsen av regler enligt följande med exempel i den högra kolumnen:

Regler som avser handlingsföreskrifter eller norm i en verksamhet.	Utbetalning av lön ska ske senast den 25:e i varje månad (handlingsregel) Arbetstiden är 40 timmar per vecka. (restriktion eller villkor)
Regler som avser en standard för korrekt tillvägagångssätt inom verksamhet eller disciplin.	Grunden på ett hus måste byggas innan man reser taket
Regler som visar hur man följer vissa steg för att uppnå ett viss syfte.	Monteringsanvisning för att sätta ihop en IKEA-bokhylla
Regler som är ungefärliga generaliseringar – tumregler	Ta lika många mått kaffe som koppar vatten

En regel kan avse olika typer av verksamheter. Vi kan avse affärsverksamhet, offentlig verksamhet, problemlösningsverksamhet, produktionsverksamhet, produktutveckling, (inklusive systemutveckling), forskningsverksamhet, rutinarbete, etc. I kapitel 5, Framtagning av sårbarhetskarta vid SGU, har vi ett exempel på regler som avser en problemlösning. Regler som avser handlingsföreskrifter eller normer i verksamhet förekommer ofta vid utveckling av administrativa informationssystem. Vi har gett exempel på en handlingsregel som förändrar ett tillstånd (d v s talar om hur och när något ska utföras) och en restriktion (eller villkor) som föreskriver ett legalt tillstånd (d v s hur något ska vara eller får vara). Kunskapsbaserade system innehåller ofta tumregler och regler som avser en viss fackverksamhet eller disciplin.

En verksamhetsregel kan alltså betyda många olika saker. Det kan vara värdefullt att klassificera regler efter deras användning. Ett sätt att göra det är att använda ovanstående klassificering i tabellen som innehåller vad en regel beskriver och kombinera det med vilken typ av verksamhet som regeln används för.

Vi har gjort ett försök att definiera vad en verksamhetsregel kan vara. Men hur vet man att man träffat på en regel? Svaret är inte enkelt. Ofta är det ju så att en mängd regler inte alls beskrivs med ett regelspråk. Regler kan mycket väl och på ett mer praktiskt och smidigt sätt beskrivas med hjälp av en tabell. En del regler kan också beskrivas med konceptuella modeller. Varifrån kommer då en regel egentligen?

1.3 Regler härstammar från olika källor

För att identifiera verksamhetsregler kan det vara till hjälp att veta vilka olika källor som kan finnas. Vi har gjort en enkel klassificering med en kort beskrivning som kan vara ett hjälpmedel vid regelmodellering. Det kan också vara viktigt att härleda en regels ursprung för att kritiskt kunna granska den[1]. Det kan vara så att regeln är ett uttryck för en viss norm som inte nödvändigtvis måste följas för att uppnå ett visst mål. Vi antar t ex ofta att en metod föreskriver en sekventiell arbetsgång, något vi i dag kraftigt ifrågasätter eftersom det bidrar till längre ledtider och kvalitetsbrister. Det kan även vara så att regler är överflödiga och saknar existensberättigande.

Metod

I metoder finns det ofta en uppsättning regler för hur ett visst problem ska lösas, vilka sekvenser av aktiviteter som ska genomlöpas etc. Dessa regler återfinns ofta i metodhandböcker. Ibland är de inte nedskrivna utan finns oformulerade i huvudet på experter med lång erfarenhet. Reglerna kan endera vara av normativ karaktär eller också utgå från metodens förutsättningar eller någon annan regel i metoden.

Disciplin

Inom en viss typ av disciplin eller fackområde som t ex medicin, programmering etc, finns det en mängd skrivna eller oskrivna regler som både kan vara av normativ karaktär och härledas från disciplinens förutsättningar.

Naturlagar

Ofta antar vi naturlagarna som givna. Men ibland måste de byggas in i informationssystemen som restriktioner, som t ex att en person enbart kan vara en man eller en kvinna, död eller levande etc.

Verksamhetsidé – mål

En verksamhets övergripande mål eller affärsidé kan konkretiseras i ett antal regler som beskriver hur målen ska konkretiseras. Det kan handla om normer och handlingsföreskrifter, tumregler m m.

Avtal

Regler kan härstamma från avtal som sluts mellan parter. I vissa fall kan dessa regler suspendera eller vara överordnade de som stipuleras i lagar.

---

[1] Det kan vara viktigt ibland att reda ut en regels stamtavla. I Tempora har vi utvecklat en teknik för att göra det (se kapitlet om verksamhetsmål och regler).

Föreskrifter
En del regler är inte obligatoriska utan kan ses som föreskrifter eller rekommendationer. De kan ändå ha en stor betydelse genom att de bakom sig har någon ekonomisk sanktion eller regel som gör att företag ofta följer en föreskrift eller rekommendation. Vi kan ta exemplet med det statliga traktamentet.

Lagar
I många fall begränsas en verksamhet av det lagrum som finns. Företagen definierar själva hur de ska följa lagarna. En verksamhetsregel kan helt enkelt formuleras som "lagarna i landet X ska följas". Detta är inte något entydigt eftersom det kan finnas många tolkningar av lagarna och deras tillämpning. Ett antal regler kan alltså vara resultat av en sådan tolkning och tillämpning.

Policy
En policy är en uppsättning regler som syftar till att uppnå ett visst mål. Ofta talar man om en personalpolicy, kundpolicy, kvalitetspolicy, säkerhetspolicy etc.

Etik och moral
Det är inte alltid som man måste vara överens om motiven bakom en viss regel eller lag. Vi kan ta ett exempel nyligen från DN:s kultursida. Moralfilosofen Torbjörn Tennsjö protesterade i DN, 22/11 -93, högljutt mot det nya förslaget om en ny läroplan som ska grunda sig på en kristen etik och västerländsk humanism. Han argumenterar för att lagarna inte kräver att alla är överens om motiven för dem men ändå acceptera att följa dem.

Icke desto mindre kan man tänka sig att uttrycka bevekelsegrunderna bakom vissa regler och vilja att alla berörda ska acceptera de etiska eller moraliska grundvalarna. Man kan då säga att reglerna har en etisk eller moralisk härstamning.

Normsystem
Ett normsystem behöver inte enligt vad som sagts ovan vara kopplat till samförstånd om en viss etik eller moral. Ett normsystem är den samling regler (oskrivna eller skrivna) som bestämmer vad som är ett accepterat beteende. Det kan t ex gälla en sak som att komma i tid till arbetet, till möten, klädsel, allmänt uppträdande etc.

Sunt förnuft
Många oskrivna regler i en verksamhet härrör från s k sunt förnuft eller vardagskunskap. Dessa regler är ofta mycket svåra att formalisera.

1.4 Tillämpning av verksamhetsregler

Inom forskningsområdet artificiell intelligens (AI) och expertsystem har man sysslat en del med frågan hur man ska formalisera lagar. Bland annat har forskare på Imperial College i London i Prolog formaliserat "The British Nationality Act" en lag som definierar vad som är en brittisk medborgare.

Det har lett till en stor diskussion mellan jurister och forskare. I debatten kan man urskilja två olika synsätt på vad en lag är. För en tekniker eller en forskare inom AI eller datavetenskap är lagarna objekt som kan formaliseras, men för juristen är lagarna enbart meningsfulla i ett sammanhang, i en situation, en *diskurs*[1]. Lagarnas innebörd kan inte förstås utifrån sig själva. Lagarna har en tillämpning och en anda som *lagstiftarna* avsåg vid lagens stiftande. Likaså är *praxis*en viktigt faktor att ta hänsyn i tolkningen av lagarna och dess tillämpning.

I en forskningsrapport kallad "Fundamental Errors in Legal Logic Programming" argumenterar Leith:

".. att de logiska metoderna inte är tillämpliga för att hantera lagen, eftersom lagens verkliga problem inte är klargörandet av individuella legala termer i lagstiftningen, utan domstolarnas kontroll. Om vi vill att domstolarna ska tillämpa lagen efter det syfte som lagstiftarna hade, kan vi inte göra det genom att ge dem 'klara' lagregler eftersom de helt enkelt kommer att använda 'sedvanerätten' som argument för att tolka dessa termer."[2]

Vidare fortsätter Leith:

"'Regelbaserade' juridiska filosofer postulerar[3] *att vi kan betrakta lagen som en samling regler som synes, åtminstone för mig, ha en 'existens för sig själva'. Det är inte orättvist att se dessa filosofer som förespråkare för en formell teknisk syn på lagen, eftersom de anser att rättvisa bäst åstadkommes genom att 'applicera reglerna' på ett formellt och tekniskt sätt. Problemet som jag ser med detta synsätt på lagregler är att det bortser från de sociala bestämningarna i denna process: genom att presentera en teknisk metod hur domstolarna borde resonera presenterar de väsentligen en konservativ syn som har lite att göra med empiriska fakta." (ibid)*

---

[1] *Diskurs* betyder samtal, tal, dryftande. I detta sammanhang en problematiserande diskussion om regler och lagars giltighet.

[2] Översatt från "Controversial Issues of Expert Systems in Public Administration", Erik Frøkjær, i Expert Systems in Public administration [Frøkjær, 1989 (1)].

[3] *Postulat* av latinet *Postulo* som betyder kräver, påstår. Postulat är en sats som accepteras utan bevis. Jämför med axiom.

Detta kan tjäna som en analogi till den konflikt som ofta uppstår mellan systemutvecklare och tekniker och personer som arbetar i eller ansvarar direkt för den löpande verksamheten. Detta är viktigt att ha i åtanke vid utvecklingen av informationssystem. För teknikerna är målet att nå fram till entydiga formaliseringar av ett informationssystem, för verksamhetsfolket kan inte reglerna separeras från deras tillkomst, syfte, och sammanhang och tillämpning i den aktuella situationen.

Vi behöver göra en distinktion mellan verksamhetsregler och regler i informationssystemet för att tydliggöra problemet. Ett ytterligare exempel kan illustrera detta. Vid en tillämpning av en lag sker samtidigt en värdering av fakta i ljuset av lagens bokstav och en värdering av lagarna i ljuset av de fakta man har för handen. Detta är en komplicerad tolkningsprocess som innebär begränsningar för hur långt man kan driva formalisering av lagar och verksamhetsregler i ett informationssystem.

En formalisering av en regel i ett informationssystem relaterar till de objekt som finns i informationssystemet, en verksamhetsregel relaterar till tolkningar av den verklighet som verksamheten utgör. Denna verksamhet och dess tolkningar är social till sin natur. Betydelser av uttryck och tillämpningar av regler är ett resultat av en social process. Vi måste skilja detta från bestämning av mening och tillämpningar av regler i ett informationssystem som ofta finns formaliserat med ett formellt språk.

I en artikel med namnet "Management Espistemology"av Ronald Stamper görs en distinktion mellan *information engineering* och *system building:*

"Systembyggaren måste leverera ett system som kan utföra den önskade datatransformationen med en rimlig ekonomi. Han sysslar inte med innebörden hos data även om han är intresserad av programmens och instruktionernas semantik i meningen av att veta exakt hur de utförs på specifika maskiner. Å andra sidan måste den som är ansvarig för informationssystemet i vid bemärkelse specificera ett system i vilket data har innebörd i termer av den verkliga världen. Så en systembyggare har helt olika uppgifter att utföra jämfört med en 'informationsingenjör'."

(översatt från [Stamper, 1985 (8)])

Inom Tempora görs också en distinktion mellan vad vi kallar *verksamhetsmodellering* och *informationssystemmodellering*. Vi gör det för att klargöra att dessa modeller har olika syften och även beskriver olika delar av verkligheten.

Om vi är medvetna om att det är stor skillnad på hur en dator tillämpar ett antal regler och hur en verksamhet gör det, finns det några tumregler som man kan följa för att undvika att regler tillämpas på ett för stelbent eller t o m felaktigt sätt?

Vi kan ställa upp några allmänna riktlinjer för att hantera denna fråga:[1]

- De regelmängder som väljs för en formalisering i ett informationssystem bör vara avgränsade från andra regelmängder. Då undviker vi beroenden mellan olika regler definierade i ett annat sammanhang och vi får ett mer hanterbart system.
- Regler som kräver sunt förnuft ska inte formaliseras. Det har visat sig mycket svårt att formalisera och hantera normalt vardagsförnuft. Överlåt detta på experterna – människorna!
- De regler som formaliseras i informationssystemet ska ha en otvetydig tolkning som accepteras av de som berörs.
- Valda regler ska inte ändras ständigt. Speciellt viktigt är det om det är en fortgående process. På detta sätt undviker man att systemet deltar i en läroprocess. Ska systemet utvecklas med verksamheten i denna lärprocess gäller det att utforma systemet på ett sådant sätt att det är enkelt att identifiera och ändra regler med full kontroll över alla konsekvenser.
- Regler som har en komplex tillämpning ska inte formaliseras till entydiga regler, som t ex "rekommenderas", "bör", "i de flesta fall". Det skapar en konflikt mellan system och verksamhet. Det är viktigt att i analysskedet identifiera tillämpningssituationer som kan vara oklara. Pressa då inte fram en precisering som inte motsvaras av praxis.

Dessa allmänna riktlinjer betonar ytterligare behovet av att på ett tidigt stadium i utvecklingsprocessen hantera verksamhetsregler på ett systematiskt sätt.

Vi har nu kortfattat behandlat verksamhetsregler och deras tillämpning. Vi ska nu ge oss in på ett nytt område – språk för att beskriva verksamhetsregler.

---

[1] Inspirerat av R.J. Wieringa och J.-J.CH. Meyer,[R.J. Wieringa , 1993 (7)])

2 Temporas modelleringsspråk

För att regler ska vara meningsfulla styrinstrument måste man ha något som ska styras. När man modellerar en verksamhet eller ett informationssystem ska man alltså uttrycka reglerna i termer av verksamheten. Det gör man enklast om man först tar reda på vilka begrepp som ingår i verksamheten genom att göra en begreppsmodell, dvs en grafisk beskrivning som beskriver förhållandena mellan begreppen i en verksamhet. I Tempora har man utvecklat ett grafiskt språk för det som kallas ERT (Entity-Relationship med Tid). Det är alltså ett datamodelleringsspråk av ER-typ som är utökat med en notation för tidsberoenden.

I Tempora har man genom fallstudier påvisat ett starkt samband mellan regler och mål. Förutom att regler kan användas för att tala om när ett mål är uppfyllt visar det sig att de flesta målformuleringar på låg abstraktionsnivå direkt passar som regelformuleringar för regler på hög abstraktionsnivå. Det gör att man med fördel kan använda målmodeller både för att strukturera och aktivt leta reda på reglerna.

Förutom att regler kan delas in efter vilka mål som de stödjer eller motiveras av, så kan man också dela in dem efter vilka begrepp de påverkar. Det kan vara fråga om både restriktioner för hur objekt *får* hanteras och direkta instruktioner för hur objekt *inte* ska hanteras. Begreppsmodellen är också ett utmärkt hjälpmedel för att hitta regler. Det kan du läsa mer om nedan.

2.1 Språk för begrepp

ERT skiljer, i likhet med ER-ansatsen, klart på objekt och samband mellan objekt. ERT[1] ger dessutom möjlighet till uttrycklig modellering av tid, generalisering och specialisering (s k ISA-relationer) och komplexa objekt.

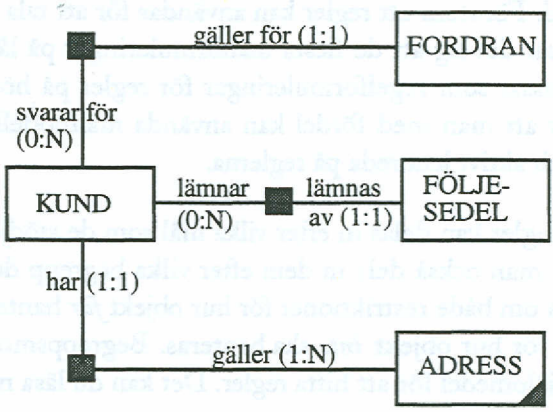
Det mest grundläggande begreppet i ERT är klass, vilket definieras som en mängd av individuella objekt med gemensamma egenskaper, dvs en samling objekt som är av samma typ och som alla exemplifierar ett begrepp. Ett ERT-schema beskriver bara klasser av objekt på samma sätt som ett databasschema. Klasserna är av två slag: entitetsklasser och värdeklasser. En entitet är en unikt urskiljbar företeelse i verkligheten, som inte enbart utgörs av en teckenföljd. Det sistnämnda objektslaget kallas värde och används för att beskriva egenskaper hos entitetsklasser.

---

[1] Temporas ERT och det språk som används för modelleringshandbokens resursperspektiv är i princip likadana (fast tid inte finns med i modelleringshandboken). Dock skiljer sig symbolerna.

Förutom objektklasserna finns klasser av objektsamband. Varje sambandstyp ses som ett rollpar där varje roll beskriver hur en enskild entitet eller ett enskilt värde ingår i ett samband. På detta sätt uttrycks varje samband med två meningar, en i varje riktning. Meningarna är syntaktiskt olika men betydelserna är ekvivalenta, dvs de är varandras motsatser men de beskriver samma förhållande. För varje roll kan man också ange hösta och lägsta värden för avbildningens kardinalitet, dvs hur många objekt som via denna sambandsroll är associerade med ett objekt i den klass som är subjekt i rollen.

I en begreppsmodell representerar entitetsklasser begrepp som används i en verksamhet. Ett exempel på ett sådant begrepp är *kund* som representerar mängden av alla instanser som är kunder (fig. 1). Värdeklasser representerar värdeförråd för tillåtna teckenkombinationer. För värdeklassen *adress* skulle värdeförrådet t ex kunna vara strängar med maximalt 256 tecken. Som ett exempel på rollpar för en förhållandetyp kan vi ta *kund har adress* respektive *adress gäller kund*, där rollparet alltså är *har/gäller*.



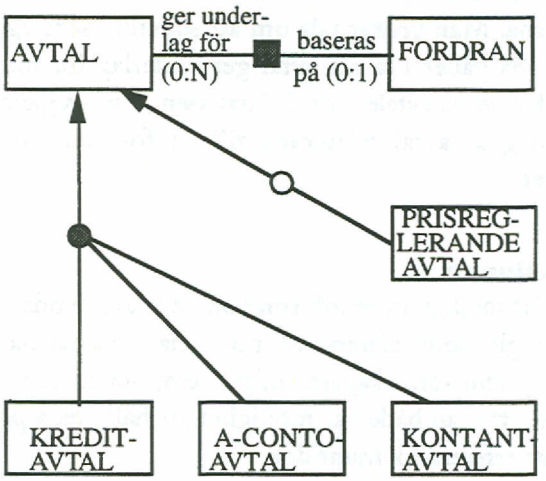
Figur 1 – ERT-modell

Figur 1 visar ett mycket enkelt exempel på en ERT-modell. I denna modell förekommer bl a entitetsklasserna "kund" och "fordran" och värdeklassen "adress". Instanserna till t ex *kund* visas inte i schemat, utan ingår som element (instanser) i mängden till klassen *kund*. Kardinalitetsvillkoret begränsar dock instanserna till *kund*. De mellanliggande objektsambandens avbildningskardinaliteter anges av uttrycken inom parentes. Alltså gäller att en kund svarar för minst noll (0) och högst godtyckligt (N) många fordringar, att en kund har exakt en adress (minst 1 och högst 1), att en fordran gäller för exakt en kund och att en adress gäller för minst en och högst godtyckligt många kunder.

Som framgått av inledningen konstrueras också i Tempora ett verktyg med vars hjälp man kan rita ERT-modeller. Till denna ERT-editor hör också en uppsättning formulär där ytterligare detaljer om klasserna kan specificeras. Bland annat gäller att värdeklasser, entitetsklasser och samband mellan objekt också kan beskrivas i sådana formulär. I de flesta formulär finns t ex ett fält där man kan fylla i en godtycklig kommentar.

Generalisering och specialisering

ERT ger också möjlighet att definiera generaliseringssamband ([1]ISA-relationer) mellan entitetsklasser. Ett sådant samband anger enkelt uttryckt att en entitetsklass står i ett delmängdsförhållande till en annan. I figur 1 används denna förhållandetyp för att specialisera entitetsklassen *avtal* i dels prisreglerande avtal dels avtal som utgår från betalningsformen.



Figur 2 – Generalisering/specialisering

Varje cirkel i grafen motsvarar ett sätt att specialisera (specialiseringskriterium). Klasser som är subklasser till samma superklass men har olika specialiseringskriterier, t ex *prisreglerande avtal* och *kreditavtal*, kan överlappa varandra och ha gemensamma element.

Om ett antal subklasser är förbundna med samma cirkel innebär det att de inte kan överlappa varandra utan att de är disjunkta[2]. I Figur 2 är det fallet med *kreditavtal*, *a-conto-avtal* och *kontantavtal*, dvs ett avtal kan bara tillhöra en av subklasserna. Om cirkeln är fylld gäller dessutom att specialiseringen är uttömmande. Det betyder att subklasserna tillsammans utgör hela superklassen. Med andra ord gäller för figur 4 att det inte finns några avtal som inte tillhör någon av de ovan uppräknade tre subklasserna (kreditavtal, a-conto-avtal och kontantavtal). Däremot kan t ex ett kreditavtal mycket väl vara prisreglerande.

En ISA-relation innebär som tidigare påpekats att mängden av entiteter som hör till subklassen är en delmängd av den mängd entiteter som hör till superklassen. Mängden av kreditavtal är alltså en delmängd av den totala mängden av avtal.

---

[1] ISA är en sammandragning av engelskans "is a", dvs "är en".

[2] Disjunkta mängder (åtskilda mängder) har inga gemensamma element (instanser).

Det är viktigt att förstå att en ISA-relation inte är ett samband mellan entiteter utan mellan entitetsklasser. Det är alltså inte så att ett kreditavtal är relaterat till ett avtal. Innebörden i ISA-relationen är i stället att om en entitet är ett kreditavtal så är den på en och samma gång också ett avtal. Den är ett element både i mängden av kreditavtal och i mängden av avtal.

Alla de samband som superklassen har med andra objektklasser gäller också för subklasserna. Man brukar tala om att superklassens egenskaper ärvs av subklasserna. Så gäller t ex att avtal ger underlag för fordran oavsett till vilken av subklasserna avtalet hör. I databasen är prisreglerande avtal likaväl som andra slag av avtal relaterade till en fordran i den ovannämnda sambandstypen.

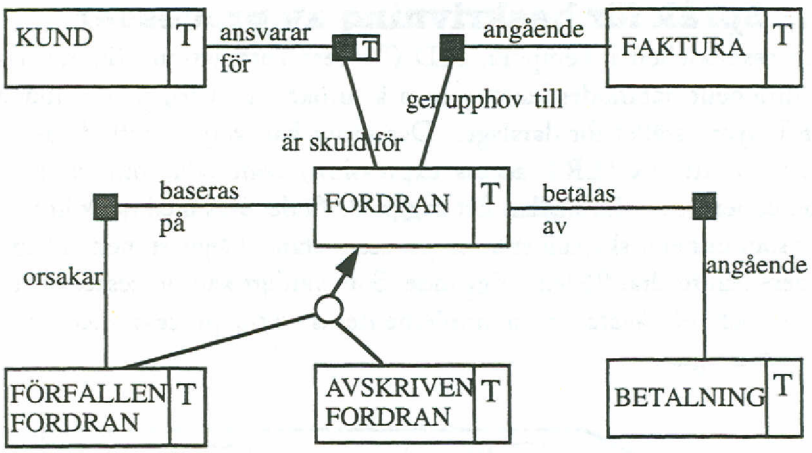
Tidsdimensionen

För att göra det möjligt att se information i ett längre tidsperspektiv och att representera regler som refererar till tiden, har den databas som hör till ett informationssystem som skapats enligt Tempora en temporal dimension. Detta innebär att man både har möjlighet att hålla reda på historisk information och att referera till framtiden.

Närmare bestämt så har varje enskilt faktum (entitet eller samband) en viss validitetsperiod, dvs en period under vilken detta faktum betraktas som sant. Validitetsperioden ska, i princip, svara mot den tidsperiod då motsvarande faktum är giltigt i verkligheten. En viss kundentitets validitetsperiod ska alltså direkt motsvara den period då vederbörande betraktas som kund i företaget. Efter denna period finns entiteten kvar som en historisk entitet som går att referera och möjligen återuppliva, men som inte är giltig just nu. På motsvarande sätt kan en entitet eller ett samband ha en giltighet som helt och hållet ligger i framtiden.

I databasen representeras validitetsperioden av en tidsstämpel som anger dess början och slut genom de verkliga tidpunkter då ett visst faktum blev sant respektive falskt. Det som representeras i databasen är med andra ord de tidpunkter då något inträffat i verkligheten (händelsetid) inte då det blivit känt i databasen (transaktionstid).

Eftersom man inte är intresserad av att hålla reda på historien för alla klasser av entiteter och samband, har man möjlighet att med bokstaven T markera de klasser i modellen vars historia man vill se. Figur 3 visar ett enkelt konceptuellt schema, där vissa klasser markerats med ett T. Det innebär att de entiteter och samband som svarat mot dessa klasser kommer att tidsstämplas medan alla övriga implicit anses vara giltiga under hela databasens livstid. Då ett icke tidsstämplat faktum görs falskt kommer det att avlägsnas ur databasen, dvs man förlorar kunskapen om att det en gång har varit giltigt, medan motsvarande operation för ett tidsstämplat faktum bara innebär att dess giltighetstid upphör.



Figur 3 – ERT-modell med T-markeringar

T-markeringar kan införas i en modell av åtminstone två skäl:

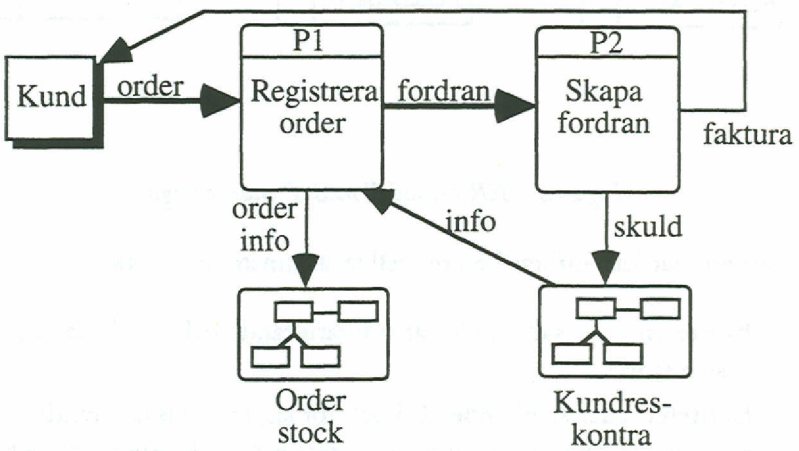
- Entiteten eller sambandet är tidsvarierande och vi vill hålla reda på dess historia.
- Entiteten eller sambandet är i och för sig inte tidsvarierande, men vi använder T-markeringen som ett sätt att modellera start- och slut-tidpunkt i stället för att införa särskilda datumattribut.

Syftet med de T-märken som införts i Figur 3 är att man i detta fall vill hålla reda på under vilken tid de olika entiteterna är giltiga. Vad som är att betrakta som validitetsperiod varierar från klass till klass. Validitetsperioden definieras av de regler som för varje klass beskriver hur entiteter uppstår och görs ogiltiga. För fordran gäller t ex att den är giltig från det att den uppstår till dess att den är betald. Sambandet *fordran är skuld för kund* har en T-markering därför att en kund kan överta fordran från en annan kund och vi vill hålla reda på vilken kund som varit betalningsansvarig under olika tidsperioder. T-markeringarna på *förfallen fordran* och *avskriven fordran* innebär att vi har information om vilka tidsperioder en fordran eventuellt intar något av dessa båda tillstånd. Validitetsperioderna för en fordrans tillhörighet till dessa klasser kan dock inte överlappa varandra eftersom mängderna som svarar mot klasserna i varje ögonblick ska vara disjunkta[1] (åtskilda).

[1] Två disjunkta intervall har inga gemensamma tidpunkter.

2.2 Språk för beskrivning av processer

Processmodellen i Tempora, PID (Process Interaction Diagram) är ett traditionellt dataflödesdiagram som är utökat med triggande flöden och ERT-vyer i stället för datalager. Dessutom kan varje dataflöde associeras med ett uttryck (ERT access expression) som talar om vilka ERT-komponenter det förmedlar. Ett triggande flöde visas med tjock linje och är det som initierar skapandet av en processinstans. I figuren nedan (fig. 4) är order- och fordranflödena triggande. Som vanligt kan processer dekomponeras och på lägsta nivån implementeras varje process med ett antal aktivitetsregler.



Figur 4 – En övergripande processmodell över orderregistrering

Reglerna i processer specificerar förhållandet mellan inflöden och utflöden. I exemplet ovan skulle en regel kunna se ut på följande sätt för process P1:

```
When order(Ordernr, Kundnr, Belopp)
if kund.K har nummer.Kundnr and <fler villkor>
then fordran(Kundnr, Belopp) and
order [gäller_för kund.K, har värde.Belopp]
```

Flöden mellan externa agenter och processer representeras av predikat, medan flöden till och från ERT-vyer representeras av direkta referenser till ERT via sina ERT-uttryck (ERT access expressions). Mer om PID kan man läsa i Temporas metodhandbok eller i någon av triadrapporterna om Tempora.

2.3 Språk för att beskriva mål

Målmodellering har inte integrerats helt i Tempora men under arbetet med fallstudier har den typen av modellering ändå använts uttryckligt hela tiden. Anledningen att det inte integrerats är framför allt resursbrist och har inte med användbarheten att göra. Snarare har man tyckt att eftersom mål är så lika regler kan man ändå representera mål med regler, vilket också delvis stämmer.

I det praktiska arbetet har vi använt oss av ett sätt att modellera mål som härstammar från det i modelleringshandboken. Det är en grafisk modell som består av en öppen mängd av olika objekttyper (t ex mål, problem, orsak, förutsättning, strategi, policy och vision) som är sammanbundna med två olika sorters förhållanden:

- Strukturella förhållanden

Strukturella förhållanden representerar lösa motivationsförhållanden som inte nödvändigtvis innebär att det motiverade målet uppfyller någon del av det motiverande. Det är mer en fråga om ren strukturering av kunskap om en verksamhets mål.

- Influensförhållanden

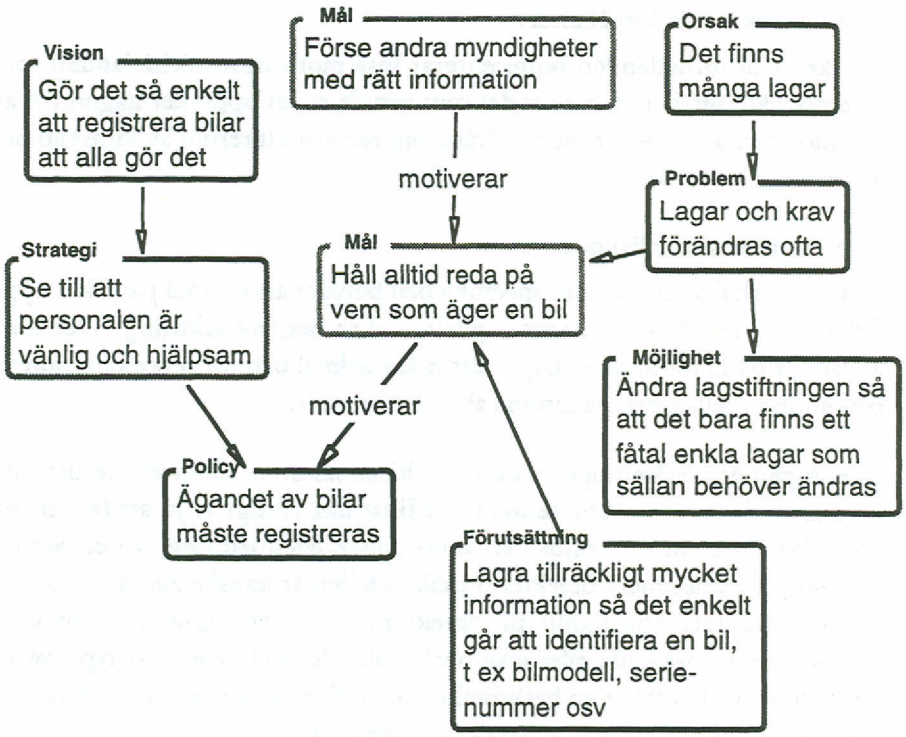
Influensförhållanden är mer specifika och betyder att ett mål påverkar uppfyllandegraden hos ett annat i positiv eller negativ riktning. Det kan användas både för uppdelning av ett mål i delmål och för representation av beroenden mellan mål på samma abstraktionsnivå.

Egentligen är benämningen på målmodellen missvisande eftersom det inte bara är mål som kan representeras[1]. Minst lika viktigt är ju att få reda på t ex vilka problem som finns i en verksamhet. Med ledning av det faktum att regler kan definiera objekten i målmodellen är kanske *tillståndsrum* ett bra sammanfattande namn på objekten. Det återspeglar att objekten representerar önskvärda eller icke önskvärda tillstånd av en viss typ i verksamheten. Nedan följer en beskrivning av vanligt förekommande tillståndsrum men det är inte någon fullständig uppräkning:

Mål	Önskat (möjligen framtida) tillstånd i verksamheten
Problem	Oönskat tillstånd i verksamheten
Förutsättning	Nödvändigt tillstånd (för ett annat tillståndsrum)
Orsak	Tillstånd som är en orsak (för något annat)
Möjlighet	Ett ännu outrett tillstånd
Vision	Ett icke direkt planeringsbart idealtillstånd
Policy	Generella riktlinjer för verksamheten
Strategi	Begränsning av möjliga handlingsalternativ

[1] En kanske bättre benämning är det intentionella perspektivet som finns i modelleringshandboken. Alternativt kan man även se målmodellen som en beslut- och argumentationsmodell.

Till varje förhållande i målmodellen ska det också finnas en förklaring av resonemanget bakom förhållandet och en lista över eventuella antaganden som förhållandet bygger på. Tills vidare skrivs det in som text i formuläret för förhållandet. Nedan ser vi ett konstruerat exempel på en målmodell (fig. 5).



Figur 5 – En fingerad målmodell för ett bilregister

2.4 Verksamhetsregler i Tempora

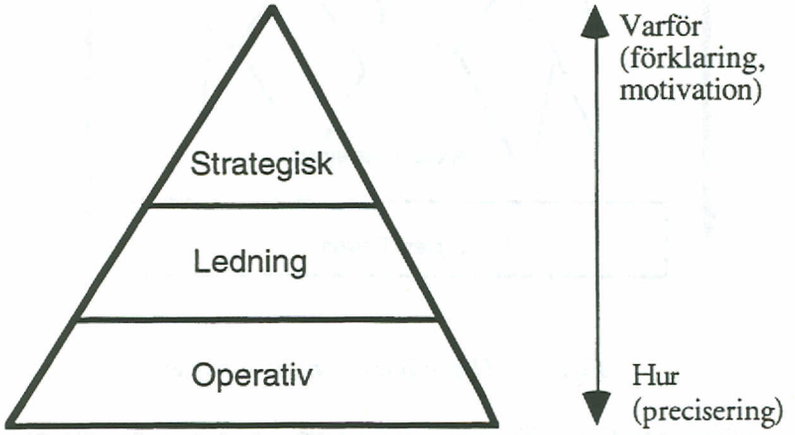
I TEMPORA är begreppet verksamhet av fundamental betydelse. Med verksamhet menas den kontinuerliga process som pågår inom en organisation för att uppfylla dess mål. Processen karaktäriseras av att den hanterar vissa fenomen och att hanteringen kontrolleras av vissa regler. Verksamhetsregler kan vara dels definierande eller begränsande regler dels regler som specificerar vilka åtgärder som ska vidtagas i vissa situationer.

Verksamhetsregler är alltså regler som kontrollerar verksamheten. Mer precist kan man säga att verksamhetsregler kan vara något av följande:

- mer eller mindre vaga högnivåmål och policies för verksamheten
- mer precisa uttalanden som beskriver det sätt som verksamheten valt för att uppnå sina mål och hur man realiserar sina policies
- olika typer av regler som påförts verksamheten utifrån, som t ex lagar och avtal.

Verksamhetsmål kan vara allt från vaga övergripande uttalanden av typen "verksamhetens mål, på lång sikt, är att maximera vinsten", till mer precisa uttalanden om hur verksamheten ska uppnå det övergripande målet. För att kunna identifiera mål och utforma strategier med vars hjälp målen ska realiseras, kan man utföra en strukturell nedbrytning som resulterar i en hierarki av mål, och medel för att uppnå dem. Vid varje nivå i en sådan analys gör man ett perspektivskifte så att det som på en nivå är en del av en strategi för att uppnå ett högnivåmål, sedan blir ett mål som kan uppnås med en ännu mer precis strategi. Delmål ska alltså ses som ett sätt att uppnå ett överliggande mål. Ju längre man kommer i analysen och ju lägre nivåer man kommer till desto större är benägenheten att kalla målen för verksamhetsregler.

Nivåerna motsvaras i målmodellen av nivåindelade strukturer av mål. På lägre nivåer finner man mer precisa mål. Man utgår alltså från vaga övergripande mål och riktlinjer via mer precisa policies och strategier. På lägsta nivå finns sedan de formella kvantifierbara (operativa) målen. Genom att gå uppåt finner man motiven till målen.

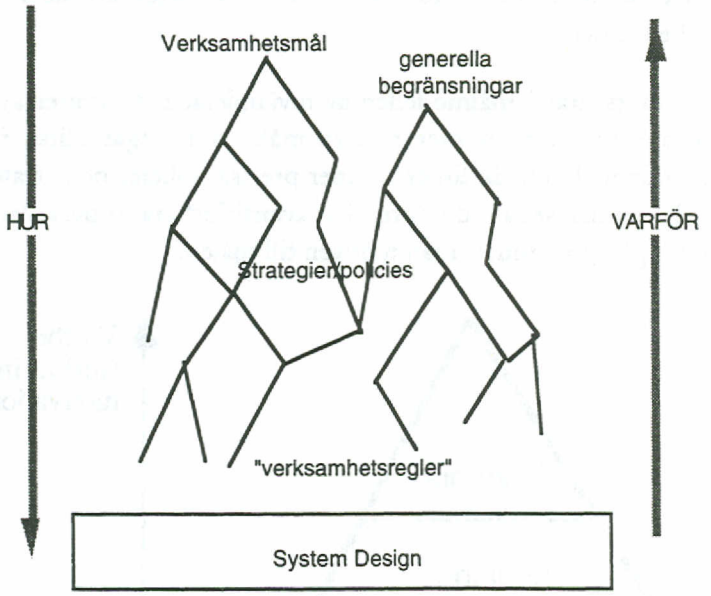


Figur 6 – Nivåer i målmodellen

Senare kommer vi att diskutera hur målmodellering kan användas för att finna verksamhetsreglerna. För kompletthetens skull och för att göra det möjligt för läsaren att relatera till andra former av regelinhämtning, ska vi också diskutera målmodellering i ett något större sammanhang.

Regler som påförs verksamheten utifrån är inte mycket annorlunda än andra eftersom de i princip kan fås fram genom att generella begränsningar delas upp. Ett exempel på en sådan regel är att "varje verksamhet måste följa lagarna i det land det opererar". Figur 6 illustrerar vagt förhållandet mellan olika abstraktionsnivåer. På lägsta nivån kan några regler vara realiserade och inbyggda i ett datorsystem.

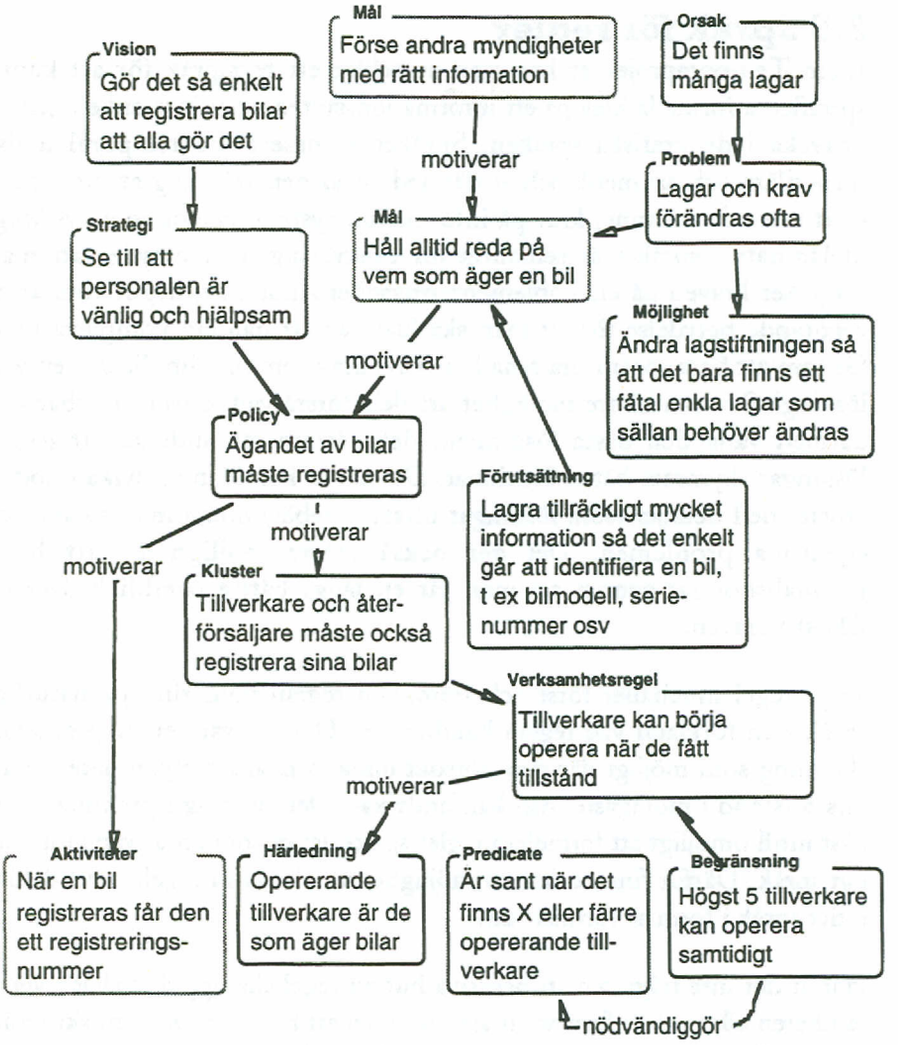
Liksom för vad som är fallet med mål och som visas i figur 7, talar regler långt ned i hierarkin om hur högre nivåers regler realiseras. På motsvarande sätt fungerar högre nivåers regler som motivation för lägre (dvs svar på frågan varför). Regler på hög nivå representerar övergripande mål och generella begränsningar (t ex lagar och kollektivavtal) som successivt bryts ned i mer konkreta strategier och policies som upprätthåller de övergripande målen och begränsningarna i verksamheten. Strategier och policies bryts ned i konkreta verksamhetsregler som i sin tur kan brytas ned i formella regler som kan implementeras i ett informationssystem.



Figur 7 – Hierarki av verksamhetsregler

Som tidigare påtalats är Temporas synsätt att "en verksamhetsfunktion kan karaktäriseras av att (1) den hanterar vissa begrepp och i hanteringen så (2) följer den vissa regler". Parallellt med den funktionella uppdelningen (PID), behövs begreppsmodellering (ERT) och explicit specificering av regler (ERL).

Som exempel på hur mål och regler kan användas tillsammans kan man göra en grafisk representation som visar mål och regler tillsammans (fig. 8). Det ger en bättre förståelse för varför regler finns och vad som kan gå fel om de inte följs.



Figur 8 – En kombinerad mål och regelmodell

Från en konceptuell/logisk utgångspunkt kan alla verksamhetsregler ses som begränsningar av möjliga vägar att uppfylla verksamhetsmålen. Gäller det däremot att implementera dem som regler för att kontrollera beteende, är det ofta bättre att implementera dem som aktiviteter eller härledningar. I modellering av informationssystem finns det därför ett behov av att skilja på villkor som begränsar antalet tillåtna tillstånd, härledningar som definierar hur ny information kan härledas från information i databasen och aktiviteter som definierar vilka förändringar i databasen eller vilka signaler som ska skickas till omgivningen, givet att vissa villkor är uppfyllda.

2.5 Språk för regler

Inom Temporaprojektet har man utvecklat ett textspråk för att kunna specificera formella krav på ett informationssystem som inte enkelt går att uttrycka i de grafiska språken. Språket är baserat på temporal logik, mängdlära och aritmetik och är därmed också generellt nog att uttrycka i stort sett alla relevanta krav på informationssystem. Något som avsiktligt utelämnats i språket är semantik för exekvering. Det betyder att man uttrycker kraven på ett implementeringsoberoende sätt, något som är av avgörande betydelse för att man ska klara av att hantera komplexa krav. Genom att bara specificera rena krav (dvs krav som inte implicerar en viss lösning) får man bättre möjlighet att dels förstå vad kraven innebär och därmed välja den bästa lösningen, dels lättare att undvika att gamla lösningar skymmer bättre lösningar. Dessutom kan man undvika onödigt arbete med detaljer (som lösningar oftast innebär) innan man förstått de egentliga problemen. Det ger också större möjligheter att hitta rationaliseringar genom att man får en långt bättre överblick över de faktiska kraven.

Varje regel innehåller först och främst en regelformulering på naturligt språk som förklarar vad regeln handlar om. Det ska vara en så precis beskrivning som möjligt där man försökt undanröja alla tvetydigheter så att missförstånd i möjligaste mån kan undvikas. Det visar sig i praktiken vara näst intill omöjligt att formulera regler så precist genom att använda naturligt språk. Därför finns också en möjlighet att skriva in regeln i ett deklarativt språk i form av formell text.

Här är det inte fråga om att beskriva hur en regel ska upprätthållas i verksamheten eller i ett informationssystem, utan att beskriva vad som ska gälla. Det är alltså bara krav på verksamheten eller informationssystemet som ska beskrivas. På så sätt får man en mer generell beskrivning med långt färre detaljer som därmed är enklare att förstå och kan realiseras på många olika sätt. Man har därmed bättre möjlighet att välja den bästa lösningen eftersom man inte i förväg har förutsatt en viss lösning.

Det formella språket i sig är baserat på temporal logik som är utökad med mängdoperatorer och algebra. Det gör att det som uttrycks i språket får en entydig betydelse som gör att man kraftigt kan reducera antalet missförstånd. Det gör också att språket blir både kraftfullt, generellt och relativt kompakt. Trots att stor möda har lagts ned på att göra språket så lättanvänt som möjligt krävs det en del övning innan någon som inte tidigare använt deklarativa kunskapsrepresentationsspråk kan författa egna regler.

För att underlätta förståelsen, har man i Tempora introducerat bl a beslutstabeller för att visa förhållandet mellan regler på ett överskådligt sätt.

Regelkategorier

I informationssystemmodellen måste alla formella regler tillhöra en kategori. De fyra nedersta reglerna i fig. 8 exemplifierar varsin regelkategori. Den formella texten syns inte i figuren utan läggs separat i ett formulär som är associerat med respektive regelobjekt. För formella regler används kategorin för att tala om hur regeln ska exekveras, medan det för informella regler endast är en klassificering.

Varje kategori är också associerad med en viss syntaktisk struktur hos den formella texten. Villkor består till exempel endast av en then-del, härledningar av en if-del och en then-del. Aktivitetsregler består av en when-del, en eventuell if-del och en then-del.

Förutom för exekveringssemantiken är kategorier användbara även för att hitta regler, för att stödja tänkandet och för att klassificera reglerna. Vilken kategori en regel slutligen kommer att tillhöra beror också på hur begreppsmodellen struktureras och vilka ERT-komponenter som är härledda. Formella regler kan alltid transformeras från ett syntaktiskt format till ett annat, men det kan man vänta med tills man slutgiltigt bestämt kategorin för regeln.

- Villkor används för begränsningar av modellstrukturer och beteende.
- Härledningar används för att härleda ny information från befintlig och för att undvika att redundant information läggs in i informationssystemet utan att upptäckas.
- Aktivitetsregler används för att specificera beteende antingen som resultat av interna eller av externa händelser – givet vissa villkor.
- Predikat används för att bryta ut delar av komplicerade regler dels för att dela upp problemet men också för att slippa kopiera delar som är lika i flera regler. Det är en struktureringsmekanism som liknar subrutiner i programmeringsspråk.

Ett exempel på vad som kan motivera ändring av kategorin är när en lagrad entitet transformeras till en härledd. Då är det sannolikt så att eventuella villkor som direkt begränsar entiteten ska göras om till härledningsregler. På motsvarande sätt kan härledningar av entiteter behöva ändras till villkor eller aktivitetsregler när entiteten transformeras till lagrad.

Nedan visas den formella texten till de fyra nedersta reglerna i figur 8.

Event/Action rule: När en bil registreras får den ett registreringsnummer.

when bil.B
if nytt_registreringsnummer(N)
then bil.B har registreringsnummer.N

Derivation rule: Opererande tillverkare är de som äger bilar.

if tillverkare.T äger bil
then opererande_tillverkare.T

Predicate: Är sant när det finns X eller färre opererande tillverkare.

if number_of {T such_that opererande_tillverkare.T} ≤ X
then högst_X_opererande_tillverkare(X)

Constraint rule: Högst 5 tillverkare kan operera samtidigt.

högst_X_opererande_tillverkare(5)

Förutom formell text innehåller formuläret också ett fält där man kan lägga in förklaringar och anteckningar, samt ett antal olika fält för att man ska kunna associera regeln till såväl andra regler som till objekt i de andra modellerna.

Förhållanden till regler:

- Regel till regel (motivation)

  Motivationsförhållanden – En regel R1 motiveras av en annan regel R2 om den definierar minst en implicerad aspekt av R2. Enklare uttryckt kan man säga att R1:s existens motiveras av att den mer detaljerat uttrycker något som R2 uttrycker direkt eller indirekt. En motiverad regel (R1) behöver inte uttrycka ett rent delmål till den motiverande regeln (R2) utan kan handla om andra saker som bara indirekt uttrycks i R2.

- Regel till regel (nödvändiggör)

  Nödvändiggörandeförhållandet är en starkare variant av motivationsförhållandet och innebär att den nödvändiggjorda regeln behövs för att den andra ska vara fullständigt specificerad. Ett typexempel på detta är en regel som använder ett predikat vars definition då är nödvändiggjord av regeln (fig. 8).

- Regel till mål (motivation)

  Eftersom mål och regler är två sidor av samma sak så används samma förhållandetyp mellan regler och mål som mellan regler.

- Regel till ERT (association)

  Associationsförhållanden – En regel kan associeras med ERT-komponenter (dvs entitetsklasser, värdeklasser och förhållandetyper) om den uttrycker en viktig begränsning för komponenten.

- Regel till ERT (referens)

  Referensförhållanden – De flesta formella regler refererar till ERT-komponenter. Referensförhållanden härleds från alla sådana referenser och kan användas för sökning i Temporas editor.

- Regel till process (implementering)

  Implementeringsförhållanden – En PID-agent (dvs en process eller en extern agent) kan implementeras av en eller flera regler. Den hierarkiska strukturen hos PID gör att det bara är processerna på lägsta nivån som behöver vara implementerade av regler.
- Regel till ERT-vy

  Varje ERT-vy i PID kan vara associerad med ett formellt uttryck (ERT access expression) som talar om vilka objekt och förhållanden ur ERT som ingår i vyn. Detta förhållande har inget officiellt namn men är närmast besläktat med referensförhållandet.

Det formella regelspråket

Syftet med följande beskrivning av regelspråket är att ge en översikt över språkets uttrycksmöjligheter. Av utrymmesmässiga skäl är det tyvärr inte möjligt att ge en fullständig beskrivning inom ramen för denna rapport även om visst material finns bifogat i bilagan. Den intresserade hänvisas till Temporas metodhandbok.

Generellt finns det ett några saker att säga om regelspråket ERL.

Variabler i ERL representeras syntaktiskt av ord som börjar med stor bokstav. På så sätt är X, _, Sture och W6r_g alla variabler.

Alla fria variabler i formella regler är allkvantifierade, dvs precis som i SQL försöker man hitta alla lösningar på en fråga. Det gäller däremot inte för ERT-objekt utan variabel som i stället existenskvantifieras (fler detaljer med exempel finns beskrivna i bilagan *Guidelines for rule formalization*).

Entitetsklasser och värdeklasser

För att vara meningsfulla måste regler uttryckas så att de reglerar någonting. I ERL regleras t ex komponenter i ERT, dataflöden i PID eller predikat som definieras i regelmodellen. Om vi börjar med ERT så sker alla referenser med s k *ERT access expressions.* Ett sådant börjar med namnet på en entitetsklass eller på en värdeklass och följs av noll eller flera par av namn på en roll för ett associerat förhållande och namnet på objektet det refererar till (t ex fordran gäller_för kund har adress).

Förhållandetyper

Ett namn på förhållanden i Tempora är bara unikt inom kontexten av de två objekt det sammanbinder. Det betyder att man måste referera till namnen på båda objekten och namnet på förhållandet för att kunna identifiera det. Ska man t ex referera till förhållandet *har* mellan objekten *kund* och *adress* så måste man säga kund har adress därför att det mycket väl skulle kunna finnas ett annat förhållande med likadana rollpar (t ex kund har kundnummer).

Dataflöden

När man vill referera till dataflöden mellan processer eller mellan processer och externa agenter, använder man sig av predikat. Ett predikat är i detta sammanhang att betrakta som en namngiven tupel (aggregat) med lika många kolumner som antalet argument i predikatet. Dvs det är en mängd med data som har ett namn och där ordningen på de ingående elementen är av betydelse. Om man vill skicka iväg ett resultat till en extern agent så refererar man ett predikat i then-delen till en regel i processen. Processen genererar sedan det flöde som man använder för att skicka resultatet till agenten. Vill man ta emot ett meddelande från omgivningen så refererar man till predikatet i if-delen i en eller flera regler i den process som tar emot flödet. I varje argument som man är intresserad av lägger man en variabel som man antingen kan testa eller använda för att söka information i databasen. Som exempel kan vi ta den regel som hör ihop med fig. 4:

When order(Ordernr, Kundnr, Belopp)
if kund.K har nummer.Kundnr and Belopp > 50
then fordran(Kundnr, Belopp) and
order [gäller_för kund.K, har värde.Belopp]

Här tar vi emot flödet *order* som överför en treställig tupel. I if-delen söker vi fram en kund som baseras på det andra argumentet som i sin tur unifieras[1] med värdeklassen nummer. Via ERT-accessuttrycket hittar vi kunden vars surrogat (intern identifierare) vi lägger i en ny variabel K. Därefter testar vi om variabelns belopp har ett värde större än 50. Sedan genereras ett nytt flöde (fordran) med två variabler som argument. Dessa två variabler kommer ha samma värden som i det första flödet. Till sist lägger vi in en ny order i databasen (ny eftersom vi inte har givit den en variabel) med ett förhållande till samma kund som i if-delen och ett annat förhållande till värdeklassen värde.

---

[1] Unifiering används i bl a logikprogrammeringsspråket Prolog och betyder fritt översatt att försöka göra lika. Om en variabel t ex inte redan är bunden till ett visst värde, kan den anta värdet av det som den ska unifieras med.

Operatorer

Logiska operatorer

Förutom de vanliga logiska operatorerna *and*, *or* och *not* finns också *xor* (exklusivt eller) samt prefixoperatorn *only_one_of* som båda är sanna när och endast när exakt en av deras operander är sanna:

true xor true ∫ false

only_one_of(true, false, false) ∫ true

Mängdoperatorer

I ERL finns två typer av mängder; ordnade <{a,b,c}> (dvs tupler) och oordnade {a,b,c}. En mängd kan konstrueras på olika sätt. Man kan t ex göra en samling (collection) baserad på antalet kombinationer av variabelsubstitut för vilka ett uttryck är sant. På den resulterande oordnade mängden kan sedan operationer som *antal* och *medelvärde* appliceras. Följande uttryck ger antalet personer som äger bil:

number_of {X,Y for_which person.X äger bil.Y}

Man kan också jämföra mängder direkt med varandra. Nedanstående exempel säger att personer som äger bil inte får vara anställda:

{X for_which person.X äger bil} intersect {X for_which anställd.X} == {}

Setof är en alternativ prefixoperator som returnerar en oordnad mängd med unika element. Nedanstående uttryck är ekvivalent med det förra.

•setof(X, person.X äger bil) intersect {X for_which anställd.X} == {}

På motsvarande sätt kan man använda *bagof* för tupler (ordnade mängder). I följande två exempel är *V* antalet lagrade personposter i databasen (även dubletter räknas här). Variabeln *U* är antalet förhållandeposter hos relationen *äger*.

V = number_of bagof(X, person.X äger bil)

U = number_of <{X,Y for_which person.X äger bil.Y}>

För tupler av tupler kan man också projicera en eller flera positioner av de inre tuplerna.

X == project(C, <{A,C}>, bagof(<{P,B}>, person.X äger bil.B))

Y ==project(<{N,C}>,<{A,C,D}>,

bagof(<{N,P,B}>, pnr.N för person.X äger bil.B))

X är då en tupel och Y är en tupel av par.

Tidsoperatorer

Det finns en stor uppsättning tidsoperatorer i Tempora. Det finns de som hanterar intervall och de som hanterar tidpunkter. Dessutom kan tid hanteras explicit med variabler och implicit med relativa jämförelser. Dessutom finns en stor mängd fördefinierade konstanter, både för intervall och tidpunkter.

Som exempel på explicit användning av tid med konstanta tidpunkter kan vi ta at-operatorn. Med dess hjälp kan man förskjuta en fråga till en angiven tid och få svaret som gäller för den tidpunkten. Tiden kan anges explicit och det finns även aritmetiska operatorer för att utföra beräkningar.

konto at 07:30:00 ≡ konto at start_of_today>>7*hours+30*minutes

Sometime_in_past är ett exempel på en operator som använder implicit tid med relativa jämförelser. Denna typ av operator kan ställa en fråga över flera tidpunkter. Nedanstående uttryck är sant om orderbekräftelse börjar gälla samtidigt som order någon gång i dåtid.

sometime_in_past (orderbekräftelse for B and order for O and B starts_with O)

Operatorn *for* returnerar ett intervall under vilket ett uttryck alltid är sant. *Starts_with* är en operator som jämför två intervall och är sant om båda intervallen börjar vid samma tidpunkt.

Man kan också strunta i tiden och då rör man sig med de fakta som är sanna just nu. Mer information om hur tiden används i ERL finns i bilagan. Hur Tempora utnyttjar tid finns beskrivet mer utförligt i Temporas metodhandbok och i bilagan.

3 Beslutstabeller för regelmodellering – erfarenheter från Tempora

3.1 Beslutstabeller för kommunikation

Flera olika fallstudier pekar på att det finns ett behov av ett kommunikativt språk för regler (förutom det vanliga språket eller de formella regelspråken). Det underlättar kommunikationen mellan experter och med systemutvecklare [Johnson, 1987 (3), Kontio, 1989 (4), Öhlund, 1992 (9)]. Kunskapen inom ett visst verksamhetsområde formuleras ofta för första gången på ett mer precist sätt då ett informationssystem ska byggas och specificeras. Regler inom ett verksamhetsområde finns ofta inte nedtecknade och även om så vore fallet är inte tolkningarna entydiga utan tillämpningskunskapen är viktig för att förstå reglerna.

Det kommunikativa språket ska också vara till hjälp i analysarbetet. Tyvärr så räcker inte det vanliga språket till för att finna och strukturera och till slut formalisera regler. Inte heller det s k konceptuella grafiska språk som vi använder vid analys av begrepp och flöden och mål är tillräckligt och heltäckande när det gäller att representera regler.

Vi har i den här rapporten t ex visat hur regler kan representeras i begreppsmodeller (eller objektmodeller om man så vill) och hur man kan använda en målmodell för att driva en regelanalys som knyter viktiga verksamhetsregler till verksamhetsmålen. Men ett stort problem är hur man på ett mer precist sätt ska formulera och analysera reglerna. De formella regelspråk som systemutvecklare använder är ofta för svårbegripliga för andra än specialister och inte heller i ett sådant språk finns stöd för strukturering och analys av regler.

I sökandet efter en tänkbar kandidat för en kommunikativ regelrepresentation har vi funnit att beslutstabeller uppfyllt våra krav. Beslutstabeller är inte något nytt men de har fallit i glömska och används inte längre så mycket. Vi kommer i detta avsnitt att visa hur vi använde beslutstabeller i vårt arbete med en fallstudie i Tempora och dela med oss av de erfarenheter och lärdomar vi fått genom detta arbete. För de som glömt eller som inte träffat på beslutstabeller tidigare börjar vi med en kort introduktion.

3.2 Beslutstabeller – en kort introduktion

Beslutstabeller användes i ganska stor utsträckning på 60- och 70-talen som ett modellerings- och analysverktyg inom systemutveckling. De har använts för att generera kod, simuleringar, analys av lagtexter, specifikation av stora system m m. Styrkan med beslutstabeller är den enkla och tydliga formen.

De är mycket kompakta när det gäller att representera problemområden och de ger en översikt av olika relevanta situationer som kan uppkomma. De ger en enkel och tydlig standard för dokumentation som kan ligga till grund för en kommunikation mellan experter och systemutvecklare. De är också kraftfulla analysverktyg för kontroll av fullständighet, motsägelser och redundans. De har också fördelen att de är formellt definierade till skillnad från naturligt språk.

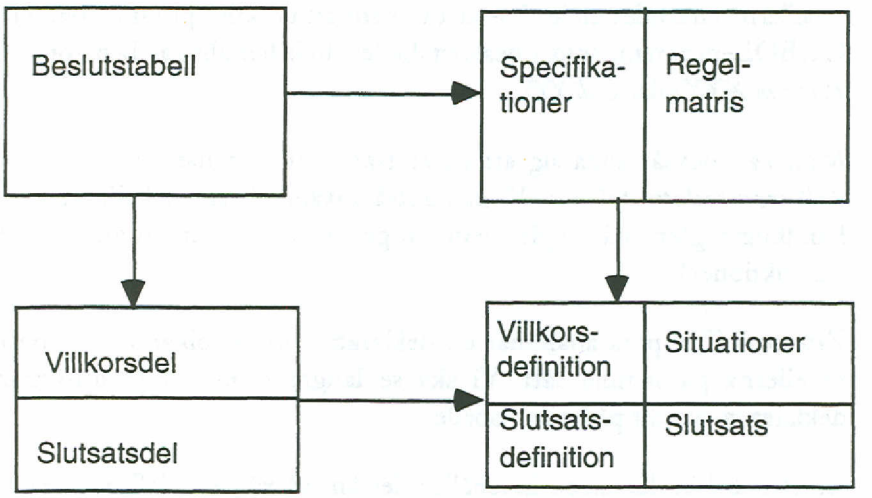
Från våra erfarenheter inom Tempora-projektet där vi genomfört en fallstudie i Posten känner vi till svårigheten att få en överblick över reglerna och problemen med att analysera utsagor i naturligt språk på ett bra sätt för att sedan kunna formalisera dem i ERL-regler (som är Temporas regelspråk). Vi tror att beslutstabeller kan bidra till ett mer strukturerat och systematiskt arbetssätt för att formulera och formalisera regler i ett formellt regelspråk. Ofta är det ett stort hopp från naturligt språk som tillåter en mängd vagheter (t ex att *och* ofta används som synonymt med disjunktion: Du kan göra A och du kan göra B) till att formulera regler formellt. Beslutstabeller reducerar dessa möjliga vagheter men tillåter fortfarande en viss informalitet. Framför allt kan de användas som en övergång mellan naturligt språk och formella regler. Beslutstabellerna är enkla att arbeta med och det finns inte någon anledning att först arbeta med utsagor på naturligt språk.

Den andra fördelen är alltså dess förmåga att vara en referenspunkt i olika faser av formulering, analys och formalisering. Innan vi går in på några exempel på hur vi använt beslutstabeller i fallstudien i Posten ska vi först göra en kort introduktion av beslutstabellernas elementära grunder. För en mer utvecklad beskrivning hänvisar vi till Staffan Perssons utmärkta bok om beslutstabeller[1].

---

[1] Beslutstabeller – 1. Beskrivning av regelsamband, Staffan Persson, Studentlitteratur 1971.[Persson, 1971 (6)]

Beslutstabellen kan sägas bestå av ett antal delar som finns uppritade i figuren som följer:



Figur 9 – En beslutstabells olika delar (fritt från [Nilsson, 1970 (5)]

Ett exempel på en mycket enkel beslutstabell:

Krav på analysverktyg		
Lätt att förstå	J	N
Lätt att överblicka	J	N
Lätta att ändra	J	N
Läs om beslutstabeller	x	
Läs om flödesscheman		x

Villkorsdel

Slutsatsdel

Tabell 1

Ovanstående tabell är en tvåvärd tabell , vilket betyder att den enbart kan ha två olika värden per villkorsdefinition i regelmatrisen. I vårt fall är de *J* (Ja - Sann) och *N* (Nej - Falsk). En regel motsvaras av en kolumn i regelmatrisen. En kombination av situationer (J, N, etc) för varje villkorsdefinition ger regelns villkorsdel och en kombination av ett antal slutsatser ger dess slutsatsdel. I denna tabell är alla dessa kombinationer underförstått logiskt sammankopplade som konjunktioner. Det bör ju naturligtvis anges för att det ska vara möjligt att läsa tabellen på ett korrekt sätt. Kombinationerna kan också vara disjunktioner. Villkorsdefinitionerna kan också de vara logiska uttryck eller utsagor som t ex *Lätt att förstå* och *Lätt att överblicka.*

Slutsatsdelen i en beslutstabell benämns ofta som beslut (därav namnet) eller en sekvens av procedurer eller handlingar som ska utföras. Vi använder här både betydelsen att dra en slutsats och att utföra något. Beslutstabellerna användes under 70-talet t ex för att beskriva programlogiken i ett COBOL-program, som i beslutstabellen fick handlingar benämnda som *Perform XXX* eller *Call YYY*.

Man kan också tänka sig att en slutsats i en beslutstabell kan vara att *Tillämpa beslutstabell xxx*. Vi kan alltså använda beslutstabellen både som handlingsregler och regler som anger vad som är tillåtna tillstånd (restriktioner).

Eftersom Tempora anammar en deklarativ ansats tolkar vi inte beslutstabellerna på samma sätt. Vi ska se längre fram hur vi utnyttjar ett deklarativt synsätt på beslutstabeller.

En flervärd beslutstabell innehåller fler än två värden (definierade) på en rad i regelmatrisen. Denna typ av beslutstabell kan vara användbar för de som inte är vana att tänka i logiska termer. Flervärda beslutstabeller ger även kompakta tabeller. Det är möjligt att översätta en flervärd tabell till en tvåvärd för att kunna göra en del kontroller.

Elementära och komplexa regler

I en regelmatris kan man beteckna alla möjliga situationer. Om vi har en tvåvärd beslutstabell med enbart två olika värden (t ex Ja eller Nej) har vi även möjlighet att specificera ett ospecificerat värde (null-value) i matrisen. Detta ska tolkas som att värdet är irrelevant för regeln. Om det finns ett ospecificerat värde i matrisen betyder det att alla situationer inte är specificerade. Då har vi en komplex regel.

En komplex regel kan expanderas till flera elementära regler beroende på antalet tillåtna värden i varje specifikation och antalet ospecificerade värden. Detta är ett effektivt sätt att komprimera en regel när vissa fall är irrelevanta.

Undantag (partiella eller icke-partiella)

Det är möjligt och användbart att lägga till en kolumn i slutet av en beslutstabell för att hantera alla fall som inte finns specificerade i regelmatrisen. Dessa undantag kan antingen vara partiella eller icke-partiella. Partiella undantag har villkor i regelmatrisen men icke-partiella har det inte. I det senare fallet fångas alla undantag upp som inte finns specificerade i regelmatrisen.

Vi hänvisar till litteraturlistan för den som vill studera beslutstabellernas anatomi, dess konstruktionsprinciper, kontroller för fullständighet, vaghet, redundans, motsägelser etc. Ovanstående enkla principer räcker för den fortsatta beskrivningen av våra erfarenheter.

3.3 Beslutstabeller i Tempora – exempel från en fallstudie i Posten

Innan vi går in på några exempel från fallstudien kommer vi att visa ett enkelt och förhoppningsfullt pedagogiskt exempel.

I fallstudien hade vi följande mycket enkla utsaga uttryckt på naturligt språk:

En faktura är en förfallen fordran om den inte blivit fullt betald på förfallodagen och inte heller blivit avskriven

Vi väljer en beslutstabell som är tvåvärd och har tre olika villkorsdefinitioner. En tvåvärd beslutstabell kan som mest innehålla 2^n elementära regler. I detta fall som mest 8 regler. Vi använder beslutstabellen för att analysera utsagan och vi betraktar även andra möjliga situationer än den som utsagan beskriver. I detta fall ska vi inte göra någonting i de andra situationerna – något som inte alltid är uppenbart i en analyssituation.

Vi får följande tabell:

Fordran helt betald		N	N	J
Förfallodag överskriden	N	J	J	J
Fordran avskriven		J	N	N
Ingen åtgärd	X	X		X
Fordran förfallen			X	

Tabell 2

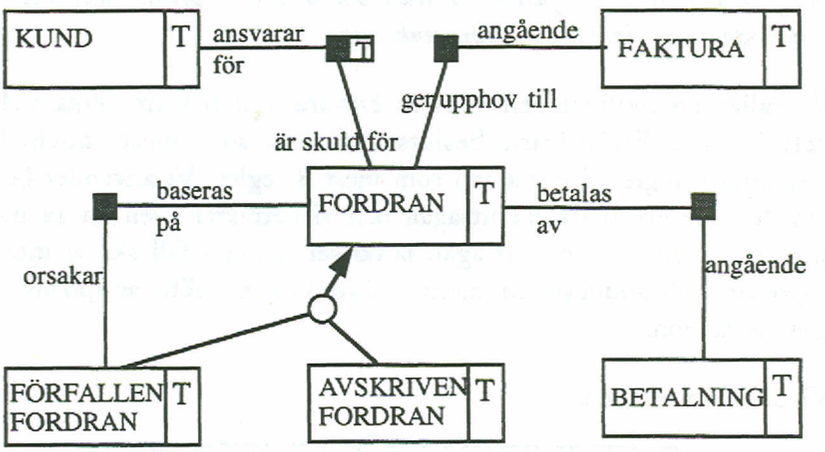
Observera att villkorsdefinitionerna i tabellen har positiva formuleringar. Anledningen är att de blir lättare att förstå och speciellt undviker man förvirringar som dubbla negationer kan föra med sig.

Denna tabell kan komprimeras till det skuggade fältet och resten av fallen kan tas om hand genom ett undantag. Vi får en regel som lyder så här på naturligt språk:

Om fordran inte är helt betald och förfallodagen överskriden och fordran inte är avskriven **så** är fordran förfallen.

Men beslutstabeller visar sina starka sidor framför allt då formuleringarna involverar ett antal regler. Vi kan t ex tänka oss att samla ett antal regler som hanterar betalningsuppföljningen i en tabell. Vid formuleringen av villkoren i en beslutstabell har vi naturligtvis stor nytta av en begreppsmodell. Begreppsmodellen är nödvändig för att man ska kunna formulera reglerna mer precist, dvs veta vad villkorsutsagorna refererar till.

Nedan finns en sådan modell i Temporas ERT-språk:



Figur 10 – Begreppsmodell över fordran

Vi kan då formulera vår beslutstabell i termer av vår tidigare definierade begreppsmodell i stället för på naturligt språk, men fortfarande på en övergripande informell nivå där vi inte definierar alla detaljer. Vi kan göra det genom att utnyttja s k accessuttryck (ERT Access Expression) som finns i Temporas regelspråk. Vi ska i korthet visa hur det kan gå till.

I det första villkoret har vi inget enkelt sätt att med hjälp av begreppsmodellen beskriva att fordran är helt betald. Det handlar om att summera delbetalningar för att ta reda på om fordran är helt betald. Det är något som måste beräknas med hjälp av begrepp i modellen. Vi antar att det görs av ett predikat som avgör om fordran är betald. Vi uttrycker det så här: *P: Fordran helt betald.* Predikatet kan vara sant eller falskt.

Nästa villkor är lättare att beskriva med hjälp av en begreppsmodell. Vi antar att FÖRFALLODAG är en egenskap hos FORDRAN (visas ej i denna bild av modellen).

Förfallodag överskriden kan uttryckas så här:

*FORDRAN.F has FÖRFALLODAG < **start_of_today***

start_of_today är här en temporal operator i Tempora och .F i FORDRAN.F är ett surrogat för ett objekt.

Det tredje villkoret *Fordran avskriven* kan vi uttrycka med hjälp av specialiseringen av FORDRAN i två underkategorier:

AVSKRIVEN_FORDRAN and FÖRFALLEN_FORDRAN: FORDRAN.F and AVSKRIVEN_FORDRAN.F

Detta uttryck betyder att entiteten FORDRAN är AVSKRIVEN_FORDRAN.

Slutligen kan vi uttrycka i åtgärdsdelen av beslutstabellen att surrogatet F antas vara FÖRFALLEN_FORDRAN.F. Detta är ett faktum som läggs till Temporas databas.

P: Fordran helt betald		N	N	J
FORDRAN.F has FÖRFALLODAG < start_of_today	N	J	J	J
FORDRAN.F and AVSKRIVEN_FORDRAN.F		J	N	N
Ingen åtgärd	X	X		X
FÖRFALLEN_FORDRAN.F			X	

Tabell 3

Detta förfaringssätt hjälper oss att få en bättre struktur på regelformuleringen. Att fordran är betald är uppenbarligen någon som är mer komplext att beräkna. Men vi är inte intresserade av några delresultat av denna beräkning utan vill bara veta om den är helt betald eller inte. Vi kan alltså bortse från hur beräkningen görs och ägna oss åt det vid ett senare tillfälle. Det innebär att vi kan arbeta uppifrån och ned i problemlösningen och vänta med detaljerna till ett senare skede i analysen.

Nu ska vi titta lite på ett exempel som är lite mer komplicerat och därmed mer realistiskt. Exemplet är hämtat från fallstudien och taget direkt från en sammanslagning av en text från det modelleringsverktyg som används i Tempora:

The price of a delivery note line is calculated using the nominal price times weight, when there is no price regulating agreement and not any specified quantity, but weight is specified in the line and the postal item is not a group rated item.

A delivery note line has a price that is calculated using the nominal kilo price, if the line has no quantity but weight specified and the article is a group rated postal item. There is no agreement specified.

The delivery note line has a nominal price for a priced article when the line does not refer to an agreement and has no weight but quantity specified.

The delivery note line has a nominal price when no agreement is specified and quantity is specified. The line does not refer to a group rated postal item and there is no price regulation agreement.

Delivery note line is the minimum of quantity times nominal price and weight times kilo price, if postal item is a group rated item without a price regulation agreement but both weight and quantity is specified in the delivery note line.

Figur 11

Denna text är inte så lätt att överblicka och arbeta med, speciellt inte i en situation då informationen kommuniceras muntligen! Under arbetet med fallstudien, som innebar en större revision, använde vi beslutstabeller för att gå igenom och få en bättre ordning på regelmassan som fanns implementerad i verktyget. Vi ritade upp den informella beslutstabellen på nästa sida under arbetet med att skriva om reglerna för prissättning av följesedelsartiklar.

Price setting for delivery_note_articles

Table 1: derivation of delivery_note_line has price			D+I							
Group rated Postal item	Y	N	N	Y	Y	Y	N		N	Y
delivery_note_line has quantity			Y	Y	N	Y	N		N	N
delivery_note_line has weight				N	Y	Y	Y		N	N
Priceregulation agreement referred in dev.note.line	Y	Y	N	N	N	N	N		N	N
PRICE=priced_article has nominal price*QUANTITY			X	X						
PRICE=group_rated_postal_item has_kilo nominal_price					X					
PRICE=article has nominal_price * weight							X			
PRICE=min(quantity*price,weight*price)						X				
price = price_regulation has price	X	X								
FALSE									X	X
	R81	R82	R80	R79	R78	R69	R77			

Table2: price_regulation_agreement(PRICE)										
reduction.amount	N	N	N	Y	N	Y	N	Y	N	Y
reduction.percentage	N	N	N	Y	Y	N	N	Y	Y	N
reduction.netprice	N	N	N	Y	Y	Y	Y	N	N	N
dev.note.line.Q	Y	Y	N	N	N	N	N	N	N	N
dev.note.line.W	Y	N	Y	N	N	N	N	N	N	N
PRICE= MIN(Q*P,W*P)	X									
PRICE= Q*P		X								
PRICE= W*P			X							
price= netprice -red.amount - %				X						
price= netprice - %					X					
price= netprice -red.amount						X				
PRICE=NETPRICE							X			
PRICE=NOM.PRICE - AMOUNT - %								X		
PRICE=NOM.PRICE - %									X	
PRICE=NOM.PRICE - AMOUNT										X
	See	R88	R88	R83	R83	R83	R83	R83	R83	R83

Table3: price_regulation_grouprating(PRICE)										
delivery_note_line has quantity	Y	N	Y	Y	N	Y	Y	Y	N	
delivery_note_line has weight	N	Y	Y	N	Y	Y	Y	Y	N	
minimum.weigth <= DNL.weigth		Y	Y		N	N	Y	N		
minimum_quantity <= DNL.quantity	Y		Y	N		Y	N	N		
price_reg.price= min(quantity*pr_by_quant,weight*pr_by_kilo)			X							
price_reg.price= price_regulation_grouprating.pr_per_item	X					X				
price_reg.price= price_regulation_grouprating.price_by_kilo		X					X			
price_reg.price=article has nominal_price*QW				X	X			X		
	R84	R84	R85	R86	R86	R81	R81	R87		
False									X	
Com: Delivery_note_line must have quantity or weight for group_rated_postal_items										

Tabell 4 – Beslutstabell från fallstudien i Posten

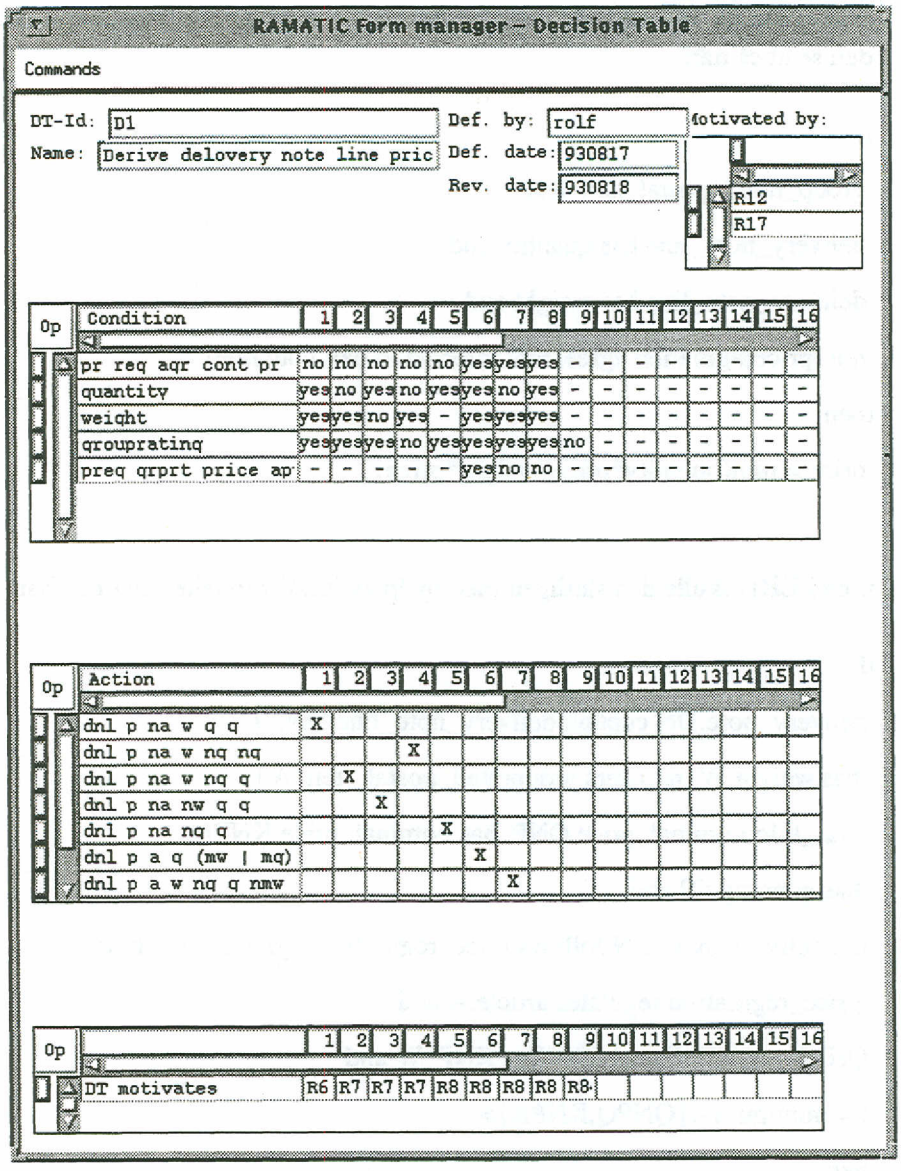
Beslutstabeller används, liksom i figuren ovan, som ett kommunikationsinstrument och som en checklista för att finna redundanser, oklarheter och motsägelser.

I figuren på föregående sida har vi sett exempel på en beslutstabell som användes i fallstudien. Observera att beslutstabellen inte är färdig utan innehåller en del oklarheter och felaktigheter som senare rättats till. Namnen på de slutliga ERL-reglerna som senare formaliserades i fallstudien (t ex R84) finns längst ned i varje kolumn. I vissa fall har flera kolumner kombinerats med en regel beroende på hur pass bra regelspråket kan uttrycka mer komplexa villkor. Det är också fallet då slutsatserna är likartade och premisserna är relativt enkla att uttrycka i regelspråket. Den tidigare texten (Figur 11) motsvaras av reglerna R69, R77, R78, R79, R80 i beslutstabell nr 1(tabell 4).

Trots att vi är rätt så specifika i beslutstabellerna, upptäckte vi vid den slutliga formaliseringen av en regel att vi behövde mer verksamhetskunskap för ytterligare preciseringar. Det visar nyttan med att formalisera; formaliserandet tvingar fram ett klargörande av en mängd underliggande antaganden. Informella regler måste alltså ofta specificeras i detalj för att kunna formaliseras. Detta understryker ytterligare behovet av att kunna arbeta uppifrån och ned.

Hur som helst kan vi med ovanstående exempel se nyttan med att använda beslutstabeller, speciellt vid omfattande och komplexa regelverk. Ännu mer uppenbar blir denna nytta då vi enbart har tillgång till muntlig information.

Tempora har anammat användningen av beslutstabeller för regelmodelleringen. På nästa sida ser vi en skärmbild från den grafiska editorn som utvecklats i Tempora i Ramatic. Bilden visar en enkel implementering av beslutstabeller. På bilden visas tabell 1 från föregående tabell (tabell 4) som gjordes i ett kalkylprogram.



Figur 12 – Beslutstabell implementerad i Ramatic

Verktyget kan generera enkla mallar från beslutstabellen. Mallarna kan sedan användas som utgångspunkt för en vidare precisering i ERL. För att få en bild av hur en regel i beslutstabellen blir implementerad i ERL visar vi ett exempel från beslutstabell 1. Det gäller regel R69. Texten börjar med en informell beskrivning på naturligt språk och följs av dess motsvarighet i ERL.

R69

Calculation of delivery_note_line price is the minimum of quantity*price and weight*kiloprice. This rule deals with articles without an agreement but both weight and quantity specified in the delivery note line.

Om vi skulle generera en mall för denna regel från beslutstabellen skulle den se ut så här:

```
if
 group_rated_postal_item and
 delivery_note_line has quantity and
 delivery_note_line has weight and
 not (priceregulation agreement referred in dev.note.line)
then
 price = min(quantity*price, weight*price)
```

Och i ERL skulle den slutligen med hjälp av ERT-modellen se ut så här:

```
if
 delivery_note.DN contain delivery_note_line.DNL [
  has weight.W, registers grouprated_postal_item.A [
   has_kilo nominal_price.QNP, has nominal_price.KNP],
  has quantity.Q] and
 not delivery_note.DN follows price_regulating_agreement contain
  price_regulation regulates article.A and
 QNPQ = QNP*Q and KNPK = KNP*W and
 P= minimum <{QNPQ,KNPK}>
then
 delivery_note_line.DNL has price.P
```

Ingenting hindrar att vi utnyttjar formell ERL-syntax i beslutstabellen för att få en exakt motsvarighet mellan beslutstabell och regel. Det har inte gjorts i detta fall. Beslutstabellen har främst utnyttjats i de tidiga faserna. Fortfarande återstår arbetet med att formalisera länken mellan beslutstabeller och ERL-regler.

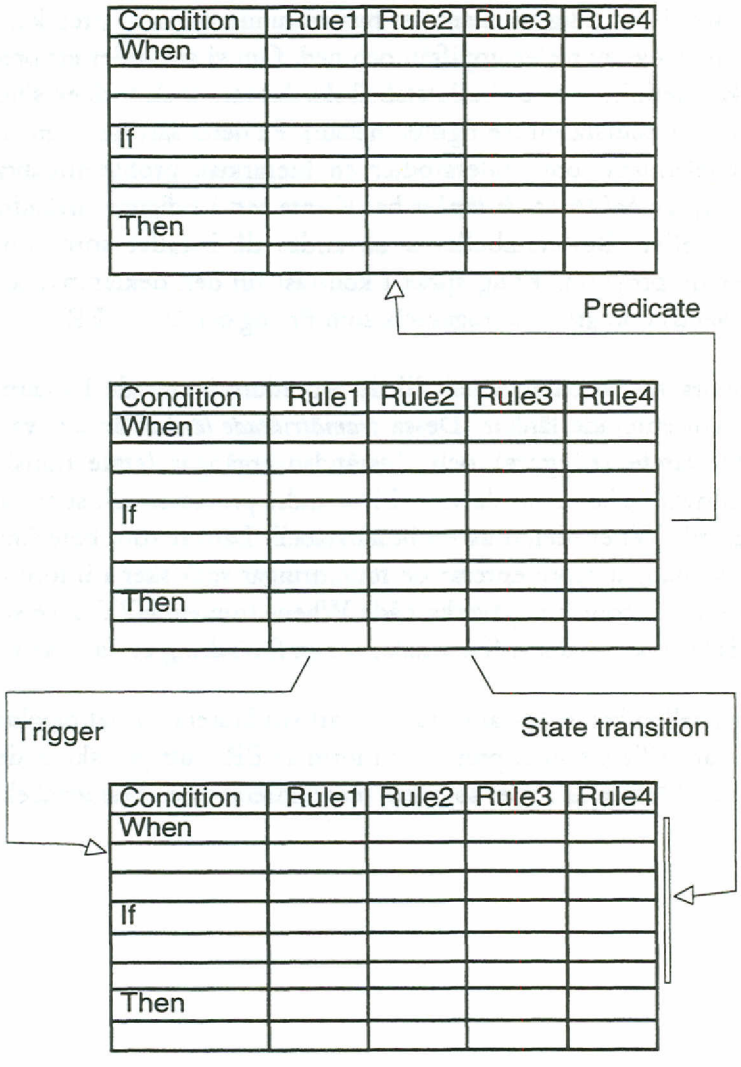
Det som var bra med beslutstabellen var att den underlättade diskussionen och kontrollerna av olika situationer.

3.4 Regelstrukturering med beslutstabeller

Vi har sett i det tidigare exemplet hur vi kunde använda predikat för att tillåta en analys av regler uppifrån och ned. Om vi använder ett predikat i en villkorsdefinition av en beslutstabell ska det vara länkat till en slutsatsdel i en annan beslutstabell (se figuren nedan). På detta sätt får vi en strukturereringsmekanism som understödjer en hierarkisk problemnedbrytning. Denna typ av *bakåtriktade länkar* har vi inte sett i tidigare användning av beslutstabeller. Beslutstabellerna användes då i stället som procedurorienterade programmeringsspråk i kontrast till den deklarativa ansatsen som används i programmeringsspråk som Prolog och även i ERL.

I Tempora har vi en parallell till de procedurorienterade länkarna eller händelseorienterade länkar. Dessa *framåtriktade länkar* är av två typer, framåtdrivande (triggers) och tillståndsövergångar (state transitions). Framåtdrivande länkar är de som driver andra processer och som därför är länkade till When-delen av en beslutstabell. Länkar som betecknar tillståndsövergångar representerar de förändringar som sker i informationsdatabasen och som kan påverka både When- (om ett ERT-accessuttryck används här) och If-delen (i det fallet när en förändring av databasen sker).

Beslutstabellen kan också användas för att strukturera en flat regeldatabas. Genom att välja ett antal premisser i form av ERL-uttryck skulle det vara möjligt att härleda de regler som kan representeras av en beslutstabell.



Figur 13 – Strukturella länkar mellan beslutstabeller

3.5 Sammanfattning och erfarenheter

Vi kan sammanfatta de erfarenheter vi fått genom att använda beslutstabeller i fallstudien:

- Det var enklare att analysera de olika situationer som kunde uppstå.

  Det är uppenbart att det är mycket svårt för en analytiker att hålla alla dessa fall i huvudet även om man är väl tränad i regelspråket. Det måste även gälla den som är verksamhetskunnig. Vi noterade många fall där vi kunde upptäcka förbiseenden, motsägelser, redundans och möjligheter att slå ihop regler.
- Beslutstabellerna underlättade kommunikationen mellan experter och analytiker.

  I vårt fall var en deltagare väl förtrogen med verksamheten och den andre var helt ny och visste inget om verksamheten. Vi analyserade verksamheten samtidigt som vi kommunicerade verksamhetskunskap via tabellen. Beslutstabellerna fungerade i det fallet som ett förklarande och pedagogiskt instrument.
- Tabellerna underlättade strukturering av reglerna genom att tillhandahålla en klustringsmekanism som hanterade en viss mängd av situationer.
- Tabellerna bidrog till att skapa en bra struktur för regler.
- Tabellerna understödde en analys som lät oss skjuta upp specificeringen av detaljer till ett senare skede.

  Vi använde predikat och härledningsregler för detta ändamål och länkade på så sätt beslutstabeller till varandra.
- Tabellerna användes i fallstudien för att från mer informella utsagor (genom att använda begrepp inom verksamheten), utveckla en mer precis och formaliserad representation i form av ett antal regler i ett formellt språk (ERL).
- Tabellerna gjorde det möjligt att upprätthålla en kongruens mellan verksamhetsregler och deras implementering i ett informationssystem.

4 Praktisk regel-modellering – erfarenheter från Tempora

4.1 Metodik

Metoden för att utveckla informationssystem med hjälp av regler bygger på ett kunskapsperspektiv. Utvecklingen går ut på att inhämta och strukturera kunskap om domän och önskad tillämpning. Det är därför viktigt att den kan förfinas successivt allt eftersom fler detaljer blir kända. Därför uppmuntras återkoppling från detaljerade modeller till abstrakta modeller. På så sätt kan brister som upptäcks i detaljerade modeller rättas till också i de abstrakta modellerna så att validiteten hos de abstrakta modellerna kan upprätthållas. Därmed kan de abstrakta modellerna användas för att strukturera de detaljerade och fungera som navigationshjälpmedel.

Genom att det är möjligt att i editorn lägga in kunskapen i valfritt delspråk är det därmed också möjligt att lägga till ny kunskap till modellen så fort den finns tillgänglig.

I Tempora delar vi in utvecklingen i fyra faser, nämligen verksamhetsmodellering, informationssystemmodellering, informationssystemdesign och implementering. Dessa motsvarar följande scenario:

1 Utveckla en konceptuell modell av verksamheten.

   Motsvarar verksamhetsmodellering där man beskriver och analyserar beteendet och strukturen hos verksamhetsområdet samt de krav som man ställer på informationssystemet. Beskrivningen omfattar med andra ord både det som ligger innanför och utanför det informationssystem som så småningom kommer att utvecklas.

2 Utveckla en konceptuell modell av informationssystemet.

   Motsvarar informationssystemmodellering där man beskriver och analyserar struktur och beteende i informationssystemet. Här ska man se till att få med all kunskap om systemdomänen som ska vara med i informationssystemet.

3 Utföra en design av informationssystemet.

   Informationssystemdesign är en utvidgning av informationssystemmodellen med detaljinformation (t ex användargränssnitt) och exekveringssemantik som behövs i implementationen.

4 Generera ett körbart system.

Motsvarar implementering som baseras på specifikationen som producerats i informationssystemdesignen. Den ska innehålla tillräckligt mycket kunskap för att mappningen ska vara en i stort sett automatisk process.

Att begreppet faser används betyder inte att en metod med strikt åtskilda faser förespråkas eller att faserna ska vara strikt sekventiella, som i Vattenfallsmodellen. Tvärt om – praktiskt taget vilken projektmetod som helst kan överlagras för att man ska kunna uppfylla olika krav på projektledning. I huvudsak använder vi faserna som en referensram i utvecklingsprocessen som underlättar beskrivningen av denna aktivitet.

Beskrivning av metoden

I enlighet med idén om successiv förfining, listar vi ett antal aktiviteter och hänsynstaganden som är aktuella vid regelmodellering. De är inte nödvändigtvis listade i någon speciell ordning, utan situationen får bestämma vilka som passar bäst.

1 *Finna regler oberoende av ERT och PID.* Det mest direkta sättet att fånga in regler är att fråga efter målen för verksamheten. Som tidigare beskrivits är skillnaden mellan regler och mål i stort sett bara en fråga om abstraktion. Exempel på sådana frågor är: "Vilken policy finns inom verksamheten?", "Vilka restriktioner gäller för verksamheten?", "Hur genomdriver man dessa?". På så sätt kan man finna i huvudsak generella regler som uttryckts på naturligt språk och som definierar mål. Med den sista frågan kan man successivt få fram allt mer detaljerad kunskap om verksamheten.

2 *Finna regler genom att använda ERT-modellen.* Givet en situation där man har en förhållandevis omfattande begreppsmodell kan man ställa frågor av typen: "Vilka policies gäller för denna typ av objekt?", "Vilka restriktioner gäller för detta förhållande?" osv. Det finns ett omvänt förhållande mellan komplexiteten hos en begreppsmodell och antalet olika typer av restriktioner som kan uttryckas grafiskt. Komplexiteten kan reduceras genom att man använder regler för att uttrycka vissa typer, speciellt sådana som inte är så vanligt förekommande. Man kan också undvika komplexitet genom att inte införa begrepp och förhållanden för att representera restriktioner. Det är oftast bättre att införa några extra regler i stället.

3 *Finna regler genom att använda PID-modellen.* Det finns åtminstone tre olika sätt att använda processmodellering för att finna regler. Man kan använda processmodellen för att fånga och strukturera regler som beskriver, styr eller begränsar beteendet hos processer. Med beskrivande regler (på naturligt språk) kan man översiktligt dela upp olika syften som processen har. Styrningen av en process kan ske med hjälp av regler som definierar beteendet utifrån de olika villkor som gäller för processen. Med begränsningsregler kan man få mer kunskap om när en process kan tillämpas eller förhindra oönskade effekter. Processmodellen kan också användas för att strukturera regler för dynamiken i ett system och presentera dessa regler på ett lättfattligt sätt.

4 *Förfina regler.* Regler kan förfinas eller detaljeras på åtminstone tre sätt genom att ställa frågan "Hur upprätthålls denna regel i verksamheten?". På så sätt kan man finna mer detaljerade regler antingen direkt eller via t ex en process som upprätthåller den mer abstrakta regeln. Man kan också använda sig av beslutstabeller för att dela upp en regel i olika fall som består av kombinationer av villkor. Detta är ofta fallet för processer eller regler som definierar processer. Det är också vanligt att det krävs fler formella regler för att beskriva en informell regel i detalj, även om den först kan verka enkel. Detta beror dels på den inneboende tvetydigheten i naturligt språk som ger utrymme för olika tolkningar dels på att informella regler tenderar att vara mer abstrakta. Även här kan man med fördel använda beslutstabeller i många fall.

5 *Formalisera regler.* Resultaten av aktiviteterna i punkt 1-3 är till övervägande del uttryckta informellt på naturligt språk. Det är inget som hindrar att verksamhetsregler uttrycks direkt på formellt ERL-språk men i så fall krävs det stor vana vid det formella språket. Det är också troligt att den formella regeln blir mycket omfattande och komplex och därmed svår att förstå. Dessutom kan det vara svårt att upprätthålla sambandet med begreppsmodellen om den ändras ofta. Därför rekommenderas att resultatet av aktivitet 1-4 ska vara informella regler men med tillräckligt detaljerade beskrivningar för att de ska kunna formaliseras i ett fåtal formella regler.

   Vid formalisering av regler som definierar processer är situationen lite annorlunda. Processer kan ju dekomponeras vilket gör att man alternativt kan välja att antingen dekomponera processerna så långt att de direkt går att formalisera eller också kan man nöja sig med en abstrakt PID där varje process beskrivs av ett antal informella regler som i sin tur ligger till grund för formella regler. Det första alternativet har fördelen att processmodellen bättre kan förklara och strukturera reglerna. Man kan (som nämnts i aktivitet 4) även använda beslutstabeller för att åskådliggöra implementeringen av en process. Man gör då en tabell över tillåtna kombinationer av ingående dataflöden och specificerar vilka utflöden som genereras för varje kombination. På så sätt kan man specificera varje kombination för sig utan att förlora överblicken över de olika kombinationerna.

6 *Konceptualiseria interrelationer mellan regler och mellan regler och objekt i andra modelltyper.* Komponenter i olika modelltyper kan knytas samman med hjälp av strukturella förhållanden. På så sätt kan man få reda på t ex vilka regler som refererar till en viss entitet, vilka entiteter som används i en process eller vilka regler som motiveras av en viss regel.

7 *Verifiera regler genom syntaktisk och semantisk kontroll.* Syntaktisk kontroll av regler kan ske redan innan en modell bedöms vara komplett. Tanken är att semantiska kontroller inte ska utföras innan en modell är färdig för att undvika allt för stora problemlistningar. Det innebär att om de flesta problemen beror på att modellen inte är komplett så är det ingen större idé att presentera dem så länge utvecklaren är medveten om dem. Dessutom tar de senare kontrollerna lång tid att utföra och kan på så sätt störa utvecklingsprocessen.

8 *Validering av regler.* Man kan förstås tänka sig ett antal olika tekniker för att validera regler. Det är inte omöjligt att generera parafraser från ERL-regler t ex till naturligt språk. Det blir då enklare att förstå om man inte kan ERL även om man riskerar viss tvetydighet.

Inom Tempora har man utvecklat ett verktyg för statisk animering av processregler. Det kan ge en första indikation om reglerna är giltiga. Det finns även ett verktyg för att generera en prototyp av systemet som gör att man kan validera hela specifikationen. Det är lite mer arbete att generera en prototyp men å andra sidan får man se exakt hur reglerna påverkar systemet. Att kunna använda strukturella förhållanden för sökningar är en enklare form av stöd som ger möjlighet att gå igenom regler.

Ytterligare en enkel form av stöd är de olika abstraktionsmekanismerna i Tempora som gör det möjligt att bibehålla överblicken också över ett relativt stort system. Dessutom är formaliseringsprocessen ett utmärkt sätt att eliminera tvetydigheter i specifikationen, genom att man tvingas bestämma sig exakt för vad en informell regel egentligen betyder. Förutsatt att den kunskap som behövs för detta kommer från domänen, så kan man säga att det är en form av utvärdering av såväl de informella reglerna som av begreppsmodellen. Den minsta effekten blir i alla fall att misstag och motsägelser bringas i dagen. Därmed inte sagt att det alltid är så enkelt att lösa dem.

4.2 Generella råd och riktlinjer för regelmodellering

Specifikationen av regler görs uppifrån och ned där specifikationen stegvis förfinas och där man gör ett antal iterationer (upprepningar) av vissa utvecklingssteg. Det sista steget gör man för att säkerställa att tidiga fasers modeller stämmer överens med ändringar som görs i senare fasers modeller och fortlöpande hålls giltiga. Utvecklingen börjar med att man gör en målmodell som övergripande beskriver målen för verksamheten. Genom att utgå från målen fångar man sedan strategier och policies som i sin tur förfinas med hjälp av de aktiviteter som beskrivits ovan. Förfiningar eller uppdelningar av regler och mål sker med *motiveras-av*-förhållanden.

Nedanstående uppräkning av riktlinjer är inte uttömmande. Bara de viktigaste finns med.

- Implementeringsförhållanden mellan processer och regler ska referera från en process till regler som antingen direkt implementerar processen eller direkt begränsar dess beteende.
- Associationsförhållanden mellan regler och ERT-komponenter ska referera till entiteter, förhållanden eller värdeklasser för vilka regeln uttrycker en viktig begränsning eller på annat sätt direkt påverkar komponenten. Den kan även användas för att gruppera regler till centrala ERT-komponenter trots att det kanske inte finns någon direkt referens. Till exempel kan regler som har att göra med prissättning associeras med entiteten prisreglering.

- Kategorin för en regel är viktig i de sena faserna av utvecklingsprocessen. Initialt får en regel kategorin *odefinierad* som indikerar att ett val av kategori måste göras. Informella regler tilldelas kategorin *oanvänd* som betyder att man inte kommer att försöka använda regeln i det exekverbara systemet. Dessa regler används i stället för att motivera andra och mer detaljerade regler och kan på så sätt förklara formella lågnivåregler. De övriga fyra kategorierna används för formella regler. Varje kategori har en viss semantik för hur regeln ska exekveras.
- En regel formaliseras i stort sett aldrig under verksamhetsmodellering om inte verksamheten är starkt formaliserad till sin natur. Det är ofta opraktiskt att formalisera regler för tidigt, dvs så länge begrepps- och processmodellerna förändras. Detta för att referenser i den formella texten till komponenter i ERT och PID blir meningslösa när komponenterna ändrar namn eller tas bort i de andra delmodellerna.
- Identifierare för regler ska börja på stora R och identifierare för beslutstabeller på stora D. Genereringen av regelmallar från beslutstabeller förutsätter denna konvention och fungerar inte bra om konventionen inte följs.
- Namnet på en regel eller beslutstabell ska vara ett beskrivande namn som gör att man åtminstone får förståelse för vad regeln handlar om i stora drag. Det är viktigt till exempel i en grafisk representation av regler samt när man söker regler och resultatet presenteras som en lista av regelnamn.

5 Sårbarhetskarta vid SGU – ett praktikfall i regelmodellering

Sten-Erik Öhlund

5.1 Introduktion

Denna beskrivning av ett praktikfall är intressant ur flera synpunkter vad gäller regelmodellering. Det problem som skulle lösas var – vilket vi snart ska se – av komplex natur. De regler som vi kom att analysera och formulera var inte några verksamhetsregler utan regler som representerade den kunskap som geologer använde för att producera en viss karttyp. Det var alltså regler av den typ som man vanligen finner i s k expertsystem. Detta praktikfall genomfördes inom ramen för ett projekt kallat modellbaserad kunskapsinhämtning (MBKI) som drevs tillsammans med Infologics AB under 1992 (se Triadrapport nr 6/93 för närmare information om detta). En annan intressant egenskap hos praktikfallet var att experterna själva arbetade utan uttryckliga regler. Reglerna fanns alltså inte dokumenterade och de var med säkerhet olika från expert till expert. Experterna visste inte vad de egentligen visste, ett inte ovanligt fenomen i samband med verklig expertis.

Problemet handlade alltså om att synliggöra reglerna för experterna själva. Detta är av allmänt intresse eftersom det säkert även gäller implicita verksamhetsregler. En tredje aspekt som gör praktikfallet intressant var den komplicerade problemlösningen. Problemet var både ett klassificeringsproblem och en beräkning av utbredning i rummet. Detta ställde större krav på analysarbetet.

Slutligen kan vi nämna att vi använde beslutstabeller som ett sätt att komma fram i detta relativt svåra analysarbete. Vi kommer att redovisa våra erfarenheter av att använda denna teknik för regelmodellering som ett sätt att förenkla ett till synes komplext problem.

Vi kommer att börja med en kort beskrivning av problemet och fortsätta med att beskriva analysarbetet. Sedan avslutar vi med att sammanfatta de viktigaste erfarenheterna.

Vi vill understryka att alla sakfel som kan ha insmugit sig rörande geologi eller framställandet av sårbarhetskartor, helt och hållet måste lastas skribentens okunskap inom dessa områden.

5.2 Bakgrund

SGU, Statens Geologiska Undersökning, producerar olika s k kundanpassade kartor för att möta olika kunders behov av geologisk information inom områden som miljö och hälsa, fysisk planering, naturresursförsörjning m m. De frågor som kunderna ställer kan oftast inte besvaras med hjälp av *en* typ av geologisk information, utan information behövs från flera källor och olika geologiska discipliner (t ex jordartsgeologi och hydrologi). SGU kan skräddarsy olika produkter som att svarar mot kundernas behov. En sådan anpassad produkt är en sårbarhetskarta över grundvattentillgångar.

En sådan temakarta kan framställas med digital teknik och baserar sig på befintliga jordartskartor, grundvattendata och hydrologiska kartor om det finns några. Dessa kartor visar var grundvatten finns och var det är sårbart genom infiltration av föroreningar. Dessa områden märks ut med olika färgkoder. På kartan finns även objekt som avfallsanläggningar, viktiga grundvattentäkter m m.

En sårbarhetskarta kan utnyttjas för att utforma regler för transporter av miljöfarligt gods, men även för att utforma åtgärdsprogram vid händelse av olyckor vid känsliga vägavsnitt m m. Dessa kartor blir då viktig information för hur och var åtgärderna ska sättas in.

Kartorna tas idag fram manuellt genom att man studerar olika kartor och annan information som borruppgifter m m. Sedan arbetar geologen med olika plastfilmer för att klassificera olika områden efter olika kriterier. Det finns inga fastlagda regler för hur arbetet ska utföras och ingen dokumentation som beskriver på vilka grunder det görs. Val av förklarande texter på kartan är inte heller standardiserat och även begreppsapparaten kan variera från karta till karta. Hur kartorna kommer att se ut beror mycket på den enskilde geologen.

För att utveckla ett datorstöd för framtagningen av sårbarhetskartor behövdes det en analys av själva framtagningsprocessen: vilken information som var nödvändig och vilka krav som ställdes på produkten. Dessutom behövde vi veta något som var ännu mer intressant i sammanhanget – på vilka grunder gjordes klassificeringen av ytorna på kartan och fanns det möjlighet att formalisera dessa grunder med hjälpa av ett antal regler?

5.3 Analysen

Det var den sistnämnda problematiken som vi inom projektet var mest intresserade av. För SGU:s del var det intressant att kartlägga framtagningsprocessen eftersom produktframtagningen skulle vara ett pilotprojekt för att ta fram ett digitaliserad produktionslinje. Frågor man ville ha svar på var vilka krav det skulle ställa på datorstödet och den lagrade informationen (Vilken information och vilken struktur skulle krävas?).

Under fallstudiens gång sammanföll mer och mer intresset från SGU:s sida och projektets, dvs att studera det tänkande som låg bakom klassificeringen och utbredningen av olika färgraster på sårbarhetskartan.

Projektet bedrevs så att experterna på framtagningen av dessa sårbarhetskartor bjöds in till SISU Studio. Under fyra modelleringstillfällen penetrerade vi problematiken. Vi som deltog från projektet hade ingen som helst kunskap om området.

Vi använde vår s k levande modelleringsvägg i SISU Studio under dessa analyspass. Vi gjorde ett antal analyser. Vi inledde med att göra en övergripande analys av informationsflöden som beskrev arbetsgången vid framtagning av en sårbarhetskarta.

Vi gjorde även en mer detaljerad flödesanalys av framtagningen av den arbetskarta som var underlag för den slutliga sårbarhetskartan. Under denna analys blev det uppenbart att problemlösningen inte var sekventiell, utan att beräkning av utbredning och klassificering av färgkoder skedde parallellt. Detta gjorde det mer komplicerat för oss att förstå och finna de regler som vi antog existerade i arbetet med att producera en sårbarhetskarta.

Vi gjorde en del begreppsanalyser av grundläggande geologiska begrepp men också av de kartbegrepp som fanns på sårbarhetskartorna och som fanns i det grundmaterial som geologen använde vid analysen.

Under analysens gång strävande vi efter att använda så många exempel som möjligt från kartorna.

Vid begreppsanalysen fann vi ganska snabbt att begreppen på kartan inte riktigt stämde överens med de olika geologernas uppfattning om begreppen. Å ena sidan hade de en inom-geologisk definition, å andra sidan hade de en definition som var kundorienterad. En slutsats som vi drog vid dessa analyser var att kartbegreppen var alldeles för vaga och många gånger förvirrade analysen. Det fanns uppenbarligen två nivåer som man ofta blandade ihop.

Senare gick vi in på det centrala problemet hur ett kartobjekt på en jordartskarta skulle översättas till ett annat ytobjekt på sårbarhetskartan. Vi var överens om att det fanns en funktion (eller regel om man så vill) som givet ett antal parametrar skulle ge en viss färgkod på sårbarhetskartan. Detta var inte något vi som modelleringsledare behövde komma fram till. Geologerna kom spontant fram till det och hade innan modelleringspasset själva försökt formulera funktionen. Vi såg att denna funktion skulle kunna brytas ned till ett antal regler.

Vi gav oss raskt in i problemet att formulera reglerna i strukturerat naturligt språk: OM mummel surr SÅ mummel surr. Detta i termer av den begreppsapparat som vi tagit fram.

Vi stötte på stora problem. Efter tre regler tappade vi överblicken och det var svårt att klara av att beskriva det som geologerna uppfattade som en och samma process, dvs att klassificera och samtidigt beräkna omfattningen av ett område. De begrepp vi hade gav inte heller mycket hjälp. Vi drog den slutsatsen att de modelleringstekniker vi hittills använt inte räckte till.

Detta inträffade under den tredje dagen. När vi stötte på dessa problem beslöt vi att göra följande tabell av de olika färgkoderna och de parametrar som vi uppfattade hade betydelse för färgkodningen:

Färg-koder på kartan	Media	Areal	Infilt-rations-benägen-het	Grundvatten-förekomst betydelse för kund (scenario = spridning av föroreningar på markytan)	Grundvatten-förekomst (Storlek)	Grundvatten-förekomst (uttagsmöjlig-heter, punktuttag,)
Röd	Grus		Mycket stor	Viktig	(Stor, liten)	(Goda)
Brun	Grus			Obetydlig	ingen	(Goda)
Oran-ge	Sand		Stor	Viktig	(Stor, Liten)	(Goda)
(Oran-ge)	Lera un-derlagrad av sand	(Större sam-man-hängande område)	Stor (tork-sprickor)	Viktig	(Stor, liten)	(Goda)
Grön	Lera	Större sam-man-hängande område	(Stor, Obetydlig) Tät	(Viktig, obetydlig)	(Stor, Liten, Ingen)	(Dåliga)
Ljus-grön	Lera un-derlagrad av sand	(Större sam-man-hängande område)	Tät	Viktig	(Stor, liten)	(Goda)
	Morän		Liten	Viktig, Obetydlig	Stor, liten, ingen	(Dåliga)
Raster	Häll		Stor, liten	Viktig, Obetydlig	Stor, Liten, Ingen	Goda-Dåliga Finns på kar-tan
Gul	Blandade-växlande förhållan-den		Växlande (stor, liten)	(Viktig, Obetydlig)	Liten	

Tabell 5 – Färgkoder och samband med geologiska och hydrologiska egenskaper

De färgkoder som finns nedtecknade till vänster i tabellen är de koder som typiskt förekommer på en temakarta och som finns förklarade i kartlegenden. De andra kolumnerna beskriver olika geologiska och hydrologiska egenskaper. Media beskriver vilken typ av jordart och eventuellt lager som ligger under den övre jordarten. Denna information kommer från t ex jordartskartor. Infiltrationsbenägenhet är en funktion av media. I tabellen finns tre typer av grundvattenförekomster. De visar

- Grundvattenförekomstens betydelse för kund och därmed dess betydelse i samband med olyckor med spridande av föroreningar. Detta kan vara väsentligt om det är en viktig vattentäkt.
- Storleken på förekomsten.
- Uttagsmöjligheter från en vattentäkt.

Nu fick vi lite bättre överblick men var fortfarande långt ifrån en lösning på problemet. Var det möjligt att analysera den tankeprocess som geologerna utförde då de byggde upp sårbarhetskartan? De var mycket medvetna om att de inte hade klart för sig på vilken grund de tog fram sårbarhetskartan. Hur skulle vi fortsätta analysen? Vi hade inte vid det här tillfället medvetet använt en viss teknik. Vi använde i stället tabellen för att ställa upp problemet på ett nytt sätt eftersom vi inte kom någon vart med att försöka formulera regler på naturligt språk.

Dag fyra anländer och vi får en ny idé. Vi vänder på tabellen och gör en beslutstabell!

RAD	BEGREPP	REGEL A	B	C	D	E	F	G	H
1	Lager. Ordning	1	1	1	1	1	1 & 2	1 & 2 & 3	1
2	Lager.In-filtrations-klass (m/h)	1 (> 10)	2 (<10 & >0.1)	2 (< 10 & >0.1)	3 (< 0.1)	4 (0)	min (E2,C 2)	min(B2,E2 ,C2)	
3	Lager. Akvifär	Ja	Nej	Ja	Ja	Nej	Ja	Ja	Ja
4	Lager. Vikt	Viktig	Ingen	Viktig	Viktig	Ingen	Viktig	Viktig	Viktig
5	Lager. Storlek	Stor	Ingen	Stor	Stor	Ingen	Stor	Stor	Liten
SLUT SATS	Ytobjekt. Typ: färg: raster (på Sårbarhets-kartan)	Röd	Brun	Orange	Ljus-orange	Grön	Ljus-grön	Ljusgrönt	Gult
SLUT SATS	Lager. Typ (x) refererar till lager. ordning	Grus	Grus och/ eller sand	Sand	Sand - silt	Lera	Lera(1) & Sand(2)	Grus-sand(1) & Lera(2) & Sand(3) Ordningen fås från kombina-tion av hy-drogeo-logiska och ??	Växlande (Morän, Häll, min-dre grus och sandomr., leromr.)

Tabell 6 – Tabell som visar färgbestämning för områden på sårbarhetskartan

Tabellen som visas ovan är direkt kopierad från det ordbehandlingsprogram som vi löpande använde under analysen. Vi presenterade tabellen i SISU Studio och gjorde alla förändringar under arbetets gång. Med hjälp av tabellen undersökte vi om det gick att förenkla problemen.

Förutom att vi systematiskt började tänka i termer av en beslutstabell där reglerna motsvaras av olika kolumner, gjorde vi även en del andra steg som var ett resultat av att problemet ställdes upp i en beslutstabell. Den beslutstabell som visas ovan utgör en version som utvecklats efter en tid av analys. Vi införde en mer precis formulering av begrepp som fick prefixet *Lager* och ett suffix efter punkten som namngav begreppet. Detta gjordes i syfte att ansluta sig till den begreppsmodell som fanns under utveckling i en databas.

Vi införde en mer precis bestämning av lagerordning som angav om det fanns underliggande jordarter eller ej. Det bestämde den infiltrationsklass som gällde för ett visst område. Infiltrationsklassen var en funktion av *Lager.Ordning* och *Lager.Typ*. Genom att införa infiltrationsklasser från 1-4 blev det möjligt att integrera flera faktorer så att de bildade ett begrepp. Infiltrationsklassen var en funktion av kornstorleken hos det medium som hade den minsta infiltrationsbenägenheten (i det fall då vi hade flera lager uttrycktes detta av funktionen *min(E2,C2)*).

I kartan fanns det en fixering vid begrepp som grus, sand, lera m m. Det skapade förvirring eftersom det rörde sig om sammansatta begrepp som både innehöll lagerordning och infiltrationsbenägenhet. Genom att ställa dessa begrepp åt sidan samt använda och införa begrepp som kornstorlek och infiltrationsklass, kunde vi bortse från de begrepp som kartan använde. Vi kunde gå till de begrepp som en geolog använder, vilket skapade en större klarhet i analysen av sambanden. Detta är ett exempel på hur man kan klargöra begreppsdiskussionen genom att ställa upp ett problem i en beslutstabell. Vi gick från kartbegrepp till de underliggande begrepp som användes för att ta fram kartan.

På vilket sätt bidrog uppställandet av problemet i form av beslutstabeller till att göra begreppsanalysen klarare? En hypotes är att beslutstabeller innehåller en beskrivning av olika fall i konkreta situationer och att det bidrar till att konkretisera analyssituationen och därmed klargöra vilka begrepp som är användbara och nyttiga i en problemlösningssituation. Att diskutera begrepp på ett abstrakt plan är mycket svårt och inte så värdefullt. Det är framför allt begreppen i dess användningssituation som är det intressanta (jämför med Wittgensteins "språkspel").

Grundvattenförekomst som hade tre olika betydelser i den tidigare tabellen fick här namnen: *Lager. Akvifär, Lager.Vikt, Lager.Storlek.*

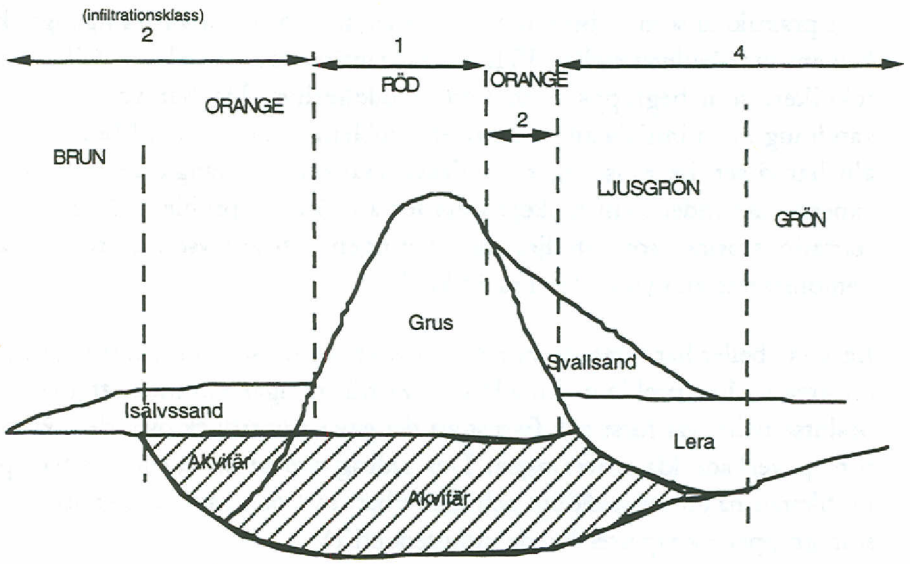
Lager.Akvifär var en benämning på existensen av grundvattenförekomst inom aktuellt område och kunde anta värdena **Ja** eller **Nej**. *Lager.Vikt* angav betydelse för kund och kunde anta värdena **Viktig** eller **Ingen vikt.** Slutligen angav *Lager.Storlek* storleken på grundvattenförekomsten och kunde anta värdena **Stor, Liten och Ingen.**

Efter att ha gjort uppställningen av beslutstabellen med de nya begreppsdefinitionerna, kunde vi gå vidare i vårt mål att finna enkla regler för att färglägga kartan efter ytobjektens sårbarhet. Vi kunde snabbt se att raderna 3-5 hade en intressant samvariation. I samtliga fall då *Lager.Akvifär* var **Ja** hade vi *Lager.Vikt* till **Viktig** och *Lager.Storlek* till **Stor**, utom i ett fall då vi hade växlande lagertyp av typen morän, häll, mindre grus och sandområde samt leromräde. I det senare fallet fick sårbarhetskartan alltid gul färg.

Vi hade alltså nu reducerat komplexiteten betydligt till att härleda en viss färgkod som en funktion av infiltrationsklass och förekomst av akvifär! Det intressanta med resultatet var att experterna inte var medvetna om det. Det var verkligen en aha-upplevelse för dem.

Men det återstod ett problem. Problemlösandet var inte endimensionellt utan hade faktiskt fler dimensioner. Dels måste geologen ta hänsyn till hur ytan såg ut på djupet, existens av lagertyper, lagerordning och förekomst av akvifär, dels måste själva utbredningen av ett visst område på den slutliga kartan beräknas utifrån dessa parametrars utbredning i ytterligare en dimension. Mycket av denna information som t ex utbredning av akvifär kan härledas från hydrologisk information samt utbredningen av lagerordningar från jordartskarta och kunskap om geologiska förlopp.

Här kände vi att beslutstabellen inte riktigt räckte till. Utifrån reduceringen av komplexiteten började vi tänka i fler dimensioner. Vi startade ett ritprogram på väggen och började att rita upp ett tvärsektion av markytan där vi införde de olika begrepp som vi utnyttjat i beslutstabellen. Vi ritade upp följande bild:



Figur 14 – Tvärsektion av jordyta som illustrerar tillämpning av regler i en beslutstabell

Denna bild sammanfattade på ett elegant sätt beslutstabellen. Vi kan ta ett exempel. Längst till höger på bilden har vi ett område som har infiltrationsklass fyra. Enligt beslutstabellen (tabell 6) innebär det en infiltrationshastighet på 0[1] meter per timme (m/h). Men hur ska vi förklara att färgen skiftar från ljusgrön till grön?

[1] I detta sammanhang försumbar.

Eftersom färgkoden var en funktion av infiltrationsklassen och förekomsten av en akvifär är frågan enkel att svara på. Att infiltrationsklassen är fyra får vi fram genom att använda funktionen som finns för regeln i kolumn F, *min(E2,C2)*. Den funktionen betyder att infiltrationsbenägenheten för en lagerordning är lika med den jordart som släpper igenom minst. I vårt fall skulle det se ut på följande vis: *min(infiltrationsbenägenhet(Svallsand), infiltrationsbenägenhet(Lera))*.

Den infiltrationsbenägenhet som är resultatet översätts sedan till en infiltrationsklass för att bestämma färgraster på en lämplig granularitet. I vårt fall har *Lera* den minsta genomsläppligheten och bestämmer därmed infiltrationsklassen till fyra. Övergången från ljusgrön till grön får vi enkelt genom att matcha infiltrationsklass och förekomst av akvifär[1]. Då akvifären upphör övergår färgkoden till grönt.

5.4 Sammanfattning och slutsatser

Det praktikfall som vi beskrivit är ett exempel på hur man framgångsrikt kan använda beslutstabeller. Vi har också använt traditionella modelleringstekniker, som begrepps- och flödesmodellering. De har varit till användning i den initiala analysen av ett problem av denna art. Men framför allt har vi sett begränsningar hos dessa tekniker i att fånga de regler som experter använder i sin tankeprocess för att lösa ett problem. Nya representationsformer som stödjer formuleringen och analysen av regler har demonstrerat sin nytta i detta praktikfall.

Beslutstabeller har visat sig vara en utmärkt teknik som fungerar också för att kreativt lösa problem. En viktig orsak till framgången med att använda beslutstabeller var först och främst att det gav en överblick över de faktorer som påverkade klassificeringen. Det bidrog dessutom till att ställa upp problemet på ett överskådligt och hanterbart sätt och gav en referenspunkt som gruppen av experter kunde diskutera utifrån.

Den andra faktorn som bidrog till en skärpning av begreppsapparaten var att beslutstabellen bygger på konkreta situationer, vilket gör det enklare att fokusera på faktiska fall. Under analysarbetet försökte vi också fokusera så mycket som möjligt på de konkreta kartorna som geologerna hade med sig, eftersom vi kände till svårigheten för experter att få tillgång till sin egen kunskap på ett direkt sätt.

---

[1] Observera att detta görs i flera dimensioner för att man ska komma fram till en yta som har ett visst färgraster.

Att vara analytisk innebär att man fjärmar sig från det enskilda. Därmed förlorar experten accessen till sin egen kunskap innan den har medvetandegjorts. I en konkret problemlösningssituation får experten access till sin problemlösningskunskap men kan inte rapportera och analysera den samtidigt. Därför måste en metod för kunskapsinhämtning både ha analytiska moment och konkreta problemlösningssituationer. Beslutstabellerna bidrog till att ställa upp de olika konkreta fallen på ett gripbart sätt samtidigt som de ger analytiska redskap.

Beslutstabellerna bidrog även till att reducera komplexiteten. Vi kunde lättare se beroenden mellan begrepp samtidigt som vi kunde se att vissa begrepp var sammansatta eller funktioner av varandra. Genom att underlätta upptäckten av samvariationer kunde vi skära bort irrelevanta begrepp och parametrar.

Förutom att bidra till att reducera komplexitet kan beslutstabellerna även utnyttjas för att testa om alla situationer har täckts (i vårt fall kunde vi se om alla infiltrationsklasser var med etc) och om det fanns några motsägelser. I detta praktikfall utnyttjade vi dock inte beslutstabeller för att kontrollera fullständighet eller motsägelser.

Förutom beslutstabeller har vi använt ett ritprogram för att illustrera och konkret testa resultatet från beslutstabellen i en representation som geologer är vana vid: en grafisk modell som visar ett tvärsnitt av jordytan. Användningen av denna grafiska modell var effektiv för att testa om beslutstabellens innehåll var korrekt.

Till slut vill vi speciellt nämna SISU:s levande modelleringsvägg och dess betydelse för att vi skulle kunna använda olika representationstekniker. Den var värdefull för de resultat som vi åstadkom.

6 Guidelines for Rule Formalization

(från Temporas metodhandbok)

6.1 Negation

Single Occurrences of Variables in the Consequence

Any variable appearing only once in the then part of a rule must have appeared at least once outside the scope of a not connective in the if or when part of the rule. This is because we do not what to specify a consequence which contains a variable which can range over all possible values, but instead is restricted to some domain specified in the trigger or condition of the rule.

Example

Create a new invoice with a unique invoice number

if not sometime_in_past invoice has number.N

then invoice has number.N

The above rule specifies an infinite set of numbers which N ranges over, being the set of integers not already used as invoice numbers. The ERL code violates the methodological rule, since N only appears in a positive sense in the then part of the rule. What gives the value of N in the consequence must be stated in a positive sense in the condition, for example:

if N is maximum bagof(V,sometime_in_past invoice has number.V)

then invoice has number.N

This gives what are probably the intended semantics; that a unique invoice number is generated for the new invoice.

Applying Negation to Formulae

Care should be taken when applying negation to anything other than a simple predicate, since many users forget that de Morgan's laws apply:

not (a or b) ≡ not a and not b

not (a and b) ≡ not a or not b

Example

not (person has salary.S and NewS=S*1.1) ≡

not person has salary.S or not NewS=S*1.1

Whilst this may seem obvious, it does produce what can be unexpected results when applied to ERT access expressions involving more than two classes, since such ERT access expressions are in effect a conjunction of simpler ERT access expressions:

person works_for company [has name.N, has address.A] ≡

exists C for_which person works_for company.C and

company.C has name.N and

company.C has address.A

The negation holds if any of the constituent expressions does not hold:

not (person works_for company [has name.N, has address.A]) ≡

not person works_for company.C or

not company.C has name.N or

not company.C has address.A

6.2 Sets and Bags

The use of the aggregate operators count, average, sum, minimum and maximum require considerable care when applied to tuples and sets generated by bagof and setof. The table below summarises the discussion as to the usefulness of applying a particular aggregate operator to a tuple or bag.

	setof	bagof
count	yes	yes
average	?	yes
sum	?	yes
minimum	no	yes
maximum	no	yes

In the table, stating *yes* means that the operator is useful in many situations, *?* that it is of limited use, and *no* that it should not be used.

• A setof finds only distinct values, and thus should be used where we wish to aggregate the different occurrences of that value. For example, count setof(N, person has name.N) finds the number of distinct names that exist in the database. We do not permit the use of minimum and maximum on sets of values since it results in a semantically equivalent formula to that using bags, but is computationally less efficient.

• A bagof finds all occurrences of values, whether duplicated or not. It thus should be used where we wish to aggregate all value instances in the database. For example, count bagof(N, person has name.N) finds the number of persons with names in the database.

6.3 Using Aggregate Operators on Surrogates, Numerical Data and Textual Data

The aggregate operators are only meaningful when applied to data of a specific type. For instance, it makes sense to find the average of a bag or set of numbers, but not for a bag or set of textual names or surrogates for entities. The combinations permitted are summarised in the following table:

	numeric data	textual data	surrogates
count	yes	yes	yes
average	yes	no	no
sum	yes	no	no
minimum	yes	no	no
maximum	yes	no	no

6.4 Existence of Classes

The ERT Access expressions of the ERL allow for finding surrogates for entities and values from the ERT database. When checking that a class is not set to a certain surrogate/value, there are many different ways of checking the presence of the different surrogates/values:

- another_class.Y relationship class.X holds for values of X which are instances of class and are related to instance Y of another_class
- not another_class.Y relationship class.X holds for values of X which are not instances of class, and may hold even if class does not have any instances, or the instances are not related to another_class
- another_class.Y relationship class=\=X holds for values of X which are not instances of class, but only if class is related to instance Y of another_class
- not another_class.Y relationship class holds if there are no instances of class related to instance Y of another_class, or there is no instance Y of another_class
- another_class.Y and not another_class.Y relationship class holds if there are no instances of class related to instance Y of another_class

The rules to follow in creating ERT access expressions are:

- State in positive (i.e. non negated) ERL expressions all classes you need to find the value of, and associate a variable with them (i.e. write <class>.<variable>).
- State in positive ERL expressions without variables all classes that must exist, but you are not interested in the value for.
- State in negative (i.e. inside the scope of not, or where possible using =\=) ERL expressions for all classes you want to ensure do not have a value, and associate with the class a variable to hold that value.
- State in negative ERL expressions without variables all classes that must not have an instance.

Example

Find all invoices which are for a person, and have an amount other than zero

invoice.I [for person, has amount=\=0]

Example

Find all invoices and their amounts, where the invoice is not for a person, and has an amount greater than 10,000

invoice.I has amount.A > 10000

and not invoice.I for person

6.5 Surrogates

When accessing a ERT entity class, it is often useful to be able to speak about the instances of that entity without having to identify them by their attributes in all positions they are used. For example, we may want to give all people who work for a company Spares Ltd a salary increase of £100:

if person [works_for company."Spares Ltd", has salary.S]

then person [works_for company."Spares Ltd", has salary=S+100]

However, written as above, it is not clear if we are talking about the same or different persons in the condition and consequence of the rule. Indeed, unless we specified an attribute which is unique to the person (such as their national insurance number), it would be correct to interpret the above rule as creating a new employee with salary S+100 for all the salaries of existing employees of Spares Ltd.

For this reason, we allow the use of surrogates in the ERL, where not only value classes, but also entity classes can have a variable associated with them. However, the value stored in the variable is only meaningful is equated with a variable also associated with either the same entity, its super-classes, or its sub-classes.

The rule to increase salaries may now unambiguously be written as:

if person.P [works_for company."Spares Ltd", has salary.S]

then person.P has salary=S+100

The rules for naming surrogates in the consequence of a rule are as follows:

(i) A rule consequence containing an entity class with a variable associated with it, where the variable also appears in the condition or trigger of a rule, or the variable is equated with another variable with appears in the condition or trigger of the rule, will only effect existing instances of the entity class.

(ii) If (i) does not hold for an entity class, the rule will create new instances, and will not effect existing instances.

(iii) If an entity class appears inside the scope of a negation, it only makes sense that existing instances of the entity class are to be removed, as so (i) must be obeyed for such classes.

6.6 Quantification

All free variables in ERL rules are universally quantified, i.e. the rule is said to hold for all possible values that the variables may take. This will normally lead to the desired semantics for a rule, since we naturally state rules as being things which must be true (the consequence) for all instances of a given situation (the combination of trigger and condition).

Example

if person.X works_in city."Stockholm" then person.X has salary>10000

States that for all possible values that may exist in the ERT model, if that value is the surrogate of a person working in Stockholm, then the same person has a salary greater than 10000.

However, some situations dictate the use of a different type of quantification, called *existential quantification* where we want to ensure that there exists some value for which the rule holds, but do not require that it holds for all possible values.

Example

if exists Y for_which person.X works_in city.Y and person.X works_in city=\=Y

then person.X has salary>10000

States that for all possible values that may exist in the ERT model, if that value is the surrogate of a person working at least two different cities, then the same person has a salary greater than 10000.

To explain the situations in which we need to use explicit quantification, we will briefly look at the logical semantics of rules more formally.

if Φ then $\Psi \equiv \forall$ X,Y,Z,... . $\Phi \rightarrow \Psi$ where X,Y,Z ... are the free variables of F,Y

$\equiv \forall$ X,Y,Z,... . $\neg\Phi \vee \Psi$

We can see now that rules can model uses of the universal quantifier, since we can rewrite any expression into the form $\neg\Phi \vee \Psi$ (trivially $\Psi \equiv \neg$TRUE $\vee \Psi$). A procedure called skolemisation allows all existential variables outside the scope of a negation to be replaced by an unknown constant (in effect a normal variable which we ignore the value of). Thus if $\exists$ X,Y,Z ... . Φ appears outside the scope of negation, we may replace it by Φ, since we do not read the values of variables outside of an ERL rule. Furthermore, negated existential quantification is equivalent to universal quantification $\neg\exists X\neg \equiv \forall X$. Using skolemisation, the above example becomes:

if person.X works_in city.Y and person.X works_in city=\=Y

then person.X has salary>10000

In short, most situations do not require the use of the exists or for_all quantifications, and they should only be used with great care. The most likely requirement for their use is inside the scope of bagof operator where the number of times a formula holds is significant.

Example

if Population=count bagof(X,person.X works_in city.Y)

then population_count(Population)

if Population=count bagof(X,exists Y such_that person.X works_in city.Y)

then population_count(Population)

The first rule will generate a population_count flow for each city, whilst the second rule generates a population_count for all cities combined.

Time in Databases

When querying and creating objects in the ERT, the following guidelines will provide accurate modelling of most situations:

- All non-T-marked objects should be treated as in any standard database system.
- A T-marked object which can be regarded as recording an event in the UoD (universe of discourse) should be

  - queried (i.e. appear in the condition) using the temporal connectives to search over time for instances of the objects at different times.

  - asserted (i.e. appear in the consequence) without using any temporal operator or connective, so that it has only the current time associated with it.
- A T-marked object which can be regarded as recording the existence of a real object in the UoD should be

  - queried (i.e. appear in the condition) without using temporal connectives, so that only objects which exist at the time being queried are accessed. Querying using temporal operators should only be made to access out-of-date information, such as old customers, invoices, addresses *etc.*

  - asserted (i.e. appear in the consequence) using temporal operators or connectives, so that it its period of validity in the UoD is established. Either a limited period of validity should be specified (by *object* for [...,...]) or the period of validity made indefinite (always_in_future *object*), and some rule given to deal with the ending of the period of validity.

- HT-marked objects should be treated like T-marked objects, except that care must be taken to establish and query relationships in the 'correct' direction, since the period of validity of such relationships differs in the direction followed.

It should be noted that the above list forms only a guideline to the most common situations - other situations may require different use of the temporal operators and connectives, but experience from case studies indicate that these are the exception to the general rules. The example in Figure 5-2 gives an ERT model for a database holding the invoice records for several companies customers. We treat the various objects as follows:

name, value and town are non-temporal (they exist over all time) and can be queried and asserted disregarding the time, since their period of validity is that of the database itself.

company, customer and invoice can be treated as objects in the UoD, since they are all objects in the real business. payment is better treated as an event, since its significance is in being an occurrence in the UoD, and not in its presence in any physical record.

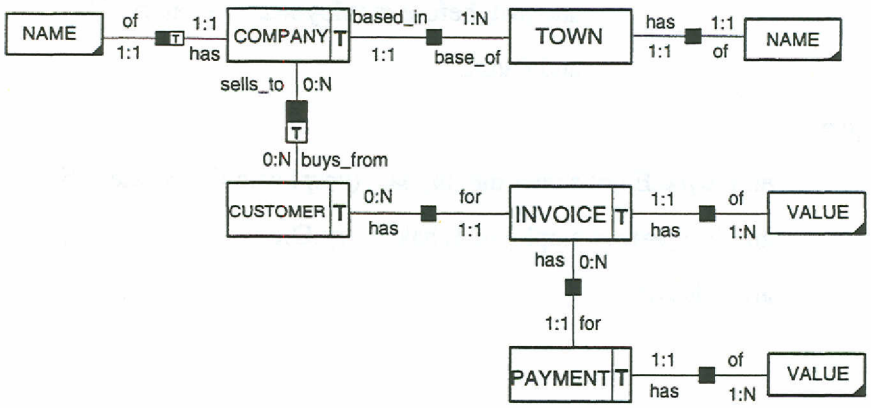


Figure 6-2 – Customer invoices ERT model

Rules that create customers and payment records then take the following form:

when new_customer(Name)

then always_in_future customer has name.Name

% define period customer will last for

when payment(Invoice,Amount)

if invoice has number.Invoice

then invoice has payment of value.Amount

% payment only holds for current tick

As an example of rules that end the period of validity of an invoice:

when invoice.I has payment.Pay

if sum bagof(OldPay, sometime_in_past invoice.I has OldPay) + Pay > value of invoice.I

then always_in_future not invoice.I

When building queries involving time, avoid using the explicit time operators, since invariably they result in rules less easy to read, and more difficult to modularise than the equivalent queries using implicit time connectives. Building a query using the implicit time connectives will normally be done 'bottom-up', by identifying the 'atomic' events and conditions of the query and joining them together.

Example

Find all employees for whom their salary has never decreased.

decrease in salary: employee.E has salary.S
and just_before employee.E has salary.Old
and Old>S

query:

employee.E not sometime_in_past (employee.E has salary.S
and just_before employee.E has salary.Old
and Old>S)

Appendix – ERL Syntax and Semantics

A.1 ERL Semantics

Program

At the outer most level, an ERL program consists of a set of *rules*, where the ordering of rules given is not significant. Execution of the system will pass through a series of discrete *ticks*, each of which represents a fixed period of time, making the rules hold as a consistent set at each tick. In the runtime system, the computational semantics of rules must be considered, and these are outlined in Section B.2.

Elements

The basic *elements* which the ERL rules deal with are:

- ERT entity classes, value classes and relationships
- *time point*, which specifies some *tick* in the flow of time recorded in the database.
- *time interval*, which specifies some contiguous series of *ticks*.
- *PID flows*, which name tuples of ERT objects or groups of ERT objects

Rules

Four basic forms of rule are permitted, as listed below, where e_1, e_2 and e_3 are any valid expressions of the language:

(1) e_1

The expression e_1 must hold at any moment or *tick* during the execution of the system.

(2) if e_1 then e_2

If expression e_1 holds at any tick for some set of variable substitutions, then e_2 must also hold for the same variable substitutions.

(3) when e_1 then e_2

If expression e_1 holds at any tick for some set of variable substitutions, but did not hold if it had been evaluated at the previous moment in real time, then e_2 must also hold for the same variable substitutions.

(4) when e_1 if e_2 then e_3

If expressions e_1 and e_2 hold at any tick for some set of variable substitutions, and for which e_1 did not hold if it had been evaluated at the previous moment in real time, then e_3 must also hold for the same variable substitutions.

An optional else clause may be added to rules of the form in (2) and (4), which obeys the following equivalence rules:

if e_1 then e_2 else e_3 ≡
- if e_1 then e_2
- if not e_1 then e_3

when e_1 if e_2 then e_3 else e_4 ≡
- when e_1 if e_2 then e_3
- when e_1 if not e_2 then e_4

Free variables are universally quantified across the rules, and no free variable should appear only in scope of a negation. Free variables that appear in an else expression must also appear in the when expression.

Access and Selection of ERT Sub-Model

We include the arithmetic relationships =, =\=, <, >, =< and >= as notionally part of any ERT model, along with the data-types ('values') of the model.

e_1	Holds for each instance of class *el* in the ERT.
$e_1.X$	Holds for variable X bound to each instance of class *el* in the ERT.
$e_1\ r_1\ e_2$	Holds for each instance of class/value *el* and ERT access e_2 for which there is a instance of relationship rl between the instance of e_1 and the leading instance of e_2.
$e_1.X\ r_1\ e_2$	As previous case, but also binds variable X to the instance of e_1.
$e_1\ [r_1\ e_2,\ r_2\ e_3]$	Equivalent to exists X such_that $e_1.X\ rl\ e_2$ and $el.X\ r2\ e_3$.
$e_1.X\ [r_1\ e_2,\ r_2\ e_3]$	Equivalent to $el.X\ rl\ e2$ and $el.X\ r2\ e_3$.

Logical Propositions

true	Always true
false	Always false

Logical Connectives

e_1 and e_2	Both e_1 and e_2 hold for some set of variable substitutions.
e_1 or e_2	Either e_1 or e_2 holds for some set of variable substitutions.
not e_1 e_1	does not hold for the set of variable substitutions applied to the expression which the connective is part of.
only_one_of (e_1 , e_2, ...)	Holds if exactly one of e_1 , e_2, ... holds.

Temporal Connectives

At any time during execution, the temporal database will have stored facts not only about the present time, but also about the past and future. We may view this as a sequence of databases each associated with some *tick*, and may query any of these databases to obtain information. ERL rules are always evaluated with respect to the database that corresponds to the real time the query is posed, any temporal connectives within the rule have the effect of transposing the time at which the part of the ERL rule within the scope of the connective is evaluated into a past or future time, using the information in the temporal database to simulate databases at those times. The semantics for the various temporal operators speak of the expressions being evaluated at certain ticks, as a shorthand for the database associated with that tick.

temporal connective	**condition**
at_any_time e_1	e_1 holds at some tick.
sometime_in_past e_1	e_1 holds at some tick before the present tick. At the first tick sometime_in_past is false.
sometime_in_future e_1	e_1 holds at some tick after the present tick. At the last tick sometime_in_future is false.
always_in_past e_1	e_1 holds for all ticks before the present tick. At the first tick always_in_past is true.
always_in_future e_1	e_1 holds for all ticks after the present tick. At the last tick always_in_future is true.
just_before e_1	e_1 holds for the tick previous to the present tick. At the first tick just_before is false.
just_after e_1	e_1 holds for tick following the present tick. At the last tick just_after is false.
e_1 until e_2	e_1 holds for all ticks after the current tick, up to and including the tick when e_2 holds. At the last tick until is always false.
e_1 since e_2	e_1 holds for all ticks before the current tick, since and including the tick when e_2 holds. At the first tick since is always false.

Set Expressions

bagof(X, e_1)	For each instantion of the free variables in e_1 excluding X, returns an arbitrarily ordered list <{v_1,v_2, ...}>, where v_1, v_2, ... are the instantions of X for which e_1 holds.
setof(X, e_1)	For each instantion of the free variables in e_1 excluding X, returns the set {v_1,v_2, ...}, where v_1, v_2, ... are the instantions of X for which e_1 holds.

Time

<time_point>	::=time> on <date> | <time>
<time>	::=<hour>: <minute>: <second>
<date>	::=<day> :: <month> :: <year>

A time point identifies a certain tick during any day. A time point given without its date field is assumed to take the date of the day of execution of the ERL formula. A set of shorthand names are provided to avoid the need to calculate explicitly commonly used time point values. Examples of such shorthands are start_of_today which returns the first time point of the current day, and end_of_week which returns the last tick of the current week. These are always calculated with respect to the time of execution of the query.

A time point is merely a shorthand for a time interval on one tick duration, i.e. t_1 is equivalent to $[t_1, t_1]$. If an interval of greater than one tick is used in the context of a time point then a runtime error will occur.

time_is t_1

If t_1 *is* a free variable, then time_is finds the current time, represented as a time point, otherwise it uses the value of t_1 to check against the current time. It is permitted to supply either a full time point, or just the *time* field of a time point. In the latter case the date is taken to be that of the day of execution.

time_point_of e_1

Returns the value that t_1 would take in e_1 time_is t_1

e_1 at t_1

The expression e_1 is checked to hold at time point t_1 , not the current time.

e_1 after t_1

The expression e_1 holds t_1 time units after the current time. A run-time error will occur if the time so referenced is after the maximum time allowed by the implementation.

e_1 before t_1

The expression e_1 holds t_1 time units before the current time. A run-time error will occur if the time so referenced is before the minimum time allowed by the implementation.

$t_1 >> t_2$

If t_1 is a time point, adds the time unit t_2 to t_1 , giving a new time point. A run-time error occurs if a time point after the last allowed by the implementation is produced.

$t_1 << t_2$

Subtracts t_2 time units from time point t_1, giving a new time point. A run-time error occurs if a time point before the first allowed by the implementation is produced.

$t_1 + t_2$	Add t_2 time unit to time unit t_1. It is intended that the pseudonyms hour, day, *etc* are used instead on the underlying integer tick values, since the real time associated with a tick will vary between implementations. Similar semantics are associated with the use of -.
t_1 / n_2	Divide time unit t_1 into n_2 equal time units. Similar semantics are associated with the use of *.

Intervals

$[t_1, t_2]$ names the contiguous series of ticks starting at t_1, and lasting up to and including t_2. As a convenient shorthand for such intervals, *interval functions* may be used, where the values of t_1 and t_2 are derived from the current time and the named period, by associating t_1 with the first tick of the named period, and t_2 with the last tick of the period. For example, tomorrow evaluated any time on Monday, would give t_7 the value of the first tick of Tuesday, and t_2 the last tick of Tuesday.

e_1 during $[t_1, t_2]$	Finds variable substitutions for which e_1 holds during any tick t and t satisfies $t_1 < t < t_2$
e_1 for $[t_1,t_2]$	Finds variable substitutions for which e_1 holds for all ticks t that satisfy $t_1 < t < t_2$
$[t_1, t_2]$ >> <time_unit>	Equivalent to [t_1 >> <time_unit>, t_2 >> <time_unit>]. If either t_1 or t_2 are made to come after the first time supported by the implementation a run-time error will occur.
$[t_1, t_2]$ << <time_unit>	Equivalent to [t_1 << <time_unit>, t_1 << <time_unit>]. If either t_1 or t_2 are made to come before the first time supported by the implementation a run-time error will occur.

$[t_1, t_2]$ <interval_relationship> *[sl,s2]*

The following table gives the conditions for the relationship holding:

Relationship Name	*Conditions*
runs_into	$t_1 < s_1 \leq t_2 < s_2$
coends	$t_1 > s_1$ and $t_2 = s_2$
contains	$t_1 < s_1$ and $s_2 < t_2$
beginning_of	$t_1 = s_1$ and $t_2 < s_2$
equals	$t_1 = s_1$ and $t_2 = s_2$
ends_before	$t_2 < s_1$
ends_at_start_of	$t_2 = s_1$
ends_during	$s_1 \leq t_2 < s_2$
ends_with	$t_2 = s_2$
ends_after	$t_2 > s_2$
starts_before	$t_1 < s_1$
starts_with	$t_1 = s_1$
starts_during	$s_1 \leq t_1 \leq s_2$
starts_at_end_of	$t_1 = s_2$
starts_after	$t_1 > s_2$

A.2 Runtime Semantics

Choosing the classification of a rule alters its computational semantics. Execution of rules is performed in *query-execute* cycles where

- for *constraint* or *derivation* rules, the consequence (i.e. the then part of the rule) is made to hold in the same time as the condition (i.e. the if part of the rule).
- for *action* rules, the consequence is made to hold in the next execution cycle after the cycle in which the condition holds.

Action rules are clustered in the PID model into transactions. The flows produced by any action rules which lead out of the transaction will be delayed until the cycle in which there are no more action rules within the transaction to execute.

A.3 ERL Syntax

The syntax is described using a modified Backus-Naur form rules, with [square] brackets indicating optional sections, and {braces} optional repeating sections. All terminals are given in sans serif typeface. We also give a short description of some additional restrictions that can be placed on ERL expression during verification and validation.

ERL Rules

<ERL_rule> ::= when <exp> if <exp> then <exp> [else <exp>] I
when <exp> then <exp> I
if <exp> then <exp> [else <exp>] I
<exp>

Comments

Any text on a line after a % character will be ignored

ERT Class Instances

<ERT_class_instance> ::= <ERT_class> | <string> | <time_point> | <arithmetic_exp>

<ERT_class> ::= <ERT_id_name> [.<atomic_element>]

ERT_id_name> ::= <small_alpha> { <alpha> | _ }

<string> ::= " { <ASCII char> } "

<integer> ::= <digit> { <digit> }

<float> ::= <integer> . <integer>

<variable> ::= <capital_alpha> { <alpha> | _ }

<atomic_element ::= <string> | <integer> | <float> | <variable> | <time_point>

<element> ::= <atomic_element> | <tuple> | <set> | <interval>

<element_list> ::= <element> {, <element> }

<structure> ::= <tuple> | <set> | <variable>

<arithmetic_exp> ::= <integer> | <float> | <variable> | <arithmetic_exp> <arithmetic_binop> <arithmetic exp> |<arithmetic_unop> <arithmetic_exp> |<time_unit> / <time_unit> | (<arithmetic_exp>) |number_of <structure> | minimum <structure> |maximum <structure> | sum <structure> |average <structure>

<arithmetic_binop> ::= + I - I * I / I mod

<arithmetic_unop> ::= + | -

<capital_alpha> ::= **A** | **B** | ... | **Z**

<small_alpha> ::= **a** I **b** I ... I **z**

<alpha> ::= <capital_alpha> | <small_alpha>

<digit> ::= **O** | **1** | ... | **9**

Access and Selection of an ERT Sub-Model

Instances of *ERT_class_instance* which are of the *ERT_class* type must be verified against the ERT model. For all instances of *ERT_class_instance* it must be checked that the *ERT_relationship* used just after is valid for the class type. In particular, using relational operators (= ,<, >, =<, >=, =\=) between an entity and any other object does not have any valid semantics, since there is no ordering defined for entities. The exception is that we may use = and =\= between instances of the same entity class.

```
        <exp>  ::=   <ERT_access>

<ERT_access>  ::=   <ERT_class_instance> |
                    <ERT_class_instance> <ERT_relationship>
                          <ERT_access> |
                    <ERT_class_instance>
                     [ <ERT_relationship> <ERT_access>
                          {, <ERT_relationship> <ERT_access> } ]

<ERT_relationship>::= <ERT_id_name> | = | < | > | =< | >= | =\=
```

Quantification

```
        <exp>  ::=   for_all <ERT_access> it_follows_that <exp> |
                     exists <ERT_access> such_that <exp>
```

Set Expressions & Tuples

The number of variables appearing in the two instances of *set* that appear in the binary set operators must be equal. Note that allowing the first two rules of <exp> means that the ERL becomes a non first-order language, since variables may be bound to sets. It is intended that this construct be used to model the use of list structures in Prolog, where assigning a set to a variable is modelling the use of setof/3. Only instances of *ERT_id_name* not used in the ERT model may be used for the names for tuples.

```
<exp>            ::=   <structure> == <structure> |
                       <element> member_of <structure>
<set_variable_list>  ::=   <variable> {, <variable> } | <tuple>
<set_binop>      ::=   union I intersect I disjoint
<set>            ::=   { <set_variable_list> for_which <exp> } |
                       [ <ERT_id_name> ] { [ <element_list> ] } |
                       setof( <structure>, <exp>) |
                       (<set>) |
                       <set> <set_binop> <set>
<tuple>::=             <{ <set_variable_list> for_which <exp> }> |
                       [ <ERT_id_name> ] <{ [ <element_list> ] }> |
                       [ <ERT_id_name> ] <{
                       <element_list>|<structure> }> |
                       bagof( <structure>, <exp> ) |
                       ( <tuple> ) |
                       project( <tuple>, <tuple>, <structure> )
```

Flows

Only instances of *ERT_id_name* not used in the ERT model and not equal to only_one_of, time_point_of and interval_of may be used for the names for flows.

```
<exp>            ::=   <ERT_id_name>( [ <element_list> ] )
```

Logical Propositions

```
<exp>            ::=   true I false
```

Logical Connectives

```
<exp>            ::=   <exp> <logical_binop> <exp> |
                       <logical_unop> <exp> |( <exp> ) |
                       only_one_of( <exp> {, <exp> } )
<logical_binop>  ::=   and I or
<logical_unop>   ::=   not
```

Implicit Time Connectives

```
<exp>  ::=   <temporal_unop> <exp> |
             <exp> <temporal_binop> <exp>
<temporal_unop>::=   at_any_time I sometime_in_past I
                     sometime_in_future I always_in_past I
                     always_in_future I just_before I just_after
<temporal_binop>::=  until I since
```

Explicit Time Operators & Connectives

<exp> ::= <exp> <interval_test> <interval> |
<interval> <interval_relationship> <interval>

<interval> ::= [<time_point>, <time_point>] |
<interval> >> <time_unit> | <interval> <<
<time_unit> |
<time_point> | <variable> | <date> |
interval_of (<exp>) |
today I yesterday I tomorrow I
this_week I last_week I next_week I
this_month I last_month I next_month I
this_year I last_year I next_year

<interval_test> ::= during I for

<interval_relationship> ::= runs_into I coends I contains I beginning_of I
equals I ends_before I ends_at_start of I
ends_during I ends_with I ends_after I
starts_before I starts_with I starts_during I
starts_at_end_of I starts_after

Time Points

Arithmetic manipulation of *time_unit* involving ground terms may be checked for semantic correctness. For example, is does not make sense to have hour - day as a *time_unit*, but day - hour gives 23 hours.

<exp> ::= time_is <time_point> | time_is <time> |
<exp> at <time_point> | <exp> after <time_unit> |
<exp> before <time_unit> | day_is <day_name> |
week_number_is <week_name>

<time_point> ::= <time> on <date> | <time> |<variable> |
<time_point> >> <time_unit> |
<time_point> << <time_unit> |
time_point_of (<exp>) |
start_of_today | start_of_yesterday |
start_of_tomorrow | start_of_this_week |
start_of_last_week | start_of_next_week |
start_of_this_month | start_of_last_month |
start_of_next_month | start_of_this_year |
start_of_last_year | start_of_next_year |
end_of_today | end_of_yesterday |
end_of_tomorrow | end_of_this_week |
end_of_last_week | end_of_next_week |
end_of_this_month | end_of_last_month |
end_of_next_month | end_of_this_year |
end_of_last_year | end_of_next_year

<time>::= <hour>: <minute>: <second>

<date> ::= <day> :: <month> :: <year>

<time_unit> ::= <time> | <variable> | <integer> | <float> | second |
minute | hour | day | week | seconds | minutes |
hours |days | weeks | <time_unit> + <time_unit> |
<time_unit> - <time_unit> |
<arithmetic_exp> * <time_unit> |
<time_unit> * <arithmetic_exp> |
<time_unit> / <arithmetic_exp>

<day_name> ::= <variable> | "Monday" | "Tuesday" |
"Wednesday" |"Thursday" | "Friday" |
"Saturday" | "Sunday"

<week_name> ::= <variable> I 01 1 ... I 53

<second> ::= 00 | 01 | ... | 59 | <variable>

<minute> ::= 00 | 01 | ... | 59 | <variable>

<hour> ::= 00 | 01 | ... | 23 | <variable>

<day> ::= 01 | 02 | ... | 31 | <variable>

<month> ::= 01 | 02 | ... | 12 | <variable>

<year> ::= 1900 | 1901 | ... | 9999 | <variable>

Precedence of Operators

The following table lists the operators of the ERL in decreasing order of precedence. Thus, operators placed higher up the table bind more tightly than those lower down.

Group of Operators	Associatively
<arithmetic_binop> << >>	left to right
on	left to right
<set_binop>	left to right
= =< >= < > =\=	right to left
<temporal_unop>	right to left
<temporal_binop>	right to left
<logical_unop>	right to left
and	right to left
or	right to left
<interval_test> at after before	left to right

Referenser

[Frøkjær, 1989 (1)] E. Frøkjær. (1989). Controversial Issues of Expert Systems in Public Administration. In E. a. Th. M. Snellen (Ed.), Expert Systems in Public Administration Elsevier Science Publishers B.V. (North-Holland).

[J-J Ch. Meyer, 1993 (2)] R. J. W. J-J Ch. Meyer. (1993). Deontic Logic in Computer Science. John Wiley & Sons,

[Johnson, 1987 (3)] N. E. Johnson. (1987). Mediating representations in knowledge elicitation. First European Workshop on Knowledge Acquisition for KBS. Reading University:

[Kontio, 1989 (4)] L. Kontio. (1989). A Graphical Framework for Knowledge Acquisition and Representation. Fifth Australian Conference on Applications of Expert systems. Sydney, Australia:

[Nilsson, 1970 (5)] B. Nilsson. Beslutstabeller – programlogik i software and hardware (23.2). Statistiska Central Byrån (1970).

[Persson, 1971 (6)] S. Persson. (1971). Beslutstabeller - 1. Beskrivning av regelsamband . Stockholm: Studentlitteratur Lund, Akademisk Forlag København.

[R.J. Wieringa , 1993 (7)] J.-J. C. M. R.J. Wieringa . (1993). Application of Deontic Logic in Computer Science. In J.-J. C. M. R.J. Wieringa (Ed.), Deontic Logic in Computer Science John Wiley & Sons.

[Stamper, 1985 (8)] Stamper. (1985). Management Espistemology. In R. H. S. L.B. Methlie (Ed.), Knowledge Representation for Decision Support Systems Elsevier Science Publishers B.V. (North-Holland).

[Öhlund, 1992 (9)] S. Öhlund, C. Nellborn. (1992). Participative Modelling technique for Knowledge Acquisition. 3rd Workshop on next generation of Case Tools. D. A. S. Dr. B. Theodoulidis, Manchester.

TIDIGARE UTGIVNA PUBLIKATIONER AV TRIADGRUPPEN

Verksamhetskrav på informationsadministration

V 1: IA och verksamhetens krav – erfarenheter från offentlig förvaltning
V 2: Fallstudie av IA-projektet vidTeleverket
V 3: IA-erfarenheter från företag och myndigheter
V 4: Den gemensamma informationsmarknaden – en referensram för handlingsfrihet och konkurrenskraft
V 5: ...fråga är guld. Lokal affärsstyrning utifrån den egna verksamhetens data

Modellering

N 1: Modelleringsansatser för begrepps- och datamodellering – Beskrivning och försök till jämförelse
N 2: Generering av konceptuella modeller från policydokument
N 3: Espritprojektet Tempora
N 4: Prövning av regelbaserad metodik inom Posten
N 5: En kokbok i remodellering – utkast
N 6: Datorstöd för modellintegration
N 7: Modellbaserad kunskapsinsamling
N 8: Modellkvalitet
N 9: Samband mellan dokument och modeller
N 10: Modelleringshandboken
1 – Översikt
2 – Modelleringsledarens bashandledning
3 – Modellering i grupp
4 – Kommunikation
5 – Arbetsgångar
6 – Modelleringsväskan
7 – Objektorienterad verksamhetsanalys
8 – Basmodeller
9 – Regelmodellering i praktiken
10 – Business Process Reengineering
11 – Namnsättning
12 – Tolkning av grafiska modeller
N 11: Ett+Ett=Ett – Två praktikers erfarenheter av modellintegrering

Kunskapsförmedling

H 1: Handledarutbildning för modelleringsledare, avancerad
H 2: Slutrapport HUMLA prototyp
H 3: Utbildning i Informationsadministration
H 4: Spridning av Hybris – en fallstudie vid Telia

Uttagssystem

U 1: Hybris i Unix-miljö
U 2: DEBRIS
U 3: Hybris DOS/PimWin på Posten
U 4: Program för sökning i databaser – en marknadsöversikt
U 5: Att nå och förstå data – möjligheter och begränsningar

Katalogprinciper

K 1: IRDS
K 2: IRDS Modeller och modellnivåer
K 3: Koppling begreppsmodell – relationsmodell
K 4: IBM:s Repository Manager – en Introduktion
K 5: IBM:s Repository Manager: Datamodelleringsbegreppen
K 6: IBM:s Repository Manager: Begreppsmodellering i Information Model
K 7: IBM Repository Manager: Attribut- och värdemodellering i Enterprise Submodel
K 8: Navigering i Repository
K 9: TRIAD Newsletter – IRDS inom ISO. Dagsläget
K 10: TRIAD Newsletter – ISO/IRDS. Händelseutveckling 91/92
K 11: Samverkan mellan resurskataloger – visioner eller behov
K 12: AD/Cycle I Information Model – Processer och och informationsflöden mellan processer
K 13: AD/Cycle I Information Model – Info Flows inom Processmodellen
K 14: AD/Cycle I Information Model – Relationsdatabasmodellering
K 15: AD/Cycle I Information Model – Härledningsspecifikationer i begreppsmodellen
K 16: IA-prototyp
K 17: Repository AD/Cycle – International Users Group
K 18: RAD-konferensen i Chicago, 1992
K 19: Vad händer inom ANSI-IRDS?
K 20: Information Warehouse – vad är det?
K 21: CDIF – en översikt
K 22: PCTE – en översikt
K 23: XLII – en öppen och flexibel utvecklingsmiljö
K 24: Hybris IA/DA – En IA-prototyp vid Telia
K 25: Introduktion till GDMO-standarderna
K 26: OpenODB – en introduktion
K 27: ANSI/X3H7 "Object Information Management"
K 28: Object Management Group

KORT OM TRIAD

Triad är namnet på ett treårigt samarbetsprojekt kring informations-administration och dataadministration, IA/DA, som Telia, Posten, Ericsson, Statskontoret och SISU bedriver. Syftet är att utveckla parternas synsätt, metoder och hjälpmedel inom detta område. Arbetet inom Triad är uppdelat i delprojekt som är sammanförda i tre block.

Beställarblocket vänder sig dels till dem som är verksamhetsansvariga och måste ta ställning till IA-/DA-satsningar, dels till dem som har ansvaret för IA/DA inom en organisation. Delprojekten inom detta block arbetar med att formulera verksamhetens krav på IA/DA samt studerar och beskriver roller, organisation och arbetsformer för IA-/DA-arbete.

Utförarblocket vänder sig till dem som arbetar med IA/DA. Delprojekten arbetar med modellering, data- och resurskataloger samt uttagssystem.

Kunskapsförmedling är det block som ser till att resultaten kommer Triad-parterna till godo. Detta sker bland annat genom kurser, seminarier samt genom att rapporter som denna ges ut.