Förord

Denna skrifts tillkomst beror på det behov som finns att modellera en affärsverksamhets domän. Jag har under årens lopp noterat att behovet finns och att resultatet av både affärsutveckling och utvecklingen av datorstöd i form av system och applikationer;

- blir enklare
- ger högre kvalitet
- blir billigare

Att ta fram dylika modeller elliminerar risken för kommunikationsproblem och missförstånd mellan olika individer, eftersom det är väsentligt enklare att kommunicera och förklara saker med hjälp av bilder och diagram jämfört med att enbart använda ord.

”En grafisk modell säger mer än tusen Kinesiska ordspråk.”

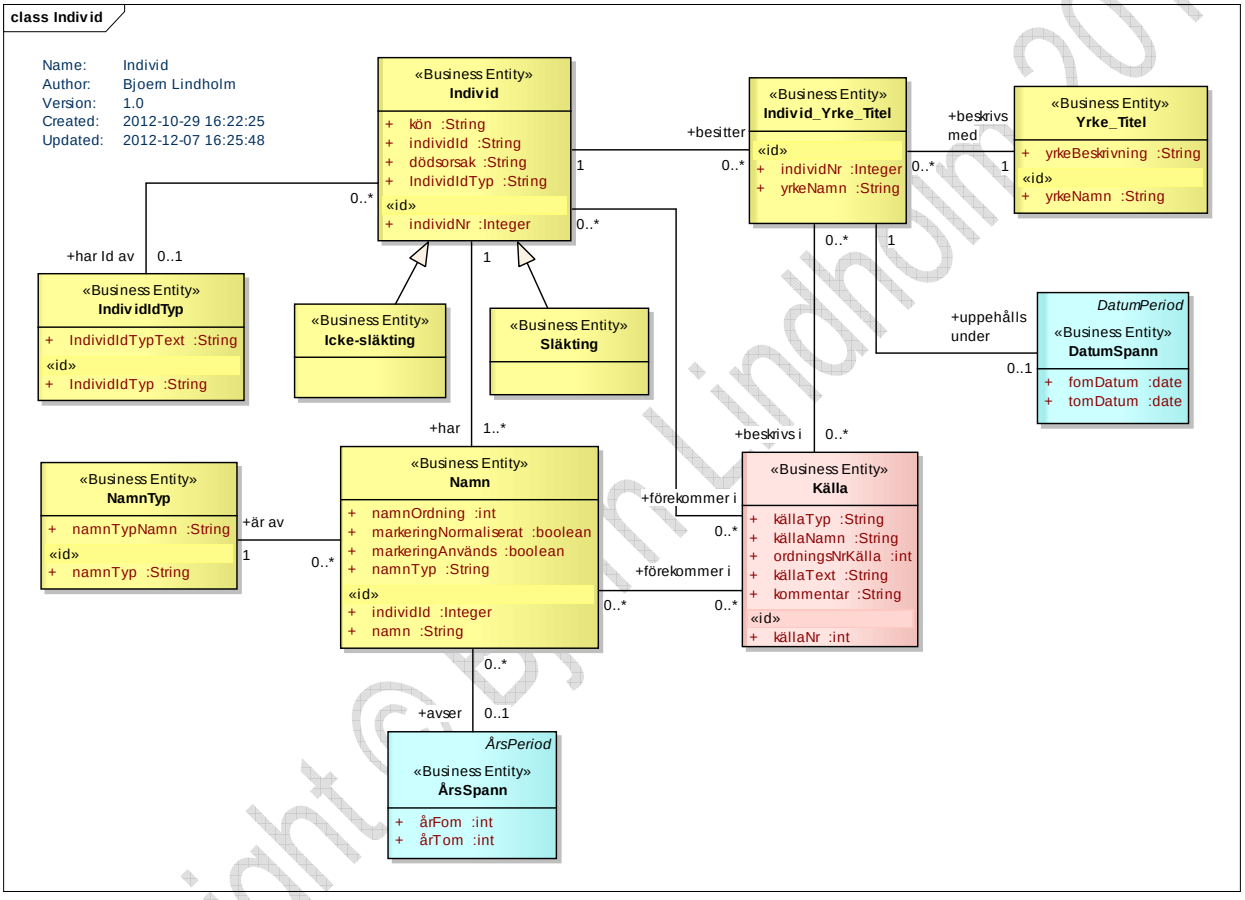
Innehållsförteckning

Vad en domänmodell är

En domänmodell är en modell över en affärsverksamhets objekt. Modellen utformas som en grafisk beskrivning över objekten.

Objekt är många gånger information av olika natur som vi vill hålla reda på. Det kan även vara fysiska saker. Vad som ingår bestäms av den verksamhet vi beskriver. De objekt som vi valt att beskriva i en domänmodell, kallar vi domänobjekt.

Nedan ser vi ett exempel på en del av en domänmodell för verksamheten genealogi (släktforskning).



I modellen ovan är Individ och Källa exempel på fysiska saker, medan DatumSpann och NamnTyp är ren information.

Vi ska senare se på detaljerna i modellen.

Modellen kan beskriva ett befintligt läge (status) ett s k ”nuläge”. Alternativt kan modellen beskriva ett kommande tänkt läge emot viket en utveckling sker, ett s k ”nyläge”. Det är sällan en bra idé att försöka beskriva båda dessa lägen i samma modell.

Varför vi bör göra domänmodeller

Det finns många anledningar till att ta fram en domänmodell. Jag ska nedan försöka lista de viktigaste och förklara varför de är viktiga.

- Skapa en modell över de objekt som är viktiga
- Lösa oklarheter
- Lösa viktiga åsiktsskillnader
- Ge en gemensam syn på verksamheten
- Vara en grund för kravspecifikationer på datorstöd
- Vara en grund för vidare utveckling

En grafisk modell ger en bra bild över verksamheten.

”Vem skulle köpa en akvarell på endast en verbal beskrivning?”

Skapa en modell över de objekt som är viktiga

Oavsett vad det är som vi tänker göra eller skapa, är det alltid väsentligt att vi har klart för oss vilka saker vi arbetar med, vilka domänobjekten är. Detta gäller oavsett om vi exempelvis gör en utredning, ska ta fram krav för en utvecklingsinsatts eller bygga en datorstöd applikation.

Om vi exempelvis ska göra en utredning i ett ämne. Hur ska den kunna presentera ett bra resultat om vi inte har klart för oss vilka saker (domänobjekt) som den berör och hur dessa förhåller sig till varandra? Detsamma gäller i andra situationer. Vi måste känna till och förstå vilka saker (domänobjekt) vi har att göra med.

Många gånger kan vi förledas att tro att vi har kontroll på situationen. Vi kan ämnet och vet vad det rör sig om. I så fall är det ju en bagatell att göra en beskrivning (en modell) och visa den för andra. De som inte vet.

Det är när vi försöker göra en sådan beskrivning som det kan visa sig inte vara så enkelt som vi trodde från början. Vi trodde vi hade allt klart för oss, men när vi ska beskriva det upptäcker vi tveksamheter och oklarheter som vi måste gräva i djupare för att bringa klarhet i. Vi behöver kontrollera med andra människor, kanske kollegor, hur det förhåller sig. Kanske måste vi konsultera någon expert.

När vi skapar en modell bestämmer vi även ett antal saker. Vi bestämmer bl a:

- vilka objekt som är viktiga och ska vara med
- vilka egenskaper dessa objekt har som vi är intresserade av
- vilka relationerna är mellan objekten

Vår modell ska endast innehålla det vi behöver för att klara vår uppgift, d v s den ska inte beskriva saker som finns men inte är relevanta för det vi ska göra.

Lösa oklarheter

När vi ritat upp delar av en modell med domänobjekt och relationer dem emellan är det enklare att diskutera okarheter. Då har vi något att utgå ifrån, en gemensam bild. Detta underlättar mycket i de fortsatta diskussionerna och gör det mycket enklare att bringa klarhet i svåra frågeställningar. Det är då viktigt att fortsätta rita. Ritandet kommer förmodligen inte direkt att leda till en färdig bild, utan är ett arbete som så succesivt leder fram till en bild som alla är överens om. Under arbetets gång har vi säker ändrat modellen flera gånger.

Lösa viktiga åsiktsskillnader

Under arbetet med att rita upp en domänmodell kan vi råka ut för att vi har olika åsikter. Dessa måste vi lösa. Det kan betyda att vi inledningsvis ritar upp de olika varianter vi kan se. Sedan får vi diskutera dessa och se om några kan konstateras vara felaktiga så att vi kan eliminera dessa. Skulle vi få fler varianter kvar än en, måste vi på något sätt bestämma oss för vilken vi väljer. Den slutliga modellen måste ha en lösning. Hur denna beslutsprocess går till varierar från organisation till organisation. Om åsiktsskillnaderna rör en befintlig verksamhet, behöver vi kanske gräva djupare i hur den fungerar. Vi behöver kanske ta in ytterligare kompetenser i arbetet. Om åsiktsskillnaderna däremot rör ett nytt framtida scenario för vår verksamhet, får vi använda vår beslutsprocess.

Ge en gemensam syn på verksamheten

Den slutliga domänmodellen kommer att spegla vår gemensamma syn på vår verksamhet. Den kan då enkelt användas för att beskriva verksamheten för andra i eller utanför organisationen. Genom enkla och tydliga diagram blir det lättare för andra att förstå verksamheten.

Vara en grund för kravspecifikationer på datorstöd

Den färdiga domänmodellen kan bl a andvändas för att beskriva krav på datorstöd, d v s krav på applikationer och system. Det är mycket lämpligt och enkelt att använda domänmodellens domänobjekt och egenskaper i kravspecifikationerna. Ifall det vid en kravspecificering dyker upp nya domänobjekt eller egenskaper som inte finns i den framtagna modellen, måste den kompletteras. Det kan alltså ibland finnas en växelverkan mellan att göra en domänmodell och ta fram kravspecifikationer.

Vara en grund för vidare utveckling

Domänmodellen ska leva vidare och uppdateras i takt med att verksamheten förändras. Vi kan då alltid relatera till den som en beskrivning av vad vår verksamhet består av för saker och hur dessa relaterar till varandra. Om modellen är gjord på ett bra, flexibelt och korrekt sätt, kommer det inte att vara några svårigheter att uppdatera den när den utsätts för förändringar och nya krav.

Domänmodellen utgör även en grund för framtagande av en datamodell. Vidare så förändras naturligtvis datamodellen i takt med att domänmodellen förändras. Datamodellen är dock teknik och används för att skapa databaser, d v s den är inte verksamhetens beskrivning av verkligheten.

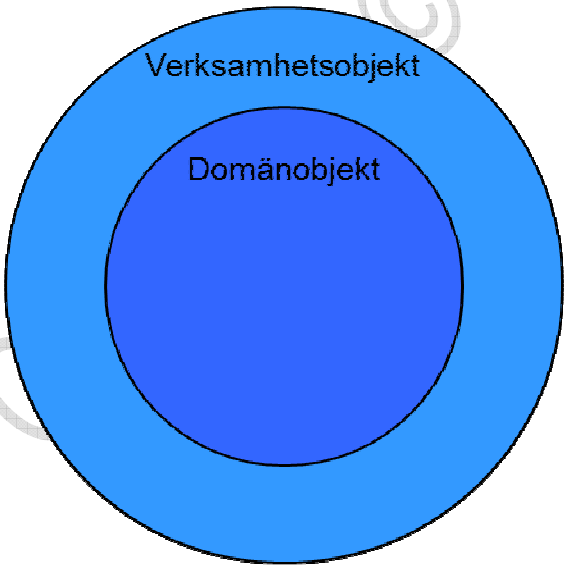
Affärsobjekt kontra domänobjekt

Det finns en viktig skillnad mellan affärsobjekt och domänobjekt. Alla domänobjekt är affärsobjekt, men det är inte tvärt om, d v s endast en del av affärsobjekten är domänobjekt.

Vi kan lämpligen dela in affärsobjekten enligt följande. Vi börjar med att definiera att alla objekt i en verksamhet över huvud taget, som verksamhetsobjekt. I denna del av beskrivningen släpper vi ordet affärsobjekt och använder i st synonymen verksamhetsobjekt. Det ordet är lite bredare eftersom alla verksamheter inte behöver ha ett affärsmässigt syfte. Exempelvis är ofta ideella verksamheter icke affärsdrivande.

Det skulle då kunna se ut enligt bilden nedan.

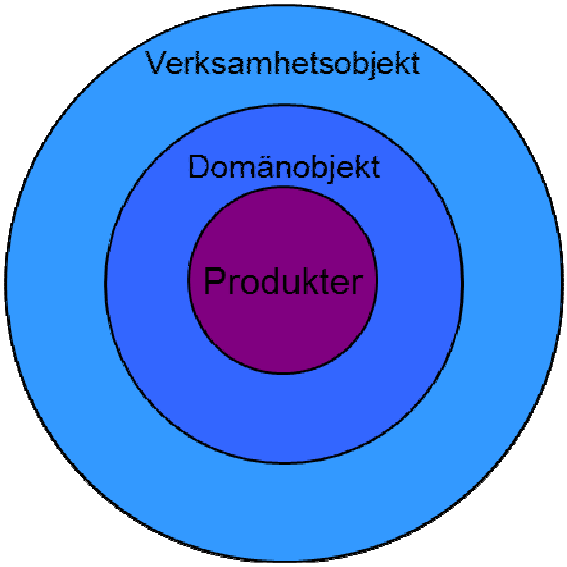
En del av verksamhetsobjekten är domänobjekt.



De är de viktigaste och de vi oftast vill lagra information om i en databas eller eventuellt annan typ av register. Utanför domänmodellen hamnar då kanske verksamhetsobjekt av typen, muntlig information, formulär, rapporter och temporära arbetsresultat.

Vad som ska ingå i en domänmodell bestäms naturligtvis av den som eller de som skapar den. Detta i sin tur beror på vilket syfte modellen har. Görs modellen för att beskriva alla delar i en verksamhet kopplat till manuella och maskinella rutiner, skulle den kunna omfatta alla verksamhetsobjekt. Görs den däremot som en del i ett arbete som syftar till att bygga maskinella datasystem eller applikationer, kommer den endast att innehålla de domänobjekt som blir underlag för databaser.

En del av domänobjekten i en affärsverksamhet är produkterna.



De är de viktigaste objekten av alla. Utan produkter finns ingen affärsverksamhet och inga affärer. En produkt kan vara en sak, d v s något fysiskt. En produkt kan även vara en tjänst som tillhandahålls. Exempelvis kan ett hundhalsband vara en produkt. En tjänst kan exempelvis vara att rasta någons hund en timme.

Om en verksamhet inte är affärsdrivande behöver den inte ha några produkter

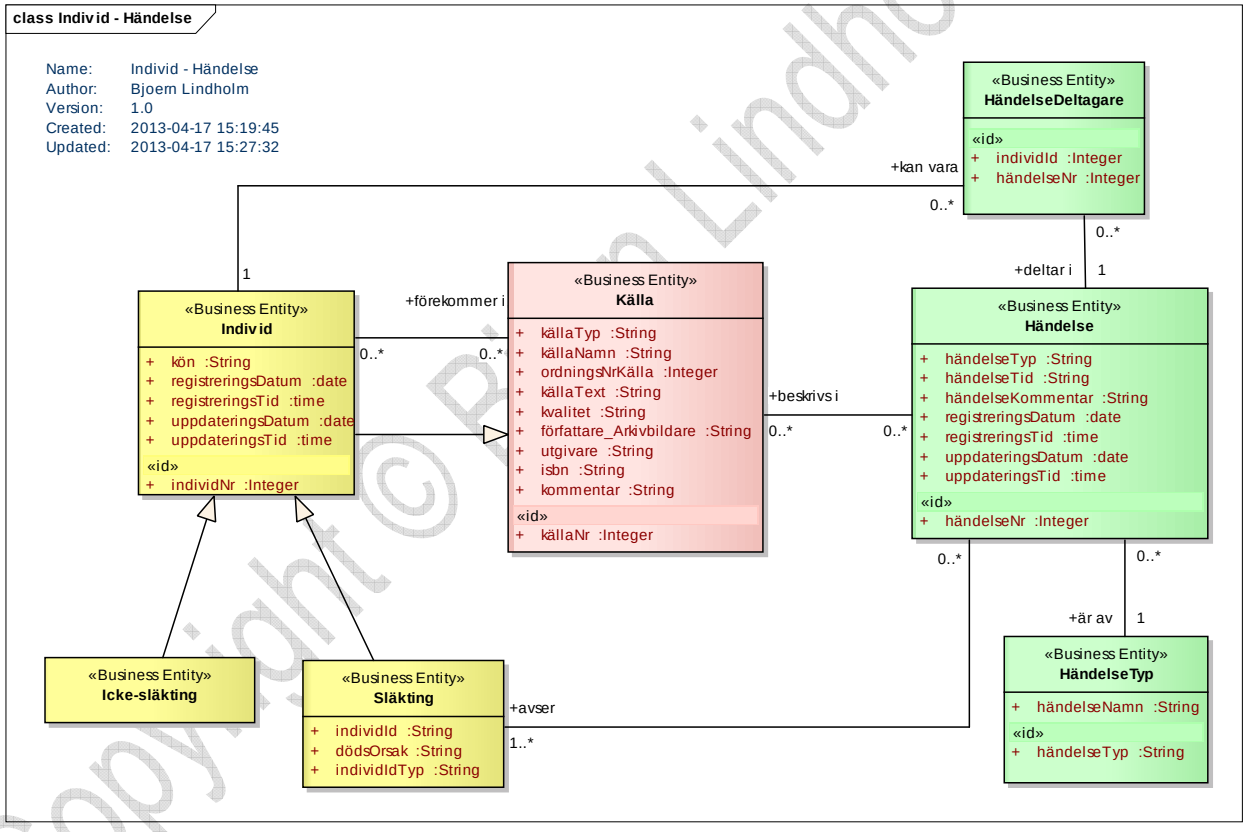
Beståndsdelar i en domänmodell

En domänmodell består enkelt uttryckt av tre delar;

- **Domänobjekt** med sina egenskaper
- Olika typer av **relationer** (kopplingar) mellan objekten
- **Diagram** (eller grafer)

Domänobjekt

Vi börjar med domänobjekten. Domänobjekten beskriver fysiska/konkreta saker eller abstrakta. Exempel på konkreta saker är, bil, båt och hus. Exempel på abstrakta saker är, datum, tid och känslor. Abstrakta saker i en domänmodell representeras oftast som ren information. Vi ska studera ett exempel av varje. I grafen nedan finns dels ”Individ” som är ett fysisk objekt. Där finns även domänobjektet ”Händelse” som är ett rent informationsobjekt. Ur ett genealogiskt perspektiv är en händelse något som har hänt, d v s framtida, kommande, eventuella händelser beskrivs inte.



Domänobjekten ska ha bra och tydliga namn som beskriver vad det är. Varje domänobjekt ska även ha en beskrivande text på en eller flera meningar. Detta för att det inte ska råda några tvivel om vad domänobjektet avser och vilka eventuella avgränsningar som gäller.

När vi modellerar stöter vi på både synonymer och homonymer. Synonymer är som bekant olika ord som betyder ungefär samma sak. Homonymer är ett ord som kan ges flera olika betydelser. ”Källa” är en typisk homonym. Källa kan exempelvis vara en brunn, d v s en vattenkälla. Ordet förekommer även i språkligt bruk i meningen ”en källa till förtret” och betyder då orsak. I exemplet ovan står källa för verk eller skrift eller dylikt där vi hämtat uppgifter, information om individer och händelser.

Den textuella beskrivningen av källa ser ut enligt följande;

” Källa från vilket olika uppgifter, information är hämtade. En källa kan vara av en viss sort eller typ. En källa kan även vara en Individ.
Exempel: Kyrkbok, literatur etc ″

Ibland kan det vara bra att kvalificera namnen för att göra det mer tydligt vad som avses. I detta fall skulle vi kunna använt ordet ”Informationskälla” för att skilja det från exempelvis ”Vattenkälla”. Det hade definitivt behövts ifall vår modell skulle innehålla båda dessa domänobjekt. I andra fall behöver vi inte göra detta eftersom det ofta framgår av sammanhanget vad det är vi avser. Visserligen kan den beskrivande texten på domänobjektet, tydligt beskriva vad det är, men det ska även enkelt gå att se i diagrammet och vi kan inte använda samma namn för flera domänobjekt.

Domänobjektet ”Händelse” är ett rent informationsobjekt. Det är inget fysiskt. Det beskriver i detta exempel något som har hänt och som en eller flera individar varit med om. Den textuella beskrivningen av domänobjektet händelse ser ut enligt följande;

”En händelse som inträffar vid en vis tidpunkt och berör en eller flera individer. Exempelvis födelse och giftemål.

Kan även avse olika typer av händelser i ett engagemang som en individ har haft under sin levnad.
Exempelvis: befordran, utnämning.”

Identifierare

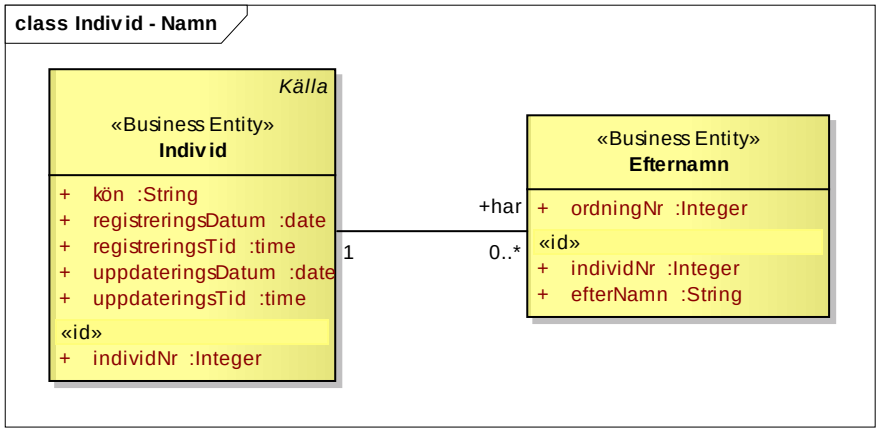
Varje domänobjekt måste vidare ha minst en identifierare (kan vara flera). En identifierare är ett attribut som **unikt** identifierar varje förekomst av domänobjektet. Domänobjektet ”Individ” i grafen ovan har som identifierare ”id” attributet individNr. Detta betyder att varje förekomst av en individ har ett unikt nummer.

Det är uppenbart att vi inte kan ha namn som identifierare eftersom det finns många individer som har samma namn. Det hjälper inte om vi tar både för och efternamn som identifierare, det finns ändå många med samma namn. Björn Lindholm ger exempelvis 44 träffar på hitta.se. Kan vi då inte ha personnummer som identifierare om vi tar med hela årtalet? Det kan kanske fungera i vissa sammanhang, men inte i ett genealogiskt av flera skäl. Dels är det en modern företeelse. Gustav Wasa hade exempelvis inget personnummer vad jag vet. Vidare så har inte personer utanför Sverige personnummer.

Ytterligare ett krav på identifierare är att de ska vara stabila över tiden och alltså inte får ändras. Om identifierare kunde förändras över tiden skulle de bli mycket besvärliga att hantera i våra databaser. Det skulle även uppstå en stor osäkerhet över om dubbletter skulle kunna förekomma, vilket absolut inte får ske. Håller vi reda på data under lång tid fungerar det inte heller. På 1970-talet lagrade många information och identiferare för individer med hjälp av personnummer 10 siffror, d v s utan sekelsiffrorna. Det fungerade så länge ingen blev över 100 år, men sen sprack det. Nu använder man hela årtalet.

Inför millennieskiftet fans ett stort och dyrt problem att åtgärda. På många platser i verksamheter lagrades datum med endast 2 siffror för årtal. Ett mycket kostsamt problem att åtgärda.

Detta betyder att exempelvis en bils registreringsnummer inte är en bra identifierare av en bil. Däremot fungerar förmodligen bilens chassinummer fint, kanske i kombination med fabrikat och modell. Om så skulle vara fallet så har vi alltså multipla identifierar, d v s 3 stycken. Ett annat exempel på multipla identifierare ser vi i diagrammet nedan.



Där kan vi se att domänobjektet Efternamn har 2 identifierare. Det är individNr och efterNamn. Båda behövs eftersom efternamn inte är unika och eftersom en individ kan ha flera efternamn.

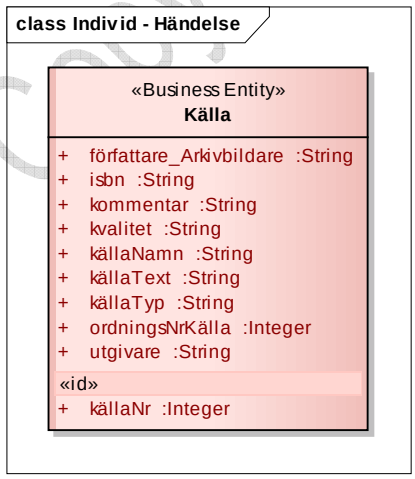
Domänobjektens egenskaper

Till varje domänobjekt knyter vi sedan ett antal egenskaper, attribut. Det är egenskaper som vi önskar hålla reda på och behöver i den verksamhet som vi beskriver. I exemplet ovan (2 diagram bakåt) har vi på domänobjektet Individ egenskapen ”kön”, som är viktig att hålla reda på i ett genealogiskt perspektiv. Vi har inte med ögonfärg och längd, vilket också skulle kunna vara attribut för en individ. De är helt enkel inte intressanta. I en annan verksamhet skulle de kunna vara det. Det gäller alltså att ta med de egenskaper som är viktiga för den verksamhet vi beskriver och inget annat.

Egenskaper ska helst även vara stabila över tiden, som exempelvis en individs ögonfärg. Om egenskapen inte är det, måste vi fundera över om vi ska bryta ut den och skapa ett eget domänobjekt. Ett exempel på detta är en individs längd som varierar över tiden. Vill vi hålla reda på hur lång en individ var vid olika datum måste vi skapa ett eget domänobjekt för detta och koppla längd till datum.

Det är även viktigt att egenskaperna är kopplade till identifieraren eller identifierarna och att dessa unikt pekar ut egenskapen. Exempelvis är inte skostorlek en egenskap hos en individ utan en egenskap hos skor. Individer har däremot egenskapen fotstorlek. Fotstorlek kan sedan uttryckas i olika måttenheter.

I domänobjektet källa som syns i bilden nedan finns förutom identifieraren ett antal egenskaper definierade.



Varje egenskap som domänobjektet har ska beskrivas med text, så att det klart och tydligt framgår vad som menas. Ingen person kan förstå vad som menas med egenskapen ”kvalitet” om det inte klart och tydligt beskrivs. Dessutom bör för varje egenskap en beskrivning finnas över vilka värden, domänvärden en egenskap kan anta. Det kan vara en uppräkning av värden eller en beskrivning av ett värdeområde som ett spann.

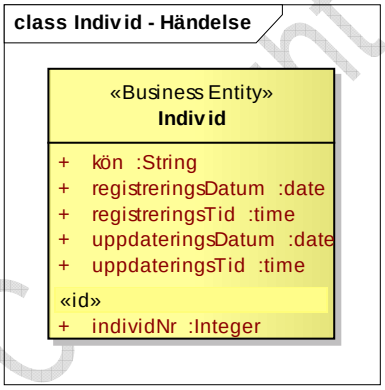
Beskrivningen av egenskapen ”kvalitet” ser ut enligt följande;
” Kvaliteten på källinformationen, d v s hur tillförlitlig och säker den är.
Domänvärde:
Primär
Sekundär
Säker
Osäker ”

Ett domänobjekt kan ha många egenskaper, men varje förekomst av ett objekt behöver inte ha alla egenskaper. Vi vet t ex att många böcker har ett ISBN-nummer, d v s egenskapen ”isbn” på domänobjektet ovan. Det finns dock gamla böcker som inte har detta och dessutom andra typer av källor där isbn-nummer inte är relevant. Det går naturligtvis bra att beskriva i texten för en egenskap om den är obligatorisk eller inte, och det är viktigt att vi gör det. Vi måste dokumentera all kunskap vi har. Vi kan sedan ta med oss den informationen till datamodelleringen. Där kan vi för en kolumn i en tabell sätta ”not null”, d v s att den ska vara obligatorisk.

Domänobjektet ”Källa” har alltså en ganska vid betydelse i det avseendet att en källa inte behöver ha alla egenskaperna.

Senare när vi ska omvandla vår domänmodell till en datamodell går vi igenom ett antal faser som kallas normalisering och senare denormalisering. Vi får då en noggrann analys av huruvida vi bör skilja på källor av olika typ.

I en domänmodell kan även domänobjekt tilldelas attribut som egentligen inte är egenskaper hos själva objektet, men som är viktiga för den verksamhet vi hanterar. Om vi tittar på objektet ”Individ” nedan kan vi se några sådana.



Det handlar om de olika attributen för datum och tid. Jag har valt att kalla dem för attribut eftersom de inte är några egenskaper hos domänobjektet individ. Naturligtvis kan man säga att alla delar är attribut, men det kan vara bra att hålla reda på skillnaderna. Attributen för datum och tid har kommit till genom de krav på funktionalitet som användare av genalogiska system och applikationer har ställt.

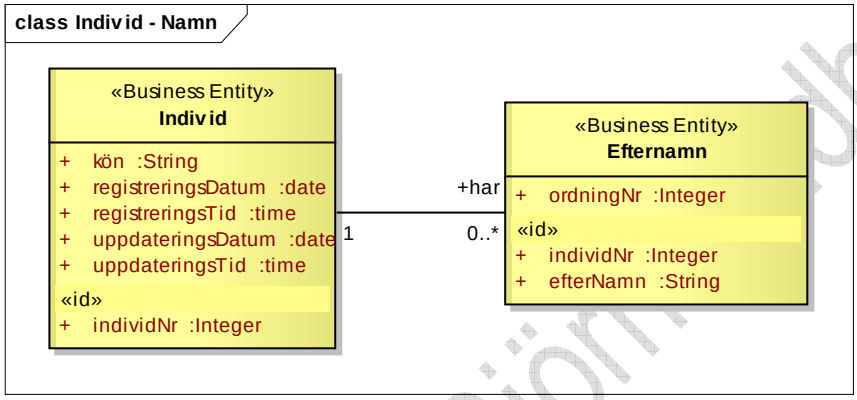
Jag kommer senare att göra en separat beskrivning av hur vi kan arbeta med användar- och beställarkrav kopplat till en domänmodell.

Multipla egenskaper

Ibland kommer vi stöta på att ett domänobjekt har multipla egenskaper. Låt oss ta exemplet med individ ovan. En egenskap för individ är ”efternamn”. Vi skulle alltså i vårt domänobjekt Individ kunna lägga till egenskapen efternamn. Vi kommer då att märka att det inte fungerar så bra eftersom det finns individer som har 2 efternamn. Vi kan då lägga till ytterligare en egenskap ”efternamn2”. Men det finns ju faktiskt individer som har fler än 2 efternamn också. Hur ska vi då göra. Om vi antar att det räcker med 5 efternamn så kan vi naturligtvis skapa 5 egenskaper efternamn1 – efternamn5, och vi kommer förmodligen att klara oss ganska bra.

Detta är dock ingen bra lösning eftersom vi inte kan vara helt säkra, plus framför allt att de flesta egenskaperna 2-5 kommer att vara tomma för de flesta domänobjekten i verkligheten.

En bättre och flexiblare beskrivning är att skapa ett eget objekt för efternamn enligt bilden nedan.



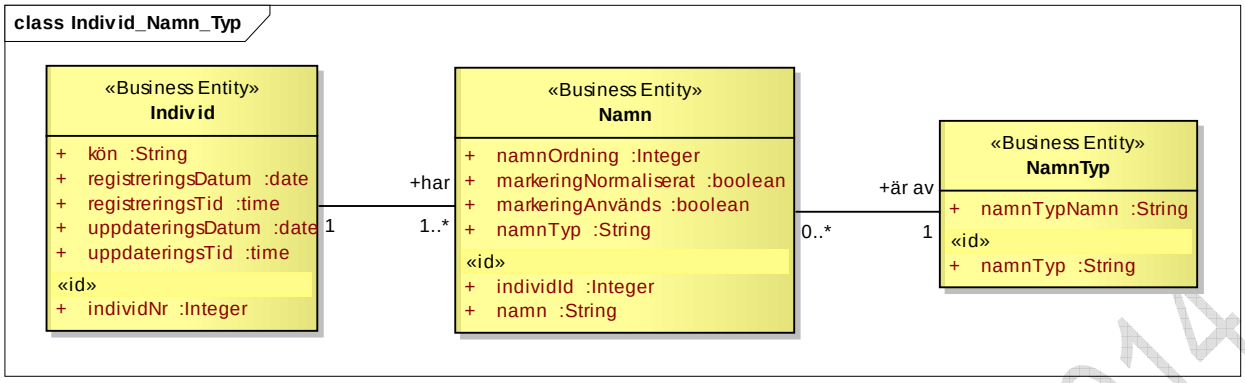
Här ser vi att domänobjektet ”Efternamn” har 2 identiteter för att bli unikt. De är individNr och efterNamn. Flera individer kan ha samma efternamn så därför måste vi även ha med individnummer.

Denna lösning gör att en individ teoretiskt kan ha hur många efternamn som helst. Domänobjektet efternamn har även fått egenskapen ”ordningsNr”, för att vi ska kunna hålla reda på i vilken ordning en individ skriver sina olika efternamn.

Klassificerade eller typade egenskaper

Men en individ har väll inte bara efternamn? Nej individer har också förnamn. Ska vi då skapa ytterligare ett domänobjekt för förnamn? Och hur är det med smeknamn? Kan det finnas flera typer av namn?

När vi stöter på detta problem löser vi det enklast genom att göra ett generellt domänobjekt som vi kan klassificera, d v s säga är av en viss klass eller typ. I modellen för genealogi ser det ut enligt följande diagram.



Domänobjektet heter här enbart ”Namn”. Det har även fått en ny egenskap som heter ”namnTyp”. Denna egenskap talar om vad det är för typ av namn som avses. Nedan syns beskrivningen av egenskapen ”namnTyp” i modellen.

” Typen av namn.
Domänvärde:
Efternamn
FlickPojknamn
Förnamn
Tilltalsnamn
Smeknamn”.

I diagrammet ovan har även ett eget domänobjekt lagts till för själva klassificeringen eller typningen. Det har identiteten ”namnTyp” och en egenskap ”namnTypNamn” som beskriver vad som avses för varje domänvärde av namnTyp. Detta domänobjekt finns för att vi endast en gång ska behöva beskriva vad varje namnTyp avser (namnTypNamn).

Genom att tillföra ett sådant s k typ-domänobjekt (för typning/klassificering) har vi även skapat full frihet för att i ett senare skede enkelt kunna lägga till nya domänvärden som vi kan behöva utan att behöva ändra i några databaser (eller i bästa fall program) av just den anledningen. Domänobjektet ”Namn” kan inte anta några domänvärden för egenskapen namnTyp än de som är definierade i domänobjektet NamnTyp. Mer om detta nedan i beskrivningen av relationer.

Relationer kopplingar

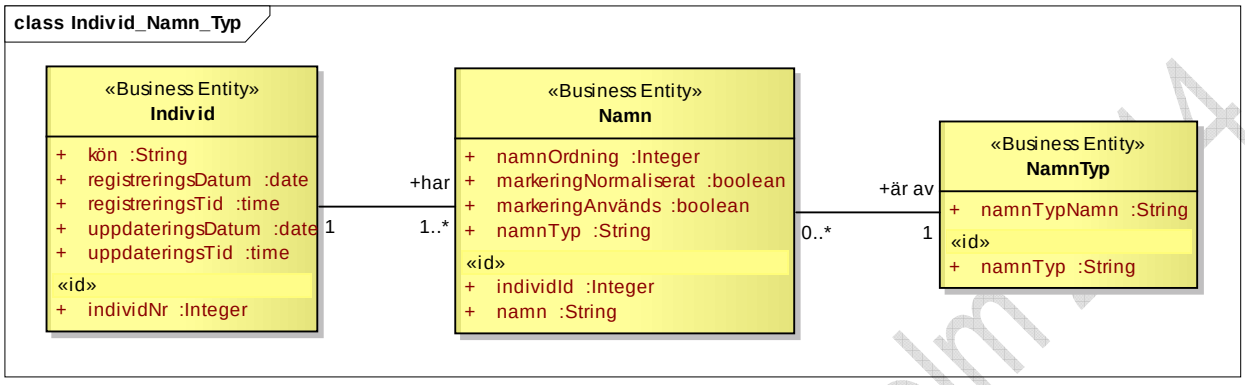
Relationerna beskriver hur olika domänobjekt hänger samman i vår verksamhet. Här gäller samma grundregel som när vi beskriver ett domänobjekts egenskaper, d v s att vi endast ska beskriva relationer som är relevanta i vår verksamhet. Vi ska heller inte beskriva relationer som kan underlätta och snabba upp sökningar. Detta blir senare kompletteringar i vår datamodell utifrån kraven på det system eller den applikation som ska byggas.

Vi kommer att använda oss av 3 grundläggande typer av relationer. Dessa är;

- Association
- Aggregat och Komposit
- Specialisering, generalisering

Association

En association är en enkel relation mellan två domänobjekt. I diagrammet nedan ser vi två stycken relationer av typen association. Den ena är mellan domänobjekten Individ och Namn och den andra mellan Namn och NamnTyp.

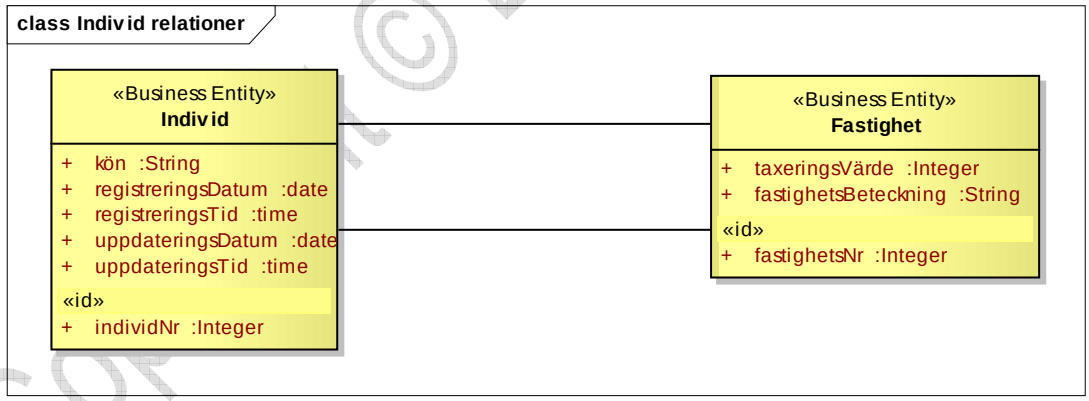


På relationerna har det i ena riktningen påförts ett litet ord med ett plus-tecken framför (plustecknet har ingen betydelse utan har i detta fall satts av programvaran jag använt för modelleringen). Det lilla ordet är till för att öka läsbarheten och förståelsen av modellen. Det talar om vad det är för en sorts relation som vi avser.

Vi kan alltså i detta fall läsa modellen enligt följande;

Individ har Namn
och
Namn är av Namntyp

Det kan tyckas enkelt och klart, men titta på följande exempel.

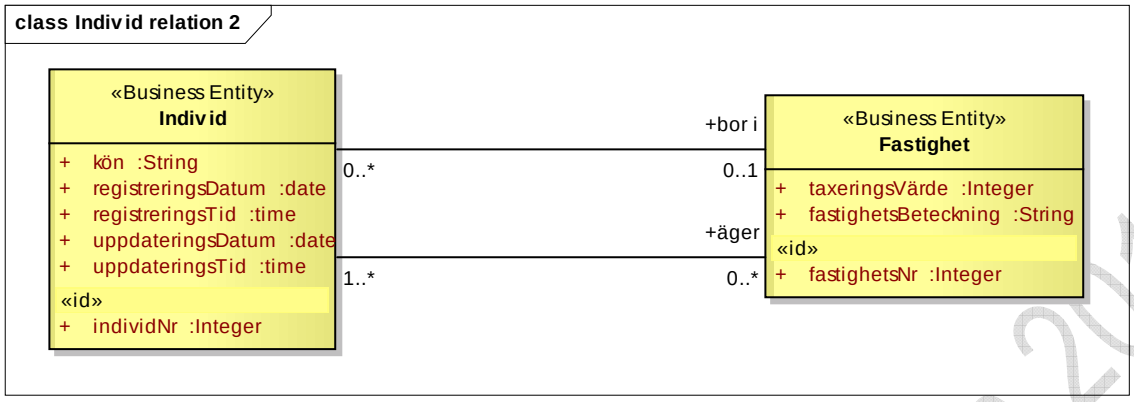


Vi ser att det finns två relationer mellan Individ och Fastighet, men vad står relationerna för. Den ena skulle exempelvis kunna vara att en Individ har **byggt** en Fastighet och den andra att en Individ har **ritat** (är arkitekten bakom) en Fastighet.

Nedan har vi samma diagram, men nu med namn på relationerna.

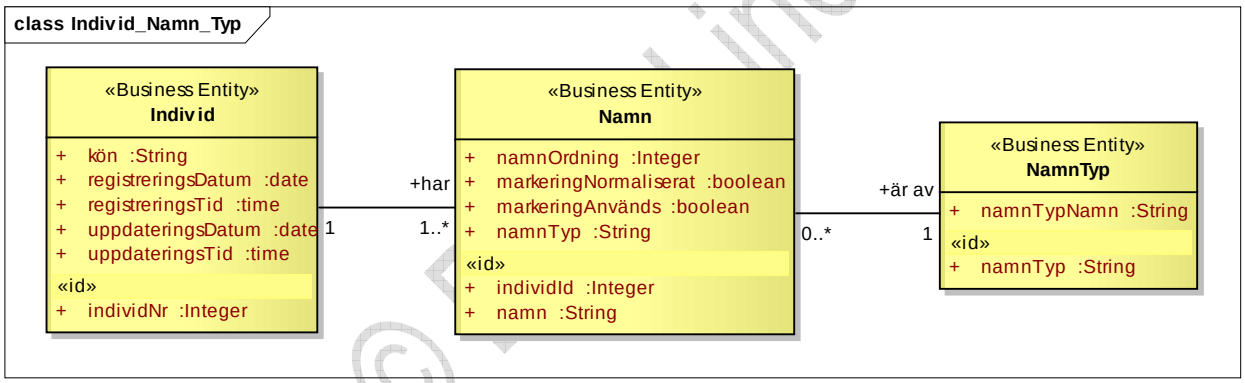
Vi kan nu konstatera att den ena relationen avsåg att beskriva individer som bor i fastigheter och den andra individer som äger fastigheter. Namnsättningen av relationer är alltså mycket viktig och helt

avgörande för förståelsen av modellen. Av detta exempel kan vi även förstå att det kan finns många olika typer av relationer mellan två domänobjekt.



Multiplicitet

Multiplicitet är ett ord för att beskriva hur många av ett domänobjekt som relaterar till hur många av ett annat domänobjekt.



I diagrammet ovan ser vi att för associationen ”har” finns vid domänobjektet namn texten ”1..*”. Detta utläses ”ett till många” eller ”en till många”. Vi kan alltså läsa modellen enligt följande;

Individ har **ett till många** Namn

I andra ändan av samma association (nära domänobjektet Individ) står det ”1”. Detta utläses ”ett/en”. Det kan vi även använda när vi läser modellen, som då blir enligt följande;

En Individer har **ett till många** Namn

Orden ”till många” kan i flera fall i svenskan bytas emot ”eller flera”. Vi skulle då få följande mening;

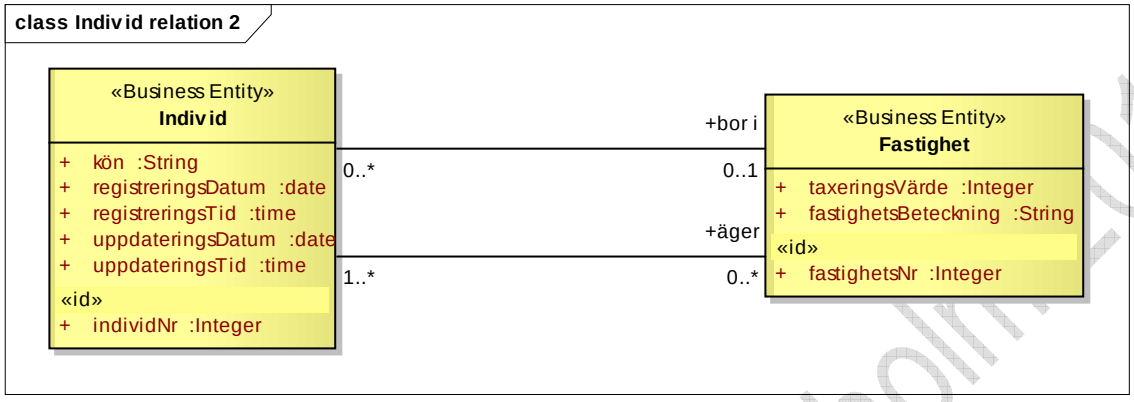
En Individ har **ett eller flera** Namn

Associationen mellan Namn och namntyp utläses;

Noll till många Namn är av **en** Namntyp

Av detta framgår alltså att ett Namn alltid måste vara av en NamnTyp. Vidare att en viss NamnTyp kan vara använd för noll eller flera Namn. Det betyder alltså att många namn kan vara av samma NamnTyp. Dessutom betyder noll i detta sammanhang att det kan finnas en NamnTyp som ännu inte används av något Namn.

Vi ska nu fortsätta med att åter igen titta på associationerna mellan Individ och Fastighet.



Vi börjar med den översta av associationerna i diagrammet ovan, d v s;

Noll till många Individer bor i **noll till en** Fastighet.

Nollan vid Fastighet betyder att en Individ **inte behöver** bo i en Fastighet. **Ettan** betyder att en Individ endast kan bo i **en** Fastighet. **Nollan** vid Individ betyder att en Fastighet **inte behövera ha några** Individer som bor i den. **Asterisken** vid Individ betyder att en Fastighet kan ha **många** individer som bor i den.

Den andra associationen kan vi utläsa enligt följande;

En eller flera (en till många) Individer äger **noll till många** Fastigheter

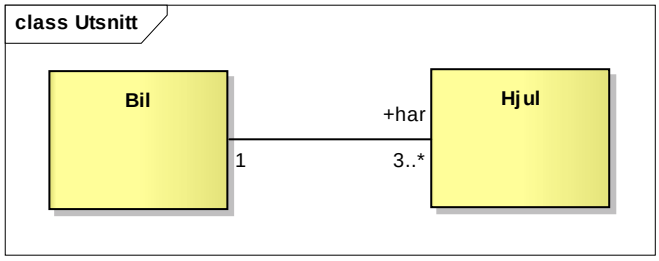
Vi kan ur modellen ovan nu även konstatera ett antal fakta för vår verksamhet. Eftersom båda associationerna till fastighet kan vara 0 (noll), behöver alltså en individ inte ha någon relation till någon fastighet. Vidare kan vi om vi studerar relationerna från fastighet emot individ se att relationen äger har en etta (1). Detta betyder alltså att det för varje fastighet som vi har, måste finnas minst en individ som ägare.

Den här sista associationen sägs också vara av typen många till många eftersom det kan vara många åt båda hållen (asterisk i båda ändarna). Så kan det vara i verkligheten och vi måste beskriva det i vår modell på detta sätt. I en datamodell däremot måste detta förhållande lösas upp. Det är alltså helt korrekt att ha med associationen på detta sätt i vår domänmodell.

De vanligaste förekommande typerna av multiplicitet är följande;

0..1
0..*
1
1..*

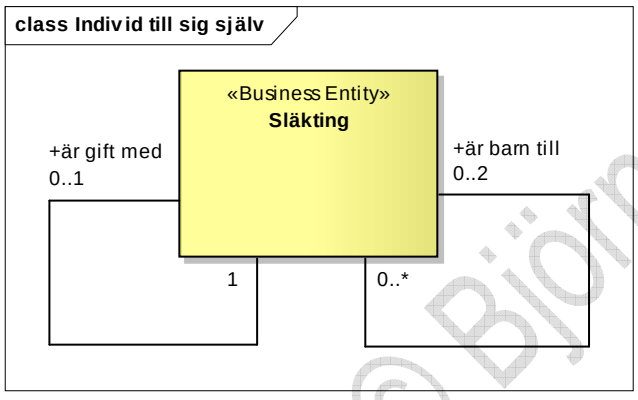
Multipliciteten ska beskriva så noga som möjligt och det ska även här vara den verksamhet som vi beskriver som styr. Titta på exemplet nedan.



Av diagrammet kan vi utläsa att, **en** bil har **3 eller flera** (många) hjul. Vi har därmed klargjort att det inte finns några bilar som har färre än 3 hjul. Samtidigt har vi inte satt några gränser uppåt för hur många hjul en bil får ha i vår verksamhet.

Association till sig själv

Domänobjekt kan ha relationer till sig själva. Det kan vara en eller flera. Nedan finns ett exempel för domänobjektet Släkting. Det skulle lika gärna ha kunnat varit ett domänobjekt Individ, men jag har i detta exempel valt att knyta an till en genealogisk domänmodell.



Från den vänstra associationen kan vi utläsa att, **en** Släkting är gift med **noll eller en** Släkting. En släkting behöver alltså inte vara gift med någon annan släkting (0), men om den är det så kan det endast vara med en (1). Vår verksamhet kan alltså inte hantera mångiften.

Den andra associationen säger att, **noll till många** Släktingar är barn till **noll till 2** Släktingar. Här hanterar vi endast blodsband. Varför står det då noll till 2? Måste inte alltid ett barn ha 2 biologiska föräldrar? Jo det är förvisso sant, men i våra modeller kommer det ofta att stå noll (till något) eftersom vi kan sakna den fulla informationen och måste kunna hantera delmängder av information. Det betyder exempelvis i fallet ovan att vi kanske bara känner till den ene föräldrarna eller **inte någon** av dem.

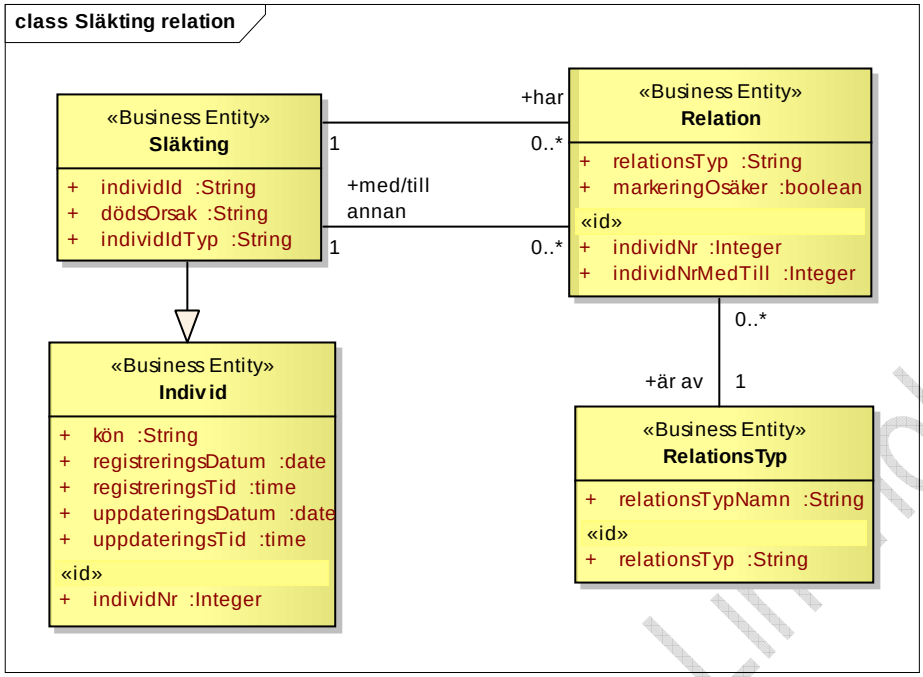
Relationsobjekt

I de fall där vi har många relationer mellan två domänobjekt, kan vi istället skapa ett s k relationsobjekt. I fallet ovan med domänobjekten Individ och Fastighet, diskuterade vi 4 relationer,

- har ritat
- har byggt
- äger

- bor i

At skapa relationsobjekt är även bra om ett domänobjekt har många relationer till sig självt. Ett exempel från den genealogiska modellen ser ut enligt nedan.



Släktingar kan ha flera olika typer av relationer till varandra. Det kan då vara praktiskt att visa detta med ett domänobjekt som i detta fall är ”Relation”. De två relationerna mellan domänobjekten Släkting och Relation kan vi utläsa enligt följande.

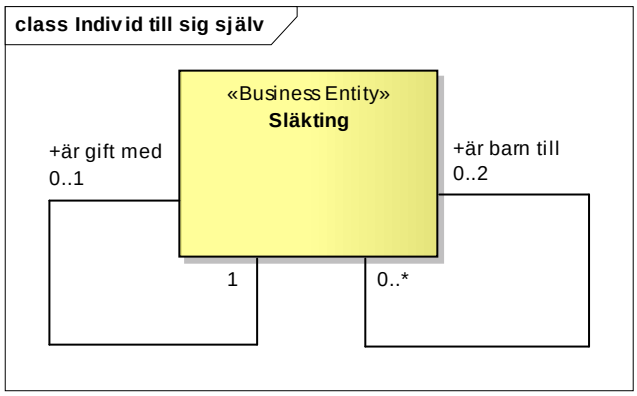
Släkting har **en eller flera** Relation(er) **med/till annan** Släkting. Ordet ”annan” finns i meningen för att det ska bli bättre Svenska, men även för att tydliggöra att det inte kan handla om att någon släkting skulle kunna ha en relation till sig själv. Av diagrammet framgår även att ett domänobjekt Relation alltid måste ha 2 relationer till domänobjekt Släkting.

Varje relation är sedan typad eller klassificerad med hjälp av en Relationstyp. Egenskapen relationsTyp beskriver vad det handlar om för relation. Så här ser beskrivningen ut.

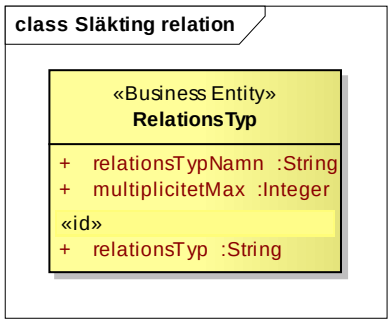
Typ av relation.
Domänvärde:
Biologiskt barn
Adoptivbarn
Forterbarn
Gift
Sambo
Särbo
Förlovad
Trolovad
Fadder
Gudmor

Fördelen är att vi i denna modell enkelt kan lägga till nya relationstyper, utan att vår modell måste ritas om. Nackdelen är att vi har tappat noggrannheten i beskrivningen. Om vi jämför med den förra bilden, se nedan, kan vi konstatera att informationen om den specifika multipliciteten för de olika relationerna gått

förlorad. Vi hade **en** släkting är gift med **noll eller en** släkting och **noll till många** släktingar är barn till **0 till 2** släktingar.



Vi kan dock lösa detta genom att tillföra en egenskap på domänobjektet som vi kallar multiplicitetMax. Domänobjektet RelationsTyp skulle då kunna ha bl a egenskaperna relationsTyp och multiplicitetMax.

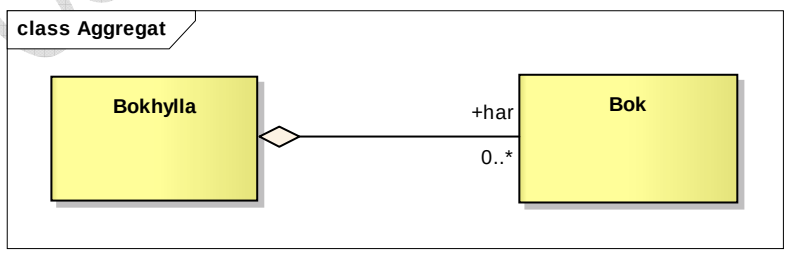


Det skulle då exempelvis kunna se ut enligt följande tabell för några av domänvärdena.

relationsTyp	multiplicitetMax
Biologiskt barn	2
Gudmor	1

Aggregat och Komposit

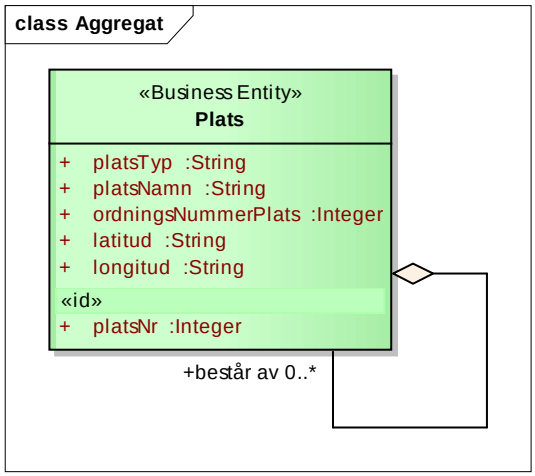
Aggregat och komposit är två vanligt förekommande relationer i bl a domänmodeller. De kan sägas vara två speciella typer av associationer eller relationer mellan två domänobjekt. Vi börjar med aggregat. Det kan exempelvis se ut enligt bilden nedan.



Vi utläser relationen enligt följande. **Noll eller en** bokhylla har **noll eller många** böcker. Ett annat sätt att uttrycka det mer allmänt är att en bokhylla består av böcker. Aggregat kan allmänt sägas vara en sort av ”består av” relation, en sammansättning.

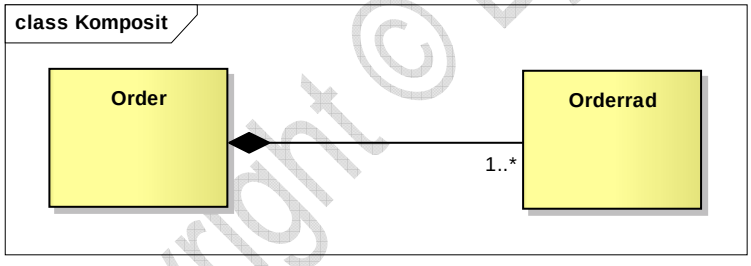
Vi kan alltså konstatera av multipliciteten 0..* att en bokhylla inte behöver ha några böcker, men att den kan ha många. Vidare betyder den **oifyllda diamanten till vänster 0..1!** Detta betyder följaktligen att en bok kan finnas i en bokhylla (1) men att den inte behöver göra det (0).

Från släktmodellen har vi följande aggregat.



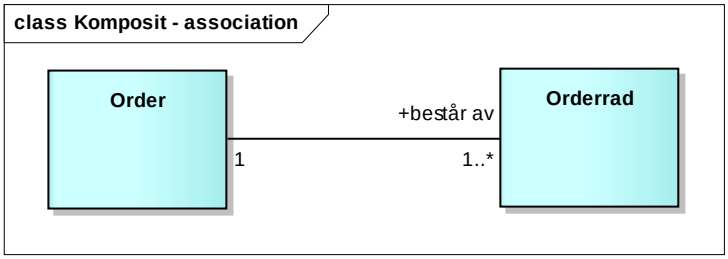
Här har vi ett domänobjekt som aggregerar till sig självt. I detta fall betyder det att en Plats kan bestå av ett antal platser som kan bestå av ett antal platser o s v. Om vi tar Sverige som exempel så skulle det kunna betyda att Ett land består av Län som består av kommuner som består av Församlingar o s v.

Nu till komposit som liknar aggregat. I bilden nedan syns ett exempel på komposit.



Vi utläser relationen enligt följande. **En** Order består av **en till många** Orderrader. Komposit är alltså en ifylld diamant. Det speciella här är alltså att en orderrad alltid måste finnas på **en** Order. Orderrad kan inte finnas för sig själv utan att det finns en order. Det verkar kanske självklart, men beror mycket på att vi naturligtvis förstår verksamheten. Det betyder i konsekvens att om vi tar bort en order så tar vi även bort alla dess orderrader.

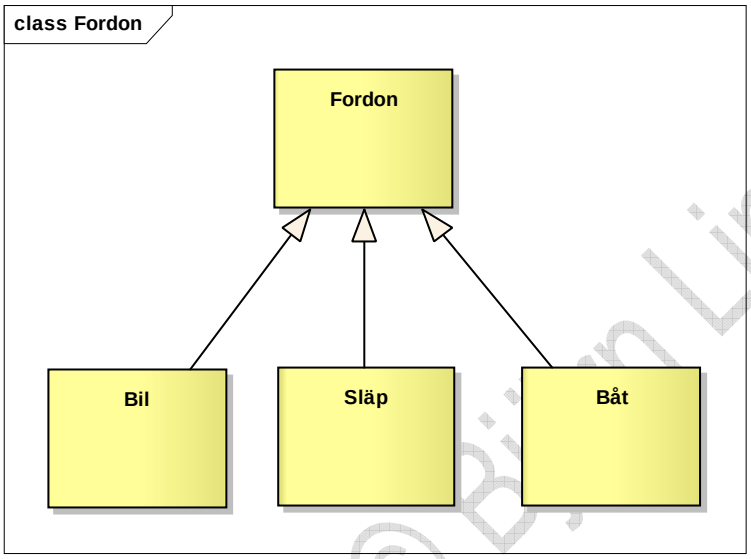
Till sist kan vi fråga oss om vi verkligen behöver dessa speciella relationer. Svaret är nej. Vi skulle mycket väl kunna uttrycka samma sak med multiplicitet på en vanlig association och texten ”består av”, som i exemplet nedan.



Trots allt kanske aggregat och komposit är något tydligare. Du avgör.

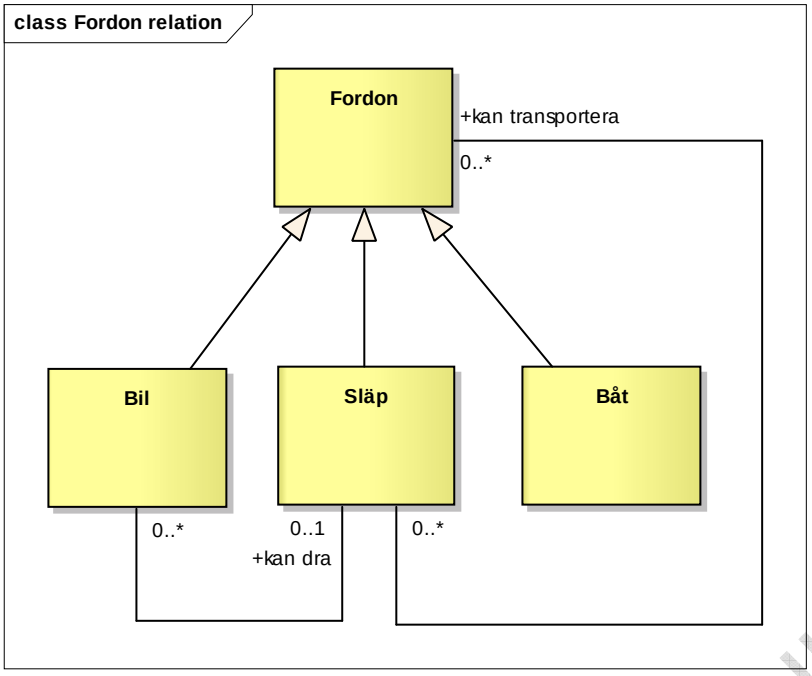
Generalisering – Specialisering

Detta är en typ av relation där ett generellt objekt delas upp i ett eller flera specialiserade objekt. Nedan ser vi ett exempel.



I exemplet ovan är Fordon det generella domänobjektet. Bil, båt och släp är olika typer av fordon och med andra ord specialiseringar av fordon, eller om vi så vill specialiserade typer av fordon.

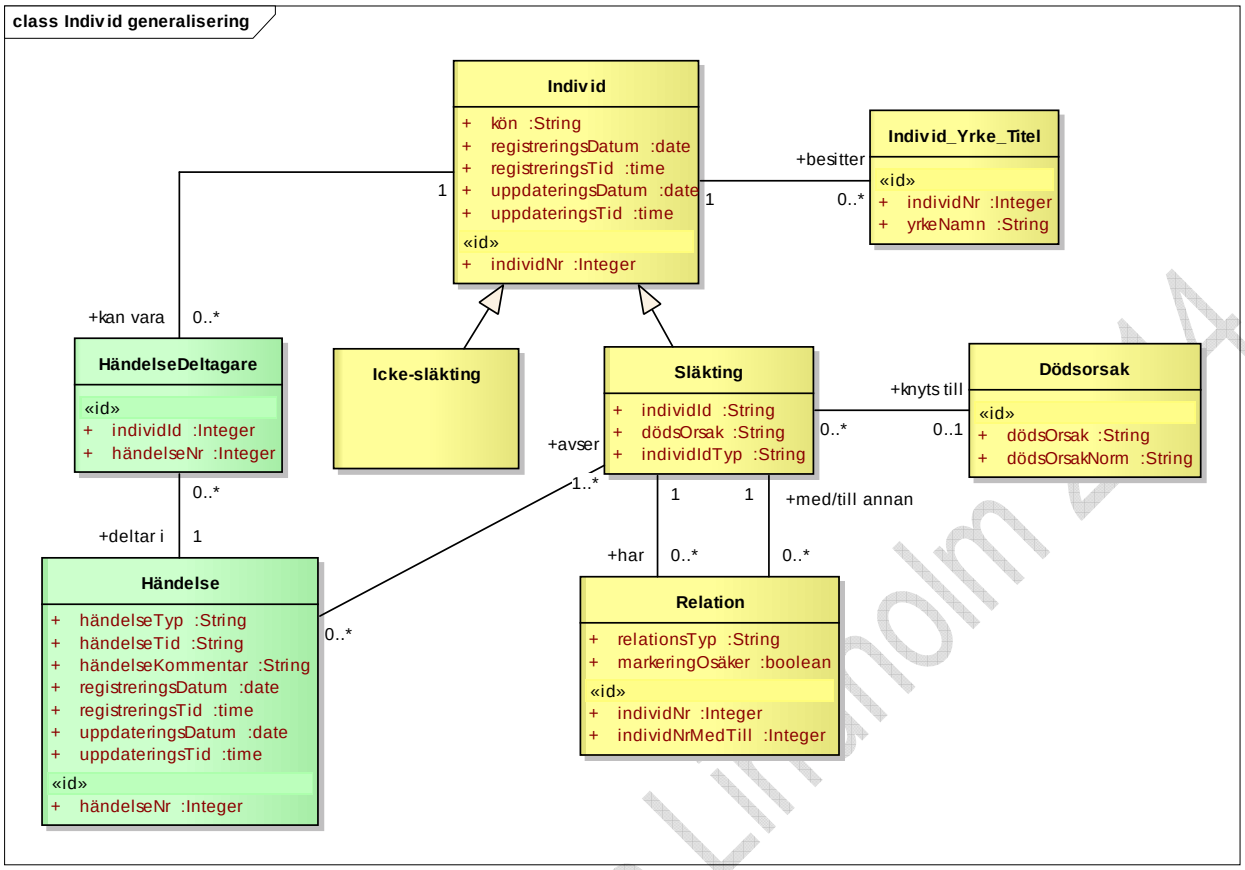
Det finns ofta en eller flera anledningar till att göra uppdelningar i mer speciella domänobjekt. En anledning kan vara att vi vill visa att de olika domänobjekten har olika typer av relationer. Ett exempel på detta syns i bilden nedan.



I exemplet finns 2 relationer. En säger att Noll till många Bilar kan dra noll till ett Släp. Det framgår alltså mycket tydligt att en Båt inte kan dra ett släp. Vidare ser vi av den andra relationen, att noll till många Släp kan transportera noll till många Fordon. Det betyder alltså att det finns släp som kan transportera vilka fordon som helst i denna modell.

En annan anledning till att dela upp ett generellt domänobjekt i flera specialiserade, är att de specialiserade har olika egenskaper. I exemplet ovan har förmodligen Bil egenskapen hjul, vilken saknas hos Båt som å andra sidan har egenskapen propeller.

Nedan har vi ett exempel från genealogin med det generella domänobjektet Individ.



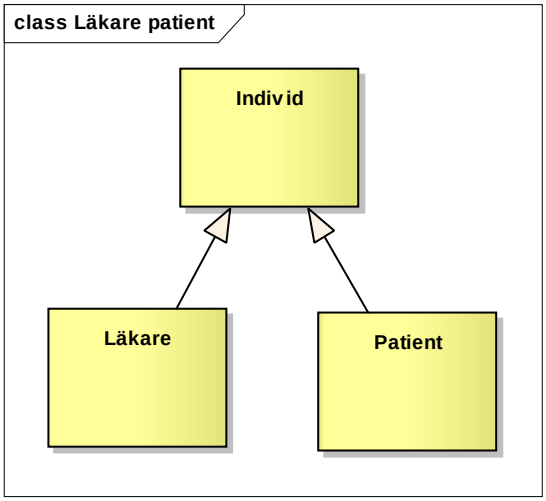
Av denna släktmodell ser vi att det generella domänobjektet har delats upp i 2 specialiserade, Släkting och Icke-släkting. Det primära i en släktmodell är att hålla reda på släktingar. Genom denna uppdelning av det generella domänobjektet Individ framgår följande;

- På domänobjektet finns ett anta egenskaper som vi håller reda på, men för Släkting håller vi reda på ytterligare några
- För Släkting håller vi via domänobjekt reda på Dödsorsak
- Vi håller via domänobjektet Relation reda på relationerna mellan Släktingar
- Vi håller reda på Händelser som avser Släktingar. Det kan vara händelser som exempelvis födelse, dop, bröllop mm
- För Händelser finns en relation till domänobjektet Händelsedeltagare som i sin tur har en relation till Individ. En händelsedeltagare kan alltså vara en släkting eller en icke-släkting. Exempelvis så skulle det vid ett dop kunna vara så att, gudmodern är en släkting medan en fadder är en icke-släkting.

Av de ovanstående exemplen kan vi alltså se att det är en fördel att använda specialisering, när det kan förtydliga och förfina beskrivningen av verksamheten. Vår modell ska vara så tydlig och precis som vi kan göra den. Det är då vi verkligen får ut största möjliga nytta av den. Vi får därför inte gömma detaljer i för generella domänobjekt, som skulle vara fallet om vi ovan exempelvis endast hade Individ utan specialisering.

När vi använder generalisering och specialisering måste det kopplas till syftet med modellen, d v s vilken verksamhet vi vill beskriva och hantera.

Till sist ska vi titta på en lite mer speciell typ av specialisering. Det sägs att det inte är bra att visa felaktigheter, men jag känner att jag måste göra ett undantag här.



I detta exempel kan vi se att Individer har delats in i Läkare och Patienter. Tanken skulle kunna var att vi vill hålla reda på olika saker om dessa. **Det finns dock en grundregel vid specialisering, och den är att det som specialiseras från det generella ska definieras unikt**. Detta betyder att om vi gör en indelning enligt ovan betyder det normalt att en individ endera är läkare eller patient. Jämför med modellen tidigare där en Invid kunde vara en Släkting eller en Icke-släkting.

I modellen ovan är det dock lite annorlunda. En läkare bör rimligen även kunna bli sjuk och alltså även vara Patient. Modellen är alltså felaktig!

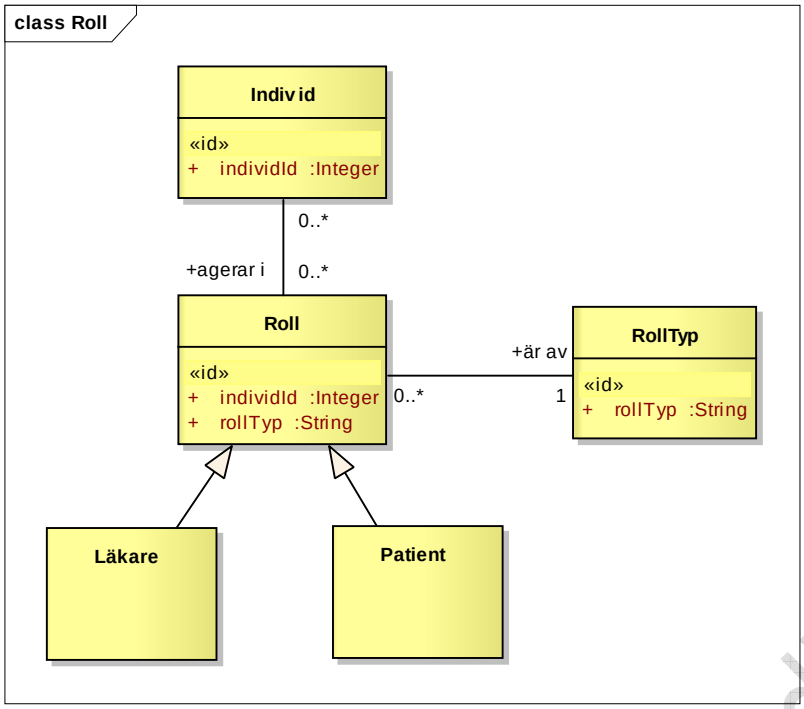
Vi kan alltså inte dela upp Individer på det sätt som visas i exemplet ovan. Specialiseringen måste ske utifrån någon egenskap hos domänobjektet. För domänobjektet Individ skulle exempelvis en specialisering kunna ske efter kön, ögonfärg eller vad som är relevant i ett visst sammanhang. Det är ingen bra ide att göra en specialisering efter egenskaper som kan förändras över tiden.

Läkare och Patient i exemplet ovan ska istället hanteras som roller, som vi som individer har i ett sammanhang. En individ kan ha många roller beroende på sammanhang, exempelvis läkare, fotbollsdomare, scoutledare, pappa, patient o s v.

Detta betyder att vi kan skapa en modell där individer kan ha olika roller, och där vi om vi vill kan typa rollerna, om de är många och vi vill ha en flexibel modell.

I exemplet nedan har domänobjektet Roll definierats. Roll har sedan specialiserats i 2 typer, Läkare och Patient. För att det tydligt ska framgå att domänobjektet Roll i detta sammanhang inte är en allmän rollbeskrivning, utan en roll för en viss individ, har jag tagit med identifierarna på domänobjekten.

Domänobjektet Roll kan ha ett antal egenskaper som är gemensamma för alla roller. Sedan kommer de olika rollerna Läkare och Patient att ha sina specifika egenskaper.



Diagram

En domänmodell kan beskrivas i ett jättediagram innehållande alla domänobjekt för verksamheten och alla relationer som finns mellan domänobjekten. Det kan vara trevligt att ha som en totalbild. Mer praktiskt är dock att ha ett antal diagram som beskriver olika delar av den totala modellen, olika utsnitt. Det gör det lätta för diskussioner olika människor emellan. Fokus sätts på ett bestämt område och inget annat distraherar.

I exemplet nedan från släktforskningsmodellen syns ett diagram som endast fokuserar på ”Plats” och hur det hanteras.