SISU rapport nr 7

Generering av naturligt språk från konceptuella scheman

Jonas Walles

SISU
Svenska Institutet för Systemutveckling
Box 1250, 164 28 KISTA

Generering av naturligt språk från konceptuella scheman

ISSN: 0282-9924

Innehåll

1. Sammanfattning

I rapporten beskrivs hur konceptuella scheman kan översättas till naturligt språk. Förutom en översättning av de fakta som anges explicit i schemat, dras också slutsatser om egenskaper hos objekten som anges implicit, bland annat egenskaper som ärvs från överordnade klasser.

Ett motiv för att göra översättningen till naturligt språk är att den kan användas för att kontrollera att schemat innehåller precis de fakta som avsetts.

För översättningen av ursprungsfakta till svenska används en så kallad DCG, Definite Clause Grammar.

Vidare diskuteras i vilka sammanhang översättningsprogrammet kan användas. Slutsatsen blir att en språkligt korrekt översättning kan erhållas endast om gränssnittet används i kombination med ett lexikon. Eftersom ett lexikon ändå utnyttjas av program som genererar konceptuella scheman **från** naturligt språk, visar det sig vara naturligt att kombinera dessa båda program med varandra.

I övriga fall (vid till exempel grafiskt gränssnitt, där lexikon inte behövs vid modelleringen) är det mer tveksamt om naturligt-språkgenerering för beskrivning av schemat ska användas, eftersom det inte finns någon större möjlighet att kontrollera vilka lexikala egenskaper de ingående orden har. Detta leder i sin tur till att det inte med säkerhet går att formulera språkligt korrekta utsagor om schemat.

2. English abstract

This report describes a program that translates conceptual schemas in the EMOL (Ericsson modelling language) notation into Swedish. A conceptual schema is a formal description of what objects and types of values (domains) there are in a data model and how they are related to each other.

The translation can be used as a validation procedure; it is an easy way of checking whether or not the formal description was the intended one.

Both explicit and implicit facts in the schema are generated. Implicit facts are for instance inherited properties of objects.

These facts and a so-called DCG (Definite Clause Grammar) serve as a basis for generation of Swedish descriptions.

One section of the report also investigates what linguistic problems the program encounters when trying to generate Swedish sentences from the conceptual schema. The conclusion is that it is absolutely necessary to have access to a lexicon in which information about plural forms of nouns etc is stored. Without the lexicon, the program will not be able to generate sentences that are linguistically correct.

Since a lexicon is used by interfaces that work in the opposite direction, i.e. generate conceptual schemas from natural language descriptions, the conclusion is that it is natural to combine these two kinds of interfaces.

However, when the conceptual schema is generated with a graphical interface, it is not so obvious that a natural language description will work, since the linguistic information that is needed cannot be obtained from anywhere.

3. Förord

Detta är en rapport som beskriver resultatet av ett examensarbete som utförts på Svenska Institutet för Systemutveckling (SISU) under hösten 1989 och våren 1990.

Handledare vid examensarbetet har varit Hercules Dalianis från SISU och Institutionen för Data- och Systemvetenskap (DSV) vid Stockholms universitet/KTH. Handledare/examinator på KTH har varit Yngve Sundblad vid Institutionen för Numerisk Analys och Datalogi (NADA).

4. Varför naturligt-språkgenerering ?

Vid konceptuell modellering erhålles scheman som kan vara svåra att överblicka för den som är ovan vid deras representationsform. Detta gäller speciellt mappinginformationen (se nedan) som är lättare att förstå i naturligt språk. För att få en större möjlighet att upptäcka eventuella fel och brister i schemat, kan då en översättning till naturligt språk göras.

Tack vare denna översättning och möjligheten att göra själva modelleringen i naturligt språk (se [Dahlgren 90]) erhålles ytterligare en fördel; vem som helst med kunskap om det aktuella tillämpningsområdet kan utföra modelleringen utan att behöva lära sig schemats representationsform.

Översättningsprogrammet kan också dra slutsatser om fakta som finns implicit i schemat och redovisa dessa slutsatser på ett enkelt sätt.

För en mer genomgående beskrivning av vad konceptuell modellering är, hänvisas till "Konceptuell modellering - Informationsanalys" [Bubenko 84].

5. Problemformulering

Problemet är följande: indata är ett konceptuellt schema skrivet i Ericsson Modelling Language, EMOL (finns beskrivet i EMOL metodhandbok [Ericsson Telecom 88]). Schemat består av en rad fakta av tre olika slag.

Objektklassdefinitioner

Objektklasser definierar existensen av en klass av objekt, och talar om vilka direkt överordnade klasser objektklassen har. I detta exempel definieras objektklassen "doktor", som har den direkt överordnade klassen "person":

entity(doktor,_,_,[person]).

En tänkbar översättning av detta faktum är "En doktor är en person". I själva verket kommer översättningen att bli: "person <- doktor". Anledningen till detta gås igenom nedan. I det allmänna fallet ser det ut så här:

entity(objekt,_,_,[lista_med_direkt_överordnade_klasser]).

Attribut

Attribut kan användas för ange två olika typer av fakta som har en sak gemensamt: de beskriver en relation mellan två begrepp. I det ena fallet är det fråga om en relation mellan två objektklasser, i det andra en relation mellan en objektklass och en domän (för definition av domän, se nästa punkt). Ett exempel på ett sådant faktum är

attribute(ligger_på,hyser,patient,avdelning,mapping(1,1,1,m),_,_).

Här beskrivs relationen "ligger_på" mellan två objekt, "patient" och "avdelning". Dessutom anges inversrelationen, som i exemplet ovan är "hyser". Detta faktum får på svenska följande utseende:

1. "Varje patient ligger på precis en avdelning".
2. "Varje avdelning hyser minst en patient".

I mapping-predikatet beskrivs vilken typ av avbildning det är fråga om. Det allmänna fallet skrivs *mapping(rel_min, rel_max, invrel_min, invrel_max).*

Detta betyder i föregående exempel att "patienter" har relationen "ligger_på" till minst rel_min (1) och som mest rel_max (1) "avdelningar". På samma sätt har "avdelning" relationen "hyser" till som minst invrel_min (1) och som mest invrel_max (m = godtyckligt **många**) "patienter".

Relationer mellan objektklass och domän anges på samma sätt som relationerna mellan två objektklasser. Den enda skillnaden är att inversrelationen (dvs relationen sedd ur domänens synvinkel) inte anges. Attribute-predikaten får således följande allmänna utseenden:

attribute(rel,inversrel,objektklass1,objektklass2,mapping(min1,max1,min2,max2),_,_).
attribute(rel,_,objektklass,domän,mapping(min1,max1,min2,max2),_,_).

Domänbeskrivningar

Domänbeskrivningar definierar en mängd värden av en viss typ, till exempel heltal eller strängar. Alla objekt i objektklasser som står i en relation till domänen kan anta ett eller fler av dessa värden. Ett faktum som vill säga att "Namn är av typen string" får i schemat följande utseende:

data_object(namn,string).

Det allmänna fallet skrivs så hår:

data_object(domän_namn,typ_av_värden_i_domänen).

Dessa tre typer av fakta ska nu översättas till svenska med ett program skrivet i SICStusprolog (se [Carlsson 88]. Dessutom ska programmet kunna redovisa information som finns implicit i schemat: ärvda egenskaper hos objekt eller egenskaper som bara vissa objekt i objektklassen har.

6. Lösningsansats

6.1 Alternativa lösningsförfaranden

Det finns andra lösningsmetoder än den som redovisas i denna rapport. En möjlighet är att inte generera den beskrivande texten direkt från det konceptuella schemat, utan istället gå via en mellanliggande form, till exempel första ordningens predikatlogik, FOL. Se [Dalianis 89].

Denna lösning gör det möjligt att använda predikatlogikens omskrivningsregler, som bland annat tillåter omdefiniering av allkvantifikatorn i termer av existenskvantifikatorn, och existenskvantifikatorn i termer av allkvantifikatorn. Därigenom kan flera utsagor om samma faktum formuleras. En utsaga av typen "Alla doktorer behandlar minst en patient" skulle då kunna omformuleras som "Det existerar ingen doktor som inte behandlar minst en patient". En fördel med att få flera formuleringar är att eventuella oklarheter i en formulering kan undanröjas med hjälp av andra formuleringar.

Det är dock tveksamt om det är värt besväret att gå via FOL. Dessa alternativa formuleringar kan enkelt åstadkommas utan predikatlogik. I denna rapport (se sid 8) visas för övrigt hur detta kan lösas.

Ett problem med att översätta till FOL är att viss information som finns i schemat inte kan representeras på ett bra sätt i FOL. Ett exempel på detta är mappinginformationen i schemat, som i FOL representeras med negation, ekvivalens samt all- och existenskvantifikatorn. Detta representationssätt är bra så länge som mappingvärdena är 0, 1 och m (många). Önskas en nyansering blir det betydligt krångligare.

Betrakta till exempel utsagan "Varje bil har exakt 4 hjul". I schemat har denna relation mappingvärdet (4,4). I FOL kräver detta faktum följande (informella) formulering: "För alla bilar y existerar det hjul x1, x2, x3, x4 sådana att inget xi = xj för i<>j och y har x1, x2, x3, x4, och om det existerar ett hjul x5 sådant att y har x5, så är x5 lika med antingen x1, x2, x3 eller x4". Problemet är alltså att mappingfall med stigande värden på argumenten ger upphov till formler i predikatlogik med fler variabler och fler villkor på dessa variabler. Dessa formler är komplicerade redan för relativt små mappingvärden och komplexiteten växer sedan kraftigt då mappingvärdena stiger.

6.2 Vald metod

För att undvika problemen med FOL används i föreliggande lösning istället en direkt omskrivning från schemat till naturligt språk. Hur skall då denna omskrivning gå till?

Betrakta till att börja med fakta av "attribute"- typ. Det visar sig att meningarna som genereras från denna typ har ett gemensamt mönster. De består alla av subjekt, predikat (verb), ackusativobjekt samt artiklar (kvantifikatorer) framför subjektet och ackusativobjektet. Exempel: *attribute(behandlar,_,doktor, patient,mapping(1,m,_,_),_,_).* ger upphov till meningen "Varje(kvantifikator) doktor(subjekt) behandlar(predikat) minst en(kvantifikator) patient(ackusativobjekt)".

Dessa fem element har en direkt motsvarighet i "attribute"-predikatets olika termer: subjektet och ackusativobjektet motsvaras av de objekt mellan vilka relationen råder (ibland också domän på ackusativobjektets plats), predikatet är relationens namn och beroende på vilken mapping-informationen är, så blir kvantifikatorerna olika. Relationsbeskrivningarna ser alltså alltid likadana ut, med undantag av kvantifikatorerna.

Detta kan nu utnyttjas av en så kallad DCG, Definite Clause Grammar (se [Pereira 80]), en beskrivning av en grammatik som i regel används för grammatisk analys ("parsning" på svensk-engelska), men som även kan användas omvänt, dvs för att generera text tillsammans med ett lexikon. I det här fallet används DCG:n för att generera text med hjälp av namnet på relationen, relationsobjekten, eventuella domäner och mappinginformationen om relationen. DCG:n består av en mängd regler med följande utseende:

```
s(short_attribute(Relation, A, B, mapping(1,m))) -->
                    [alla], [A1], [Relation], [minst], [Obest_art], [B],
                    {boej(A,A1),  art(Obest_art,B)}.
```

Kort förklaring av notationen: vänsterledet omskrivs som de ord som står inom hakparenteser i högerledet, om mappinginformationen inom "short_attribute" är (1,m). Inom mängdklamrarna står ett antal operationer i Prolog som utförs på indata för att de skall få rätt format vid utmatningen. Exempel på detta är val mellan "en"/"ett" respektive "ingen"/"inget" vid kvantifiering av relationsobjekten samt att ett eller flera relationsobjekt måste böjas till pluralform.

I DCG:n har de vanligaste mappingfallen en egen regel, medan de övriga mappingfallen omfattas av tre allmänna regler: den första används då objektet står i relation med "exakt n st" objekt av det andra slaget, där n är större än ett, den andra regeln används då objektet står i relation med "mellan n och m st" objekt, där n är större än ett och m är större än n, och den tredje regeln om objektet står i relation med "minst n st", där n är större än ett. De mappingfall som betraktas som vanliga är (0,1), (0,m), (1,1) och (1,m).

Med hjälp av DCG:n kan nu också flera olika formuleringar för varje faktum anges, genom att flera regler för samma mappingfall införs. Regeln ovan kan nu kompletteras med följande formel, som motsvarar omskrivning av allkvantifikatorn i termer av existenskvantifikatorn:

s(short_attribute(Relation, A, B, mapping(1,m))) -->
[det], [finns], [Neg], [A], [som], [inte], [Relation],
[minst], [Obest_art], [B],
{neg(A,Neg), art(Obest_art,B)}.

Programmet ger bara en formulering, men detta exempel visar i alla fall att det är möjligt att formulera flera utsagor om samma faktum.

Den andra typen av fakta är beskrivningar av domäner. Här sker också en direkt omskrivning av den ursprungliga prolognotationen.

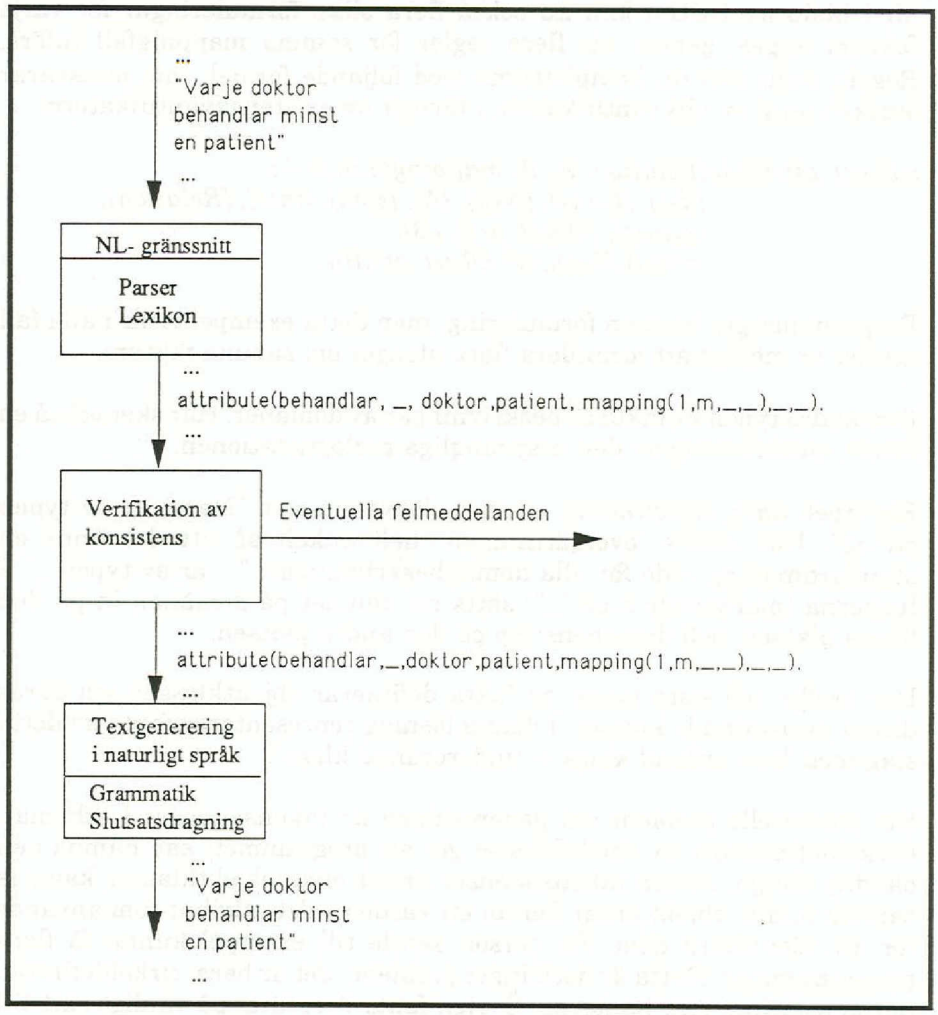
Exempel: *data_object(name, string).* skrivs om som "Name är av typen string". Här utförs "översättningen" helt enkelt så att det finns en standardmening redo för alla domänbeskrivningar: " _ är av typen _". I luckorna (markerade med "_") sätts nu namnet på domänen in på den första platsen, och domänens typ på den andra platsen.

Den tredje och sista typen av fakta definierar objektklasser och deras direkt överordnade klasser. I denna lösning representeras fakta av detta slag med "överordnad klass <- underordnad klass".

Ett potentiellt problem vid genereringen är inkonsistenser i schemat. Cirkeldefinitioner av objektklasser gör att programmet kan hamna i en oändlig slinga. Andra inkonsistenser är att olika objektklasser kan ges samma namn, objekt antar fler än ett värde av det attribut som används för att identifiera dem. En person skulle till exempel kunna få flera personnummer. Detta är dock inget problem; det är bara cirkeldefinitionerna som inte kan hanteras. Övriga fakta översätts på vanligt sätt till svenska, men nu blir det lättare att upptäcka de inkonsistenser som finns i schemat.

Det finns ett program för diagnos av konceptuella scheman som konstruerats av Rolf Wohed vid SISU [Wohed 88]. Programmet upptäcker bland annat de ovan nämnda inkonsistenserna. Genom en inledande diagnos med Woheds program kan därmed cirkeldefinitioner i schemat lokaliseras och tas bort.

Nedanstående figur (figur 1) visar en möjlig omgivning för programmet. Ett gränssnitt genererar först ett konceptuellt schema i EMOL från naturligt språk (se [Dahlgren 90]). Konsistensen för detta schema kan därefter kontrolleras med hjälp av Woheds program.



Figur 1. En tänkbar omgivning för programmet: Ett gränssnitt som tillåter modellering i naturligt språk ger upphov till ett konceptuellt schema, vars konsistens kan undersökas. Därefter översätts schemat tillbaka till naturligt språk samtidigt som vissa slutledningar dras om egenskaper hos objekten i schemat.

7. Vilken information kan utvinnas med programmet?

7.1 Egenskaper hos objekt

Varje objekt i en objektklass har ett antal egenskaper, nämligen att vara medlem av överordnade klasser, ingå i en relation med andra objektklasser, eller att anta vissa värden ur ett värdeförråd (domän). Dessa egenskaper anges explicit i schemat, men det finns ytterligare en mängd **implicita** egenskaper, nämligen de som ärvs från överordnade klasser eller som innehas av underordnade klasser till den aktuella objektsklassen. De implicita egenskaperna kan erhållas ur schemat med hjälp av en mängd regler som beskrivs nedan.

Här följer en beskrivning av hur objektet i en klass kan erhålla sina egenskaper samt hur naturligt-språkbeskrivningar genereras av dessa egenskaper.

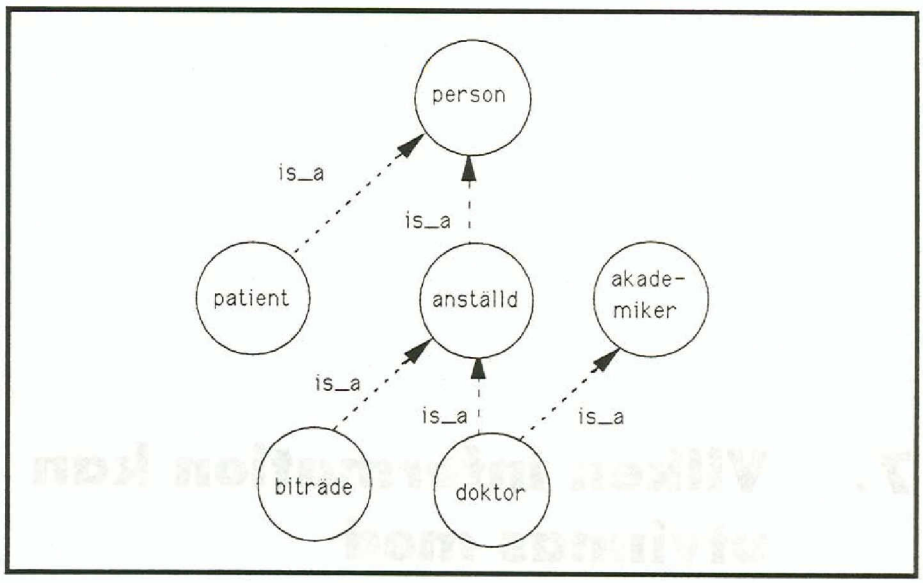
Klassrelationer

Varje klass står i regel i en viss generisk relation till ett antal andra klasser. Relationerna kan vara över- eller underordnade (se figur 2). Exempel är "person" och "anställd". "Person" är en överordnad klass till "anställd", och "anställd" en underordnad klass till "person". Relationen mellan en klass och dess överordnade klass brukar kallas "is_a-relation". I schemat ser det ut så här:

entity(anställd,_,_,[person])

Det allmänna fallet skrivs:

entity(Klass,_,_,[Lista_med_direkt_överordnade_klasser]).



Figur 2. Exempel på klasshierarki.

Skillnaden mellan en överordnad klass och en direkt överordnad klass är att det i klasshierarkin inte kan ligga någon klass mellan en klass och dess direkt överordnade klass, medan detta är möjligt för en klass och dess överordnade klass.

Istället för en enkel uppräkning av under- eller överordnade klasser så är det önskvärt att också få en beskrivning av deras struktur. På så vis skulle det i beskrivningen av klassrelationerna i figur 2 ovan framgå att "person" och "anställda" är en grupp för sig, och "akademiker" en grupp för sig, även om alla tre grupperna är överordnade klasser till "doktorer". Ett sätt att få med strukturen i representationen är att använda sig av "klasskedjor".

En klasskedja som beskriver en överordnad relation utgår ifrån objektklassen och går sedan så långt upp i hierarkin den kan komma. För klassen "doktor" betyder detta att två kedjor kommer att genereras: *"person <- anställd<- doktor* " och *"akademiker <- doktor"*. Pilarna mellan objektklasserna kan ses som implikationspilar: "Om någon är en doktor, så är han en akademiker". På samma sätt beskrivs de underordnade klasserna för en klass. Beskrivningen utgår då ifrån objektklassen och går så långt ner i hierarkin den kan komma. För klassen "person" i figur 2 gäller då att tre kedjor kommer att genereras: *"person <- patient"*, *"person <- anställd <- biträde"* och *"person <- anställd <- doktor"*.

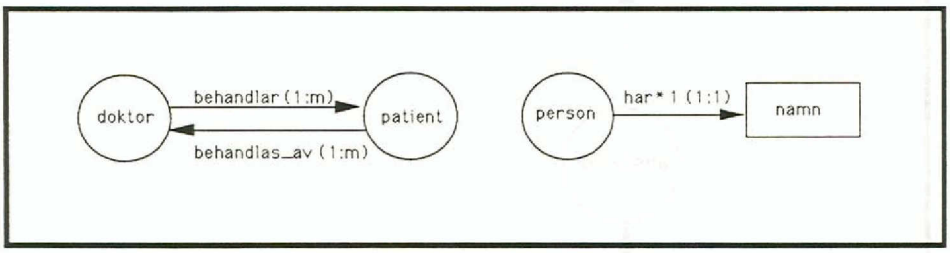
Anledningen till att formuleringen med klasskedjor har valts är just för de långa kedjornas skull. Om dessa blir alltför långa kan det bli tungt att läsa den alternativa formuleringen. Jämför "En kirurg är en doktor som är en anställd som är en person" med *"person <- anställd <- doktor <-kirurg"*.

Objektspecifika egenskaper

Objektspecifika egenskaper står explicit uttryckta i schemat. Med andra ord, varje sådant faktum motsvaras av ett prologpredikat av följande slag:

attribute(behandlar,behandlas_av,doktor,patient,mapping(1,m,1,m),_,_).

Detta är en beskrivning av en relation mellan två objekt, "doktor" och "patient". Relationens namn är "behandlar", och inversrelationens namn är "behandlas_av" (se även figur 3). På svenska ger detta faktum upphov till meningarna "Varje doktor behandlar minst en patient" samt "Varje patient behandlas av minst en doktor".



Figur 3. Exempel på relation mellan två objekt respektive objekt och domän.

I fallet relationsbeskrivningar ingår även de relationer som används för att identifiera objekt, så kallade identifierande attribut. Dessa känns i schemat igen på att relationens namn avslutas med en stjärna. Ibland måste flera relationer tillsammans användas för att entydigt identifiera ett objekt. Då avslutas alla relationsnamnen med en stjärna.

Det är möjligt att använda flera alternativa identifierare som var och en för sig räcker för att identifiera objektet. En person som är anställd vid ett företag kan till exempel identifieras dels med ett anställningsnummer, dels med sitt personnummer. Detta markeras genom att stjärnorna avslutas med olika heltal. Attributen kallas då primär- och sekundärnycklar etc.

Exempel:
*attribute('har*1',_,person,namn,mapping(1,1,_,_),_,_).*
entity(person,_,_,[]).
data_object(namn,string).

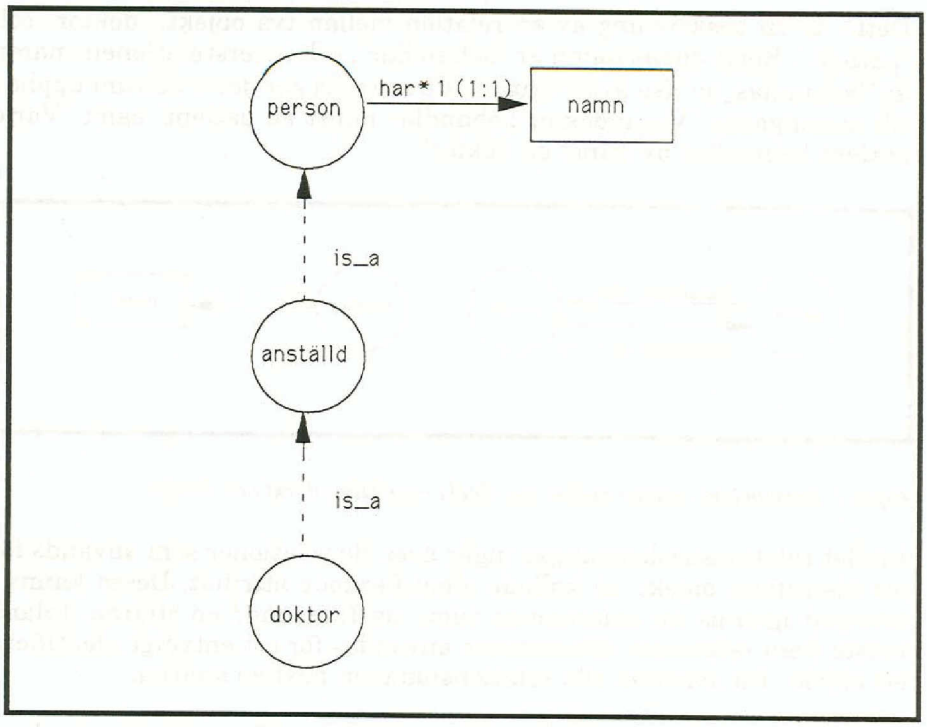
Detta faktum behandlas precis som alla andra relationsbeskrivningar förutom att programmet talar om att det aktuella attributet är ett identifierande attribut, samt eventuellt också vilken typ av nyckel det är fråga om. I det här exemplet är det alltså fråga om en primärnyckel, eftersom relationens namn avslutas med en etta.

Varje mappingfall får nu varsin regel i en DCG, precis som visats i avsnittet "vald metod" ovan.

Ärvda egenskaper

Objekt kan ärva egenskaper från överordnade klasser. Exempel (se figur 4): "Alla doktorer har exakt ett namn". Denna egenskap ärvs av "doktor" via "anställd" från den överordnade klassen "person". I prologrepresentationen av schemat ser det ut så här:

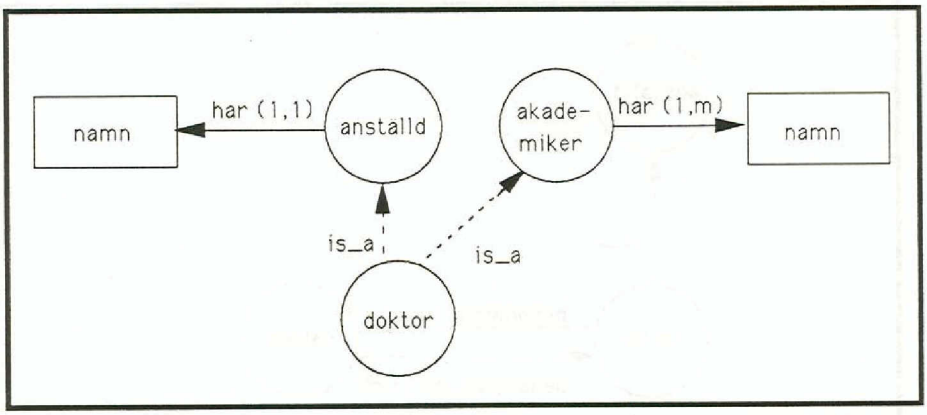
*attribute('har*1',_,person,namn,mapping(1,1,_,_),_,_).*
entity(doktor,_,_,[anställd]).
entity(anställd,_,_,[person]).
entity(person,_,_,[]).
data_object(namn,string).



Figur 4. Klassen "doktor" ärver egenskapen "namn" från sin överordnade klass "person".

Objekt kan som synes ärva egenskaper "i flera klassled". Informationen som DCG:n får som indata är densamma som i föregående fall, med den skillnaden att objektet "person" nu är utbytt mot objektet "doktor". Förutom själva egenskapsbeskrivningen redovisas också varifrån egenskapen har ärvts, dvs "från person". Detta är dock inte någon del av DCG-regeln eftersom ärvda och objektspecifika egenskaper delar på samma regler.

Vid så kallade multipla arv kan namnkollisioner uppstå (se figur 5). I figuren ärver klassen "doktor" dels egenskapen att "ha exakt ett namn" (från den överordnade klassen "anställd"), dels egenskapen att "ha minst ett namn" (från den överordnade klassen "akademiker"). Beskrivningen kan då göras klarare genom att egenskapen förses med prefix för respektive överordnade klass; med andra ord skulle begreppen "personalnamn" och "akademiker-namn" införas, och på detta vis ge ett förtydligande. Detta gör det förstås inte lättare att göra en eventuell implementering. Eftersom detta examensarbete handlar om naturligt-språkgenerering kommer dock detta problem inte att behandlas vidare.



Figur 5. Namnkollision vid multipelt arv.

Egenskaper som utmärker vissa av objektklassens underordnade klasser

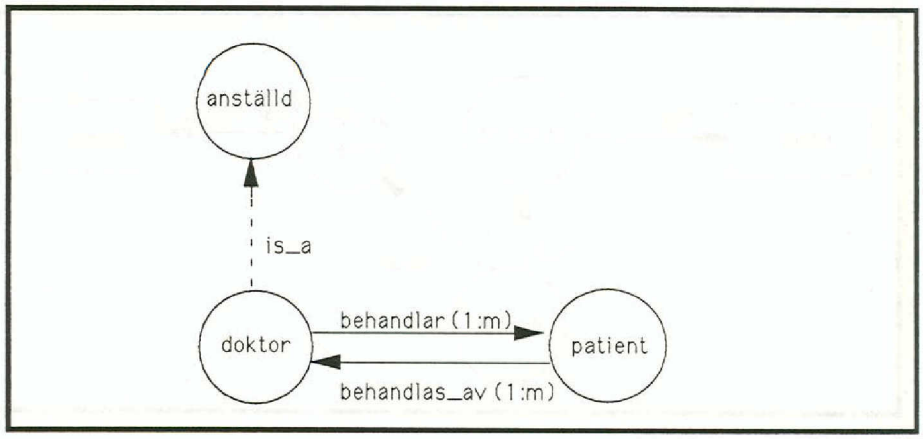
Klassen som objektet tillhör kan bestå av flera underordnade klasser där vissa av dessa har egenskaper som är utmärkande för just de klasserna (se figur 6). Detta kan uttryckas med hjälp av en relativsats. Exempel: "De anställda som är doktorer behandlar minst en patient". Denna egenskap är egentligen utmärkande för doktorer. I ursprunglig prologform:

attribute(behandlar,behandlas_av,doktor,patient,mapping(1,m,1,m),_,_).
entity(doktor,_,_,[anställd]).
entity(anställd,_,_,[person]).

Informationen som används av DCG:n i detta fall måste nu kompletteras så att i fallet ovan både "anställd" och "doktor" kommer med (förutom "patient" och mappinginformationen). I formuleringen av detta faktum har dock den överordnade klassens namn strukits, eftersom det alltid framgår av sammanhanget vilken klass som åsyftas. DCG-regeln kan se ut så här:

```
s(short_attribute(Rel,_,B,C,mapping(1,m))) -->
    [de], [som], ['är'], [B1],[Rel], [minst], [Obest_art],[C],
    {boej(B,B1),, art(Obest_art,C)}.
```

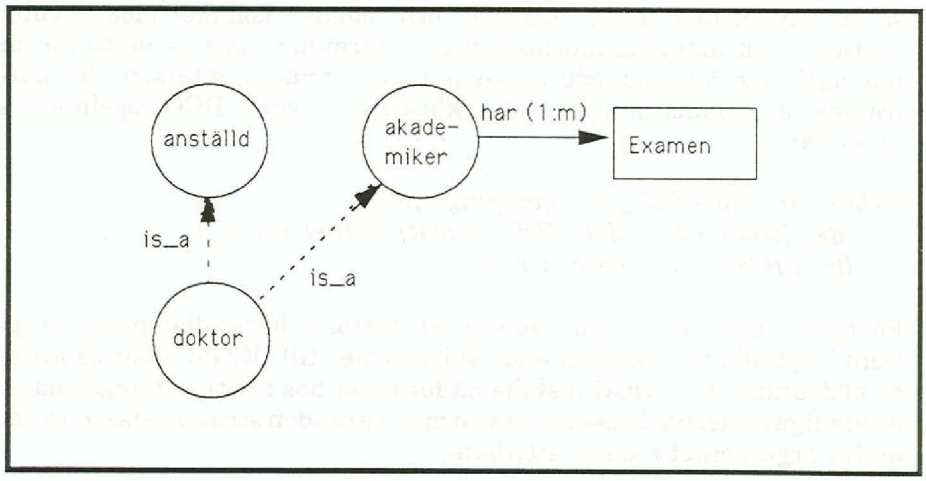
Formuleringen blir alltså "De som är doktorer behandlar minst en patient". Att alla tre objekten ändå skickas med till DCG:n beror på att det är nödvändigt att i DCG:n skilja på formatet hos relativsatsreglerna och de vanliga reglerna. I exemplet ovan motsvaras den strukna klassen av det andra argumentet i short_attribute, "_".



Figur 6. Den underordnade klassen "doktor" har en speciell egenskap.

Ett specialfall av detta fall inträffar då den underordnade klassen **ärver** ett attribut från en annan överordnad klass än den som beskrivs (se figur 7). Exempel: "De anställda (objekt som ska beskrivas) som är doktorer (underordnad klass) har minst en examen". Doktorer har ärvt detta attribut från akademiker, som också är en överordnad klass till doktorer. Akademiker är här inte en underordnad klass till anställda, eftersom det kan finnas akademiker som inte är anställda på sjukhuset (det här schemat grundar sig på en sjukhusmodell), till exempel klassen "agronomer". I ursprunglig prologform:

attribute(har,_,akademiker,examen,mapping(1,m,_,_),_,_).
entity(anställd,_,_,[person]).
entity(doktor,_,_,[anställd,akademiker]).
entity(akademiker,_,_,[]).



Figur 7. Den underordnade klassen till "anställd" ("doktor") ärver från "akademiker" .

7.2 Övrig information: listning av befintliga objekt och domäner

Övrig information som är av intresse är vilka domäner som finns i schemat, och vilken typ av värden som kan antas av de objekt som är knutna till dem. Programmet letar nu efter klausuler på formen

data_object('Domännamn', typ_av_värden_i_domänen).

Alla sådana klausuler som påträffas ger upphov till utsagor av typen "Domännamn är av typen typ_av_värden_i_domänen".

Det finns också en möjlighet att få en listning av alla objekt som finns i schemat. Dessa skrivs helt enkelt ut i bokstavsordning.

Till slut kan användaren av programmet också få en översättning av schemat med bara de egenskaper som finns angivna explicit i schemat, med andra ord utan angivande av uttömmande information om klassrelationer, ärvda egenskaper eller egenskaper som underordnade objektklasser har.

8. Lösningsmetod

Huvuddelen av programmet går ut på att beskriva egenskaperna hos objekt. Två frågor som är intressanta är hur dessa egenskaper hittas i schemat och om alla egenskaper hittas.

Alla de typer av egenskaper hos objekt som beskrivits i rapporten har genererats med en så kallad "failure driven loop". En sådan konstruktion fungerar så att programmet misslyckas efter det att det skrivit ut en lösning. Detta medför att nya lösningar kommer att genereras tills det inte finns några fler. Denna konstruktion används för att gå igenom den klasshierarki som objektet tillhör, dels för att hitta överordnade klasser, dels för att hitta underordnade. Programmet har tillgång till hierarkin genom de objektklassdefinitioner som finns i schemat. För varje under- eller överordnad klass som påträffas undersöks om den har någon egen egenskap. I så fall kommer lämplig beskrivning genereras.

På samma sätt hittas alla definitioner av domäner. Genom användandet av en "failure driven loop" kommer alla domäner till slut att hittas.

9. Språklig analys

I avsnittet "vald metod" (sid 8) visas att det bara är det bara en typ av faktum som ställer till med språkliga problem, nämligen "attribute"-beskrivningar. Det visar sig att dessa problem inte kan lösas utan tillgång till ett lexikon. Till att börja med tas här upp vilka "språkliga operationer" som måste utföras på indata vid generering av beskrivningar i naturligt språk, och därefter vilket informationsbehov operationerna ger upphov till.

I avsnittet "Vald metod" visades att den genererade meningen i normalfallet (ej relativsats) består av fem delar; ett subjekt och ett ackusativobjekt (båda substantiv), två kvantifikatorer (motsvarande mappinginformationen) samt ett verb (motsvarande relationsnamnet).

Eftersom beskrivningarna är i presens kommer verb inte ställa till med några problem. Detta beror på att verben i schemat är i presensform och verb i presens ser alltid likadana ut oavsett person och numerus. Ett problem är dock att relationer inte nödvändigtvis måste anges med verb (se sid 20).

Kvantifikatorerna, till exempel "flera" och "alla", kan i regel skrivas ut direkt i DCG:n och har alltså en entydig motsvarighet i respektive mappingvärde. Det finns dock undantag (se nedan), nämligen då det är möjligt att välja mellan flera kvantifikatorer, eller snarare flera instanser av samma kvantifikatortyp.

För substantiven gäller att programmet måste kunna böja dem i plural i vissa fall.

9.1 Nödvändig lexikal information

Vilket genus har substantivet?

Genus kan vara neutrum ("ett-ord") eller utrum ("en-ord"). Anledningen till att genusinformationen måste vara känd är att programmet vid kvantifieringar måste välja mellan "en"/"ett" eller "ingen"/"inget". Detta är ett problem som inte kan lösas utan tillgång till lexikon.

Hur böjs substantiv i plural?

I vissa mappingfall kommer den genererade naturligt-språkbeskrivningen att innehålla substantiv i pluralform. Eftersom schemat ursprungligen innehåller substantiv i singularform, måste programmet hitta ett sätt att böja orden.

Om programmet skall klara sig utan lexikon finns det några böjningsregler som ger en viss hjälp, t ex att alla ord som slutar på "-ing" böjs "-ingar" i plural. De flesta regler har dock undantag, t ex i fallet ovan, "ett ting" resp "flera ting". Dessa regler är alltså inte speciellt pålitliga. Därför kommer de programmet bara användas som en sista utväg, när orden inte finns i lexikonet. Om orden saknas i lexikonet eller om ingen böjningsregel kan tillämpas, så kommer ordet lämnas oböjt i singularform.

Ytterligare ett argument för att lexikon måste användas är de substantiv som böjs oregelbundet i plural, till exempel "bok"/ "böcker".

Vissa substantiv ska inte böjas alls i plural. I engelskan kallas denna grupp av ord "uncountables" (i motsats till "countables"). I svenskan finns det dock ingen heltäckande term för detta begrepp. Dessa exempel på företeelsen är hämtade ur "Nusvensk grammatik" [Jörgensen 86]:

- Gruppbetecknande substantiv: "personal", "ohyra" och "bagage".
- Ämnesbeteckningar: Exempel: "olja", "guld", "cancer" etc.
- Abstrakta massord: "fruktan", "bilism", "ålder".

De gruppbetecknande substantiven för också det problemet med sig att ibland vill tala om dem som en helhet (kollektivt) och ibland om en eller flera av dess medlemmar (distributivt). Ett exempel på en relation av det första slaget är "Varje avdelning har precis en personal", och den andra "Varje personal behandlar minst en patient". I det andra fallet är den önskade formuleringen kanske "Varje person ur personalen behandlar minst en patient".

Vilken ordklass tillhör ordet?

Anledningen till att denna fråga måste besvaras är att programmets ursprungliga översättningsregler i DCG:n förutsätter att relationer mellan objekt uttrycks med transitiva verb (dvs verb som tar objekt). Objekten mellan vilka relationen råder är då subjekt respektive ackusativobjekt i satsen.

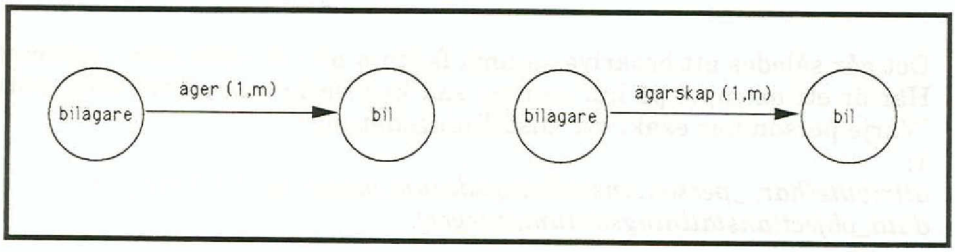
Exempel: "Alla programmerare(subjekt,relationsobjekt) har(verb,relation) minst en terminal(ackusativobjekt, relationsobjekt)".

Problemet består i att det inte finns något direkt krav i EMOL att det **måste** vara så. Det finns exempel på hur relationens namn uttrycks med substantiv, se figur 8. Relationens namn är dock den enda platsen där verb **kan** förekomma. Finns det då inget verb där, så måste en annorlunda typ av mening genereras. I den meningen måste det finnas ett nytt verb som ersätter det som borde ha stått där. Det finns två möjligheter att lösa problemet: med hjälp av substantivet som beskriver relationen skulle det gå att hitta motsvarande verb. Ett exempel på detta är att med substantivet "ägarskap" och ett antal regler sluta sig till att motsvarande verb måste vara "äger".

En betydligt enklare lösning är att i DCG:n införa följande typ av regel:

```
s(short_attribute(Rel,A,B,mapping(1,m)))-->
    [Obest_art1], [godtyckligt], [Vald_t], [A], [har], ['förhållandet'],
    [Rel], ['till'], [minst], [Obest_art2], [B],
    {noun(Rel), art(A,Obest_art1), art(B,Obest_art2),
     vald_t(A, Vald_t)}
```

Den här beskrivningen blir mer stelbent, men den fungerar. Regeln används bara om variabeln Rel (som står för relationen) är ett substantiv. I prologkoden görs en kontroll av detta med hjälp av predikatet noun. Detta kräver förstås tillgång till ett lexikon. Endast om relationsnamnet inte är ett substantiv används de ursprungliga reglerna, som kräver att relationsnamnet är ett verb.



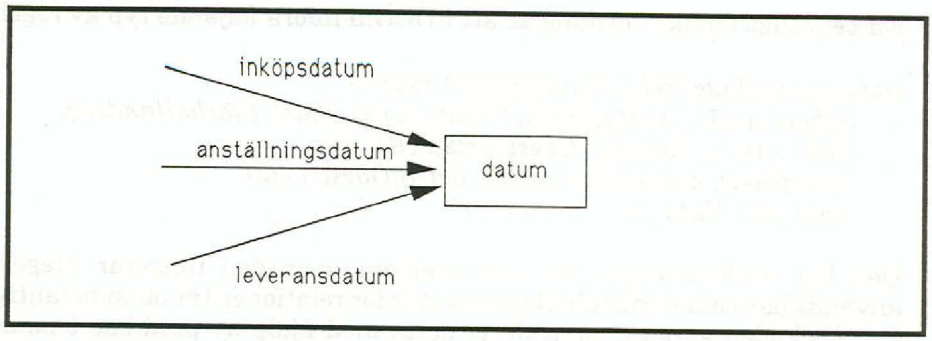
Figur 8. Relationer kan uttryckas både med substantiv och verb.

Harriet Dahlgrens naturligt-språkgränssnitt som omnämnts ovan i avsnittet "vald metod" fungerar så att det genererar relationsbeskrivningar där relationsnamnet är ett verb. Detta är ju självklart, eftersom verb måste användas i naturligt språk om meningen ska bli grammatiskt fullständig. Verbet blir då namn på relationen mellan de objekt som förekommer i meningen. Beskrivningar av scheman som genererats med naturligt-språkgränssnitt kommer därför bara att använda de ursprungliga verb-översättningsreglerna.

Om istället ett grafiskt gränssnitt används vid modelleringen så finns bägge möjligheterna kvar; dels relationer angivna med verb, dels angivna med substantiv. Problemet blir då att veta vilken variant som används. Det går inte att skilja på orden bara genom att titta på dem. Ett lexikon måste finnas tillgängligt för att ge information om ordens ordklasstillhörighet.

Slutsatsen blir således att programmet går bra att använda tillsammans med ett naturligt-språkgränssnitt, därför att det då går att använda lexikonet som måste finnas där. Däremot kan det bli problem tillsammans med ett grafiskt gränssnitt eftersom de inte behöver ha något lexikon. Problemet med det grafiska gränssnittet skulle förstås lösas om det även där infördes ett lexikon.

Problemet med ordklassval för relationsnamnet gäller förstås även för relationer mellan objektklass och domän. Även i detta fall kan relationens namn anges både med substantiv och verb. En anledning till att använda substantiv som beskrivning av relationen är att substantivet ger mer information om relationen än ett verb. I figur 9 visas hur det går att ge informationen att typen av datum det handlar om i själva verket är "inköpsdatum", "anställningsdatum" och "leveransdatum".



Figur 9. Relationsnamnen ger ytterligare information om relationerna (modifierad figur ur EMOL Metodhandbok [Ericsson Telecom 88] sid 63).

Det går således att beskriva samma faktum på två olika sätt i schemat. Här är ett exempel på hur samma sak kan beskrivas på två olika sätt; "Varje person har exakt ett anställningsdatum":
1:
attribute(har,_,person,anställningsdatum,mapping(1,1,1,m),_,_).
data_object(anställningsdatum,integer).

2:
attribute(anställningsdatum,_,person,datum,mapping(1,1,1,m),_,_).
data_object(datum,integer).

Det går dock inte att säga vilken variant som används i varje enskilt fall utan att använda ett lexikon som kan avgöra om relationsnamnet (i det här fallet "har" respektive "anställningsdatum") är ett verb eller ett substantiv. Programmet klarar båda varianterna, men det måste veta vilken variant som används i respektive fall, och det gör det inte utan ett lexikon. Utan lexikon hade det andra fallet gett upphov till utsagan "Varje person anställningsdatum precis ett datum".

Det går också i detta fall att använda den nya typen av DCG-regler då relationsnamnet är ett substantiv. Här har dock valts att låta det nya relationsnamnet bli verbet "har" och domänens namn ersättas av relationen, som ju är ett substantiv. Detta kan ses som att predikatet *attribute (anställningsdatum,_,person,datum,mapping(1,1,1,m),_,_).* tillfälligt omtolkats som *attribute(har,_,person,anställningsdatum, mapping(1,1,1,m),_,_).*

10. Slutsatser

Här har redovisats ett antal fall som visar att generering av naturligt språk med utgångspunkt i ett konceptuellt schema kräver tillgång till ett lexikon som kan ge information om de i schemat ingående ordens lexikala egenskaper. Meningar som genereras är dock mycket enkla. Mängden information som krävs är därför inte speciellt stor.

Det går till exempel att använda sig av den lexikala information som finns i Harriet Dahlgrens naturligt-språkgränssnitt. Där finns registrerat

1. vilken ordklass ett ord tillhör, verb eller substantiv

2. vilket genus substantiv har, utrum ("en-ord") eller neutrum ("ett-ord")

3. vilken deklination (böjningsklass) substantiv tillhör, vilket behövs för pluralböjning

4. i förekommande fall, hur oregelbundna substantiv böjs

Dock gäller för genusinformationen att den bara finns för regelbundna substantiv. Informationen för oregelbundna substantiv behöver alltså en komplettering med information om genustillhörighet.

10.1 Vem ska skapa lexikonet ?

Något som tyvärr är nödvändigt är att skapa ett lexikon för alla ord som ska ingå i schemat. Lexikonet behövs för parsningen vid skapandet av det konceptuella schemat, och sedan även för generering av språkligt korrekta meningar.

Eftersom det i schemat ofta kan vara fråga om förhållandevis ovanliga ord, är det bara den som senare ska skapa schemat som i förväg kan känna till alla de ord som måste matas in i lexikonet. Detta resulterar i ett dubbelarbete. Först måste orden matas in i lexikonet, och därefter också i schemat.

Därmed kan det tyckas att en del av tjusningen med naturligt-språk-gränssnittet går förlorad, eftersom tanken är att en användare skall kunna sätta sig vid en terminal och börja skapandet av schemat "från noll". Möjligen kan detta lösas genom att det finns tillgång till ett mycket stort lexikon som innehåller alla ord som kan tänkas uppkomma, och som bara behöver kompletteras efter hand med de fåtal nya ord som införs. Det är dock tveksamt om alla användare känner till de nya ordens alla lexikala egenskaper som ju behövs i lexikonet. Vem kan till exempel utan vidare tala om vilken deklination varje ord tillhör?

En möjlighet att komma förbi problemet med nya ord för lexikonet är att låta programmet ställa frågor till användaren med hjälp av exempel. Dessa exempel skulle då ge användaren den information han behöver för att besvara frågorna utan svårighet. Om programmet till exempel frågar efter genustillhörighet, så ges exempel på de två fallen: "ett bord" och "en stol".

Det går i detta sammanhang förstås att komma med motargumentet att lexikoninmatningen görs endast en gång, och att lexikonet därefter kommer att användas flera gånger vid olika modelleringar inom samma tillämpningsområde.

12. Referenser

Bubenko, Janis & Lindencrona, Eva : Konceptuell modellering - Informationsanalys. Studentlitteratur, Lund 1984.

Carlsson, Mats & Widén, Johan: SICStus Prolog User's Manual.

Dahlgren, Harriet : Konceptuell modellering med naturligt språk. SISU Rapport nr 6.

Dalianis, Hercules : Generating a Natural Language. Description and Deduction from a Conceptual Schema. SYSLAB, Working Paper 160, nov 1989.

Ericsson Telecom : EMOL metodhandbok 1988 Ericsson Telecom, Avdelningen för konstruktionsstöd, system och programvara.

Jörgensen, Nils & Svensson, Jan : Nusvensk grammatik. Liber, Malmö 1986.

Wohed, Rolf : "Diagnosis of conceptual schemas". SYSLAB report no. 57 Royal Institute of Technology and the University of Stockholm. To appear in IFIP WG2.6/WG8.1, "The Role of AI in Database and Information Systems", Canton, People´s rep. of China July, 1988.

13. Appendix

13.1 Exempel på körning av programmet

```
korat% sicstus
SICStus 0.6 #15: Mon Oct 23 10:50:58 MET 1989
Copyright (C) 1987, Swedish Institute of Computer Science.
All rights reserved.
| ?- ['source_file2'],['lex_nouns'].
   {consulting /home/korat/jonasw/source_file2...}
   {/home/korat/jonasw/source_file2 consulted, 5680 msec 61330 bytes}
   {consulting /home/korat/jonasw/lex_nouns...}
   {/home/korat/jonasw/lex_nouns consulted, 2220 msec 22638 bytes}

yes
| ?- loop.


 Möjliga alternativ (svara med siffra följt av punkt):
 ====================================

 1. Läs in ett nytt schema och ta bort det senaste.
 2. Lista alla objekt i schemat.
 3. Egenskaper hos enskilda objekt.
 4. Lista alla domäner i schemat.
 5  Lista all information, utan arv etc.
 6. Lista all information, med arv etc.
 7. Ta bort nuvarande schema.
 8. Avsluta körning.
 |: 1.
Vad heter filen som schemat är lagrat på? |: sve_sjukhus.

{consulting /home/korat/jonasw/sve_sjukhus...}
{/home/korat/jonasw/sve_sjukhus consulted, 380 msec 3242 bytes}

Schemat i filen sve_sjukhus är inläst.
```

Listning av objekt

Möjliga alternativ (svara med siffra följt av punkt):
==========

1. Läs in ett nytt schema och ta bort det senaste.
2. Lista alla objekt i schemat.
3. Egenskaper hos enskilda objekt.
4. Lista alla domäner i schemat.
5 Lista all information, utan arv etc.
6. Lista all information, med arv etc.
7. Ta bort nuvarande schema.
8. Avsluta körning.

|: 2.

Schemat innehåller följande objekt:
==========
akademiker
anställd
avdelning
biträde
civ_ingenjör
doktor
högre_examen
nödutgång
patient
person
sjukhus
specialist

Egenskaper hos enskilda objekt

Möjliga alternativ (svara med siffra följt av punkt):
==========

1. Läs in ett nytt schema och ta bort det senaste.
2. Lista alla objekt i schemat.
3. Egenskaper hos enskilda objekt.
4. Lista alla domäner i schemat.
5 Lista all information, utan arv etc.
6. Lista all information, med arv etc.
7. Ta bort nuvarande schema.
8. Avsluta körning.

|: 3.
Ange objekt: |: doktor.

++

doktorer har följande överordnade klasser:
==========
person <- anställd <- doktor
akademiker <- doktor
< >

doktorer har följande underordnade klasser:
==========
doktor <- specialist
< >

Ärvda egenskaper:
==========
från person: varje doktor bär precis ett namn
från akademiker: varje doktor innehar minst en högre_examen
< >

Egna egenskaper :
==========
varje doktor behandlar minst en patient
< >

Endast vissa doktorer har följande egenskaper:
==========
< >

Dessa egenskaper är ärvda av underklassen från en annan överordnad klass:
==========
< >

Möjliga alternativ (svara med siffra följt av punkt):
==========

1. Läs in ett nytt schema och ta bort det senaste.
2. Lista alla objekt i schemat.
3. Egenskaper hos enskilda objekt.
4. Lista alla domäner i schemat.
5 Lista all information, utan arv etc.
6. Lista all information, med arv etc.
7. Ta bort nuvarande schema.
8. Avsluta körning.

```
|: 3.
```

Ange objekt: `|: anställd.`

```
++++++++++++++++++++++++++++++++++++++++++++++++
```

anställda har följande överordnade klasser:
==========
person <- anställd
< >

anställda har följande underordnade klasser:
==========
anställd <- biträde
anställd <- doktor <- specialist
< >

Ärvda egenskaper:
==========
från person: varje anställd bär precis ett namn
< >

Egna egenskaper :
==========
< >

Endast vissa anställda har följande egenskaper:
==========
de som är biträden arbetar på precis en avdelning
de som är doktorer behandlar minst en patient
< >

Dessa egenskaper är ärvda av underklassen från en annan överordnad klass:
==========
de som är doktorer innehar minst en högre_examen
< >

Möjliga alternativ (svara med siffra följt av punkt):
==========

1. Läs in ett nytt schema och ta bort det senaste.
2. Lista alla objekt i schemat.
3. Egenskaper hos enskilda objekt.
4. Lista alla domäner i schemat.
5 Lista all information, utan arv etc.
6. Lista all information, med arv etc.
7. Ta bort nuvarande schema.
8. Avsluta körning.

|: 3.
Ange objekt: |: sjukhus.

++++++++++

sjukhus har följande överordnade klasser:
==========
< >

sjukhus har följande underordnade klasser:
==========
< >

Ärvda egenskaper:
==========
< >

Egna egenskaper :
==========
varje sjukhus har mellan 3 och 22 avdelningar
varje sjukhus har exakt tre nödutgångar
< >

Endast vissa sjukhus har följande egenskaper:
==========
< >

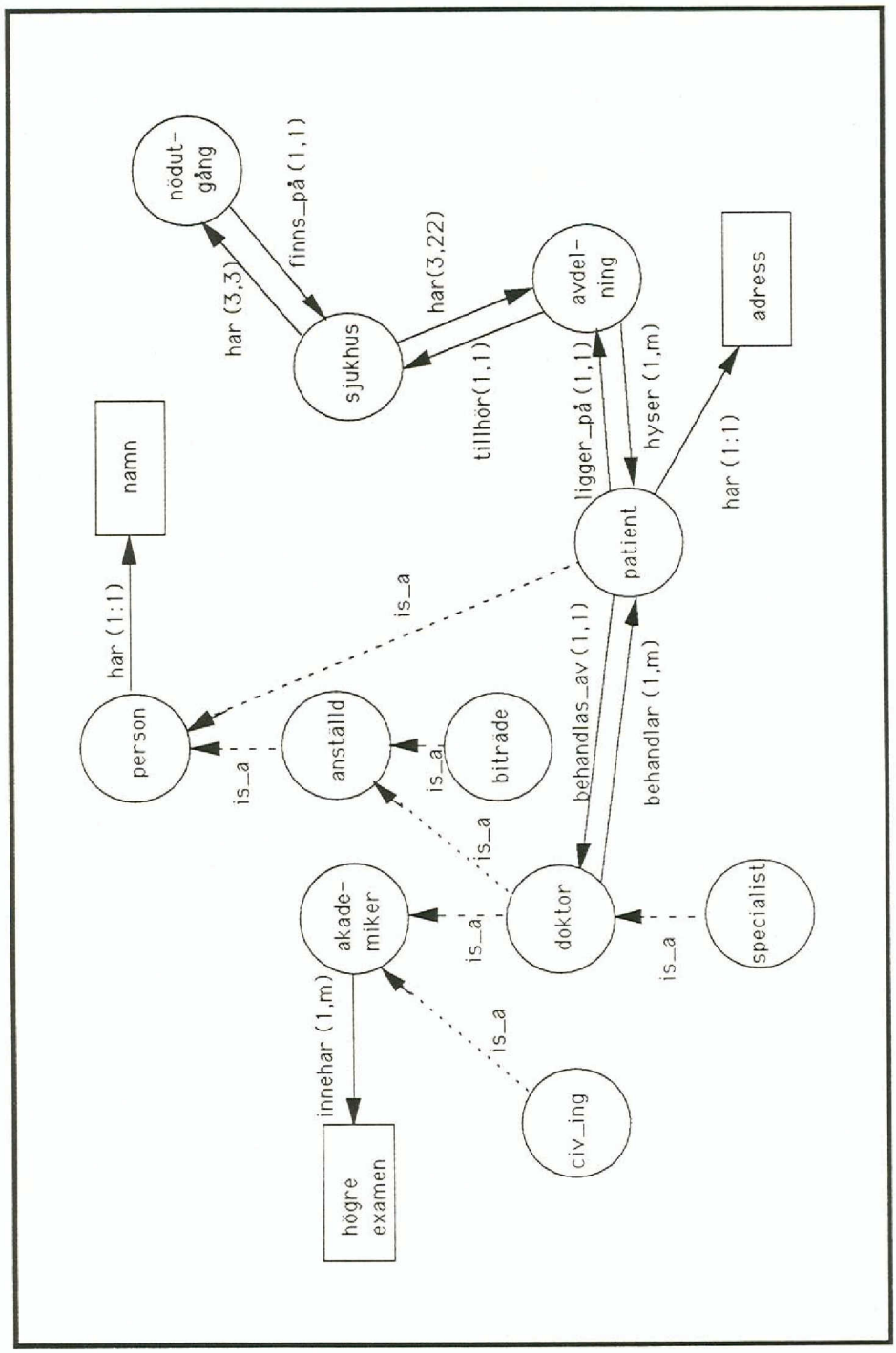
Dessa egenskaper är ärvda av underklassen från en annan överordnad klass:
==========
< >

Listning av domäner

```
Möjliga alternativ (svara med siffra följt av punkt):
=============================================

1. Läs in ett nytt schema och ta bort det senaste.
2. Lista alla objekt i schemat.
3. Egenskaper hos enskilda objekt.
4. Lista alla domäner i schemat.
5  Lista all information, utan arv etc.
6. Lista all information, med arv etc.
7. Ta bort nuvarande schema.
8. Avsluta körning.
|: 4.

************************************************************

Beskrivning av domäner
==================

högre  examen är av typen string
nummer är av typen integer
namn är av typen string
adress är av typen string


Möjliga alternativ (svara med siffra följt av punkt):
=====================================

1. Läs in ett nytt schema och ta bort det senaste.
2. Lista alla objekt i schemat.
3. Egenskaper hos enskilda objekt.
4. Lista alla domäner i schemat.
5  Lista all information, utan arv etc.
6. Lista all information, med arv etc.
7. Ta bort nuvarande schema.
8. Avsluta körning.
|: 8.
Adjö!

yes
|?-
```

13.2 Schema som ligger till grund för genereringen

```
attribute('behandlar','behandlas_av','doktor','patient',mapping(1,m,1,m),_,_).
attribute('ligger_på','har','patient','avdelning',mapping(1,1,0,m),_,_).
attribute('innehar',_,'patient','adress',mapping(1,1,1,m),_,_).
attribute('har','tillhör','sjukhus','avdelning',mapping(3,22,1,1),_,_).
attribute('har','finns_på','sjukhus','nödutgång',mapping(3,3,1,1),_,_).
attribute('arbetar_på','har','biträde','avdelning',mapping(1,1,1,m),_,_).
attribute('bär',_,'person','namn',mapping(1,1,1,m),_,_).
attribute('innehar', _, 'akademiker', 'högre_examen', mapping(1,m,1,m).
entity('anställd',_,_,['person']).
entity('biträde',_,_,['anställd']).
entity('doktor',_,_,['anställd', 'akademiker']).
entity('patient',_,_,['person']).
entity('specialist',_,_,['doktor']).
entity('akademiker',_,_,[]).
entity('civ_ingenjör',_,_,['akademiker'].
entity('person',_,_,[]).
entity('avdelning',_,_,[]).
entity('sjukhus',_,_,[]).
entity('nödutgång',_,_,[]).
data_object('högre_examen',string).
data_object('nummer', integer).
data_object('namn', string).
data_object('adress', string).
```



Figur 10. Grafisk representation av schemat på sid 32.