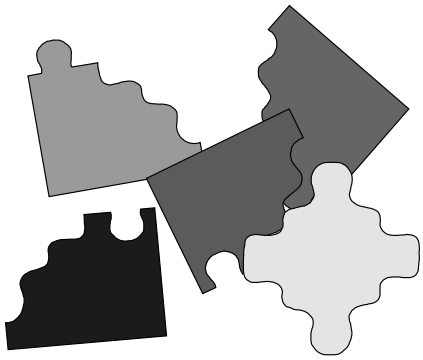
Objektorienterade mjukvarukomponenter
i datorbaserade informationssystem



Abstract

Fördelarna med att använda mjukvarukomponenter vid konstruktion av datorbaserade informationssystem är ingen nyhet. Men det är först nu som det börjar existera användbara arkitekturer och arbetssätt för skapandet av DBIS med mjukvarukomponenter. I denna rapport finns en beskrivning av vad en sådan ansats innebär samt en kartläggning och värdering av idag existerande ansatser. Värderingen visar att de flesta av dagens ansatser fokuserar på antingen infologiska eller datalogiska problem.

Benneth Christiansson

Institutionen för informatik, matematik och statistik
Högskolan i Karlstad

1997-03-22

Innehållsförteckning

1. Inledning

1.1 Bakgrund

Reuse (återanvändning) är modeordet för dagen inom mjukvaruindustrin. Dagens mjukvaruindustri är utsatt för en konstant press från sina kunder eftersom det hela tiden ställs högre krav på funktionalitet, leveranstider, kvalitet, kostnad och flexibilitet. Dessutom blir utvecklingsmiljön för mjukvara alltmer komplex med en alltmer integrerad användarmiljö och verktygsutveckling. Ett sätt att eventuellt förbättra denna situation är att tillverka standardiserade mjukvarukomponenter för att bygga dator-baserade informationssystem (DBIS). Douglas Mcllroy (Vaughn, 1990) introducerade konceptet mjukvarukomponent som byggblock vid mjukvaruproduktion redan 1968. Hans vision var en mjukvaruindustri som massproducerade standardkomponenter som kunde köpas över disk. Han pratade om en produktkatalog där varje komponent fanns beskriven, som användaren (programmeraren) kunde använda för att beställa de komponenter han behövde för att sedan foga samman dem till ett skräddarsytt DBIS.

I dagens mjukvaruindustri förekommer det återanvändning av mjukvara. Detta sker oftast internt inom en organisation med olika grader av framgång. Det ställs nya krav på en organisation för att kunna skapa återanvändning av mjukvara, det krävs nya arbetsmoment och roller i organisationen. När ex. systemering bedrivs fokuserar man på behov och sedan hittar/beställer man komponenter som uppfyller behoven, därefter ställs det nya krav på konstruktörerna, de skall kunna foga samman komponenter istället för att konstruera unika lösningar osv. Men idag finns det ingen enhetlig strategi eller standard för hur återanvändning skall ske. Så länge som det saknas en standard för hur återanvändbar mjukvara skall konstrueras kommer marknaden för återanvända mjukvarukomponenter vara begränsad. Det krävs dels att det finns ett produktionsled som kan producera komponenterna och dels ett konsumentled som kan köpa och använda komponenterna. Det enda sättet att skapa dessa två led är att producenterna vet hur och vad de skall producera och konsumenterna vet vad som finns att konsumera och hur de kan använda produkten.

”A standard approach to building and using components must be set and universally practiced if the software engineering community is to reap the benefits…of reusable software components…” (Vaughn, 1990, p.1)

Det har dock hänt en hel del sedan 1990 då Vaughn propagerade för en enhetlig standard för mjukvarukomponenter. Idag finns det flera olika

standarder för att skapa mjukvarukomponenter exempelvis CORBA, se OMG (1995).

Jag har intresserat mig för effekter på de faser/processer ett DBIS genomgår i sin livscykel då ett objektorienterat komponentsynsätt (OKS) använts/används. För att kunna göra detta måste jag beskriva vad ett OKS är samt dessutom vad ett DBIS livscykel är.

Denna rapport är en dokumentering av resultaten från fas 1 i forskningsprojektet KOMPASS (komponentbaserade informationssystem). Resultaten är baserade på litteraturstudier samt strukturerade samtal med aktörer inom mjukvaruindustrin.

1.2 Syfte

Syftet med denna rapport är att:
Beskriva vad ett objektorienterat komponentsynsätt på DBIS innebär samt *kartlägga, kortfattat redogöra för och värdera* de objektorienterade komponentansatser som finns idag.

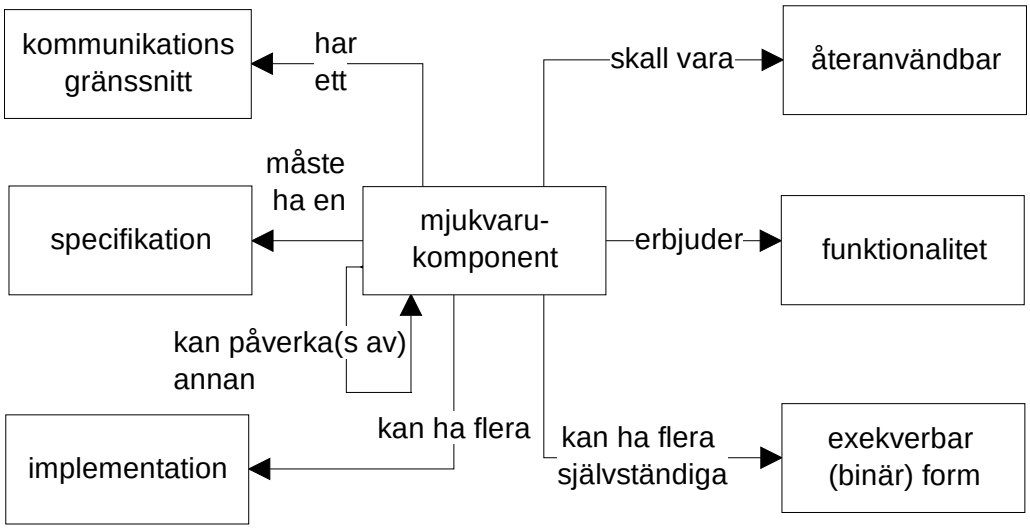
2. Vad är ett objektorienterat komponent-synsätt?

2.1 Definition

Ett OKS innebär att man betraktar ett DBIS som bestående av objektorienterade mjukvarukomponenter. Ett OKS kan användas under hela livscykeln eller under delar av den (exempelvis enbart för konstruktion men inte för analys av DBIS). Syftet med ett OKS är att ett DBIS kan bestå av köpta, återanvända eller egentillverkade mjukvarukomponenter. Man kan alltså konstruera skräddarsydda DBIS bestående av helt eller delvis återanvända mjukvarukomponenter.

2.2 Mjukvarukomponent

En mjukvarukomponent är en självständig, återanvändbar exekverbar enhet som erbjuder definierad funktionalitet via ett specificerat kommunikationsgränssnitt. En mjukvarukomponent kan påverka/påverkas av andra mjukvarukomponenter. En mjukvarukomponent måste ha en specifikation, kan ha flera implementationer samt exekverbara (binära) former. (se figur 2.1 för begreppsgraf).



Figur 2.1 Begreppsgraf över mjukvarukomponent.

En mjukvarukomponent skall finnas representerad på tre nivåer.

1. Den binära nivån (formen) innebär att komponenten är exekverbar. Detta ställer krav på det övriga systemets arkitektur (se kapitel 2.2.2).
2. Implementationsnivån är en beskrivning av hur komponenten kan implementeras och eventuellt förändras/utvecklas. Ur implementationen genereras den binära nivån. För att en komponent skall kunna föändras måste man ha tillgång till implementationsnivån (kan jämföras med källkoden till en applikation).
3. Specifikationsnivån beskriver komponentens funktionalitet, kommunikationsgränssnitt samt övriga systemmiljökrav såsom exempelvis operativsystem, plattform eller krav på mellanmjukvara för att den skall kunna identifieras och användas.

Asker (et. al. 1996) påpekar att man bör betrakta mjukvarukomponenter utifrån två synsätt, möjlighet till återanvändning och möjlighet till kombinerbarhet.

2.2.1 Möjlighet till återanvändning

”Återanvändning är att använda, och vid behov anpassa, en generaliserad komponent i många olika sammanhang” (Asker et. al. 1996, p. 21).

Till denna definition av återanvändning bör dock tilläggas att återanvändning sker för att reducera resursåtgång. Återanvändning är alltså mest intressant i de fall vi (tror att vi) minskar resursåtgången genom att skapa återanvändbara resurser.

”Consumption of already completed artifacts is a stratey for completing a task with as little effort as possible.” (Goldberg & Rubin, 1995, p.102)

I de flesta fall går det åt resurser för att skapa en generalisering som är möjlig att återanvända, detta innebär initialt en ökad resursåtgång för att förhoppningsvis leda till en reducerad resursåtgång då generaliseringen kan börja återanvändas. Dessutom så måste en återanvändbar resurs förvaltas för att förbli återanvändbar, även förvaltning förbrukar resurser. Att införa återanvändning ställer alltså nya krav på både organisation och arbetssätt.

”When we speak of reuse in software engineering we mean everything that can be reused at a later time.” (Jacobsson, 1992, p.289)

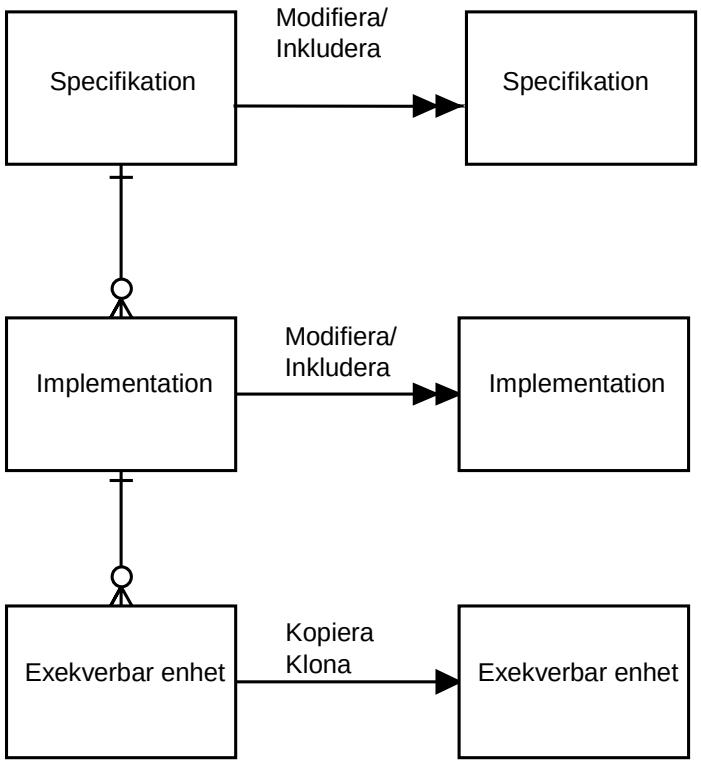
Ovanstående citat särskiljer inte användning och återanvändning, bara för att något identifieras som användbart i mer än en situation har vi inte uppnått återanvändning. Återanvändning är en planerad, medveten satsning som skall leda till att verksamheten medvetet skapar och för-

valtar återanvändbara resurser, om man ad-hocmässigt använder tidigare skapade resurser är det inte återanvändning utan enbart användning.

Då man diskuterar återanvändning inom mjukvaruindustrin kan man identifiera två grupper av återanvändare. Det finns en konsumentgrupp som vill förvärva mjukvara samt en producentgrupp som vill producera mjukvara. En verksamhet kan vara både producent och konsument. Den övergripande orsaken för återanvändning är att minska resursåtgång genom att återanvända mjukvara. En producent är intresserad av återanvändning för att kunna skapa nya produkter (mjukvara) med en lägre resursåtgång, en konsument är intresserad av återanvändning för att minimera behovet av nya produkter för att få en lägre resursåtgång, detta är ett motsatsförhållande som man måste ta hänsyn till då återanvändning skall diskuteras.

En mjukvarukomponent kan återanvändas på något av följande sätt, se figur 2.2:

- En specifikation kan återanvändas för att skapa en ny mjukvarukomponent
- En specifikation kan återanvändas för att skapa nya implementationer
- En implementation kan återanvändas för att skapa en ny en mjukvarukomponent
- En implementation kan återanvändas för att skapa nya exekverbara former
- En implementation kan återanvändas för att skapa en ny implementation av samma komponent
- En exekverbar form kan användas för att klona (kopiera) den.



En befintlig specifikation kan återanvändas för att skapa en modifiderad variant som passar de nya kraven, eller i sin helhet utan modifikation inkluderas som en del i en ny komponent.

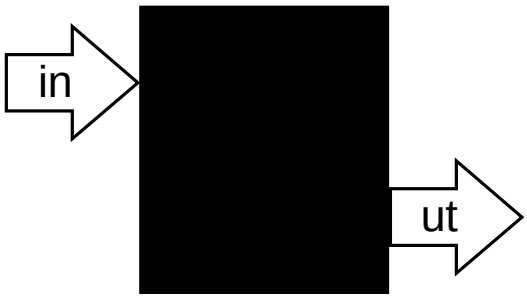
En befintlig implementation kan återanvändas för att skapa en modifierad variant som passar de nya kraven, eller i sin helhet utan modifikation inkluderas som en del i en ny komponent.

En befintlig exekverbar enhet kan återanvändas i original eller kopia.

Figur 2.2 En beskrivning av hur de olika nivåerna hos en mjukvarukomponent kan återanvändas.

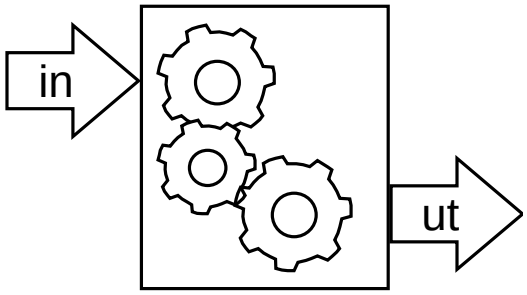
De olika sätt som en komponent kan konstrueras och distribueras på påverkar hur komponenten kan återanvändas. För att komponenten skall vara användbar som byggblock vid konstruktion av DBIS måste kommunikationsgränssnittet vara väl definierat. Om kommunikationsgränssnittet hos en komponent ändras kan detta starkt påverka stabiliteten hos ett DBIS. En komponent kan vara konstruerad och distribueras enligt någon av följande modeller:

Svart låda: Komponenten distribueras enbart i exekverbar form och förhoppningsvis även specifikationsnivå. Komponenten har ett definierat kommunikationsgränssnitt men hur komponenten ser ut internt är inte påverkbart (se figur 2.3). Kommunikationsgränssnittet och funktionaliteten finns beskriven på specifikationsnivån.



Figur 2.3 Komponenten återanvänds som svart låda, funktionaliteten är inte påverkbar.

Vit (genomskinlig) låda: Samma som svart låda men även implementationsnivån finns med. Komponentens funktionalitet är påverkbar (anpassningsbar). Figur 2.4 illustrerar en vit låda där kugghjulen representerar den inre synliga och förändringsbara funktionaliteten. Via implementationsnivån har vi tillgång till komponentens interna uppbyggnad som därigenom är påverkbart.



Figur 2.4 Komponenten återanvänds som vit låda, funktionaliteten är påverkbar.

2.2.2 Kombinerbarhet/systemarkitektur

För att kunna bygga DBIS med mjukvarukomponenter måste komponenterna ha möjlighet att påverka/påverkas av andra komponenter. Man kan även kalla det för komponenternas kommunicerbarhet. Situationen idag är att det inte finns någon enhetlig standard för hur komponenter kommunicerar med varandra. Skall man använda mjukvarukomponenter i DBIS måste aspekten kommunicerbarhet hanteras på en övergripande nivå (systemarkitekturnivå) för att möjliggöra kommunicerbarheten mellan existerande komponenter och framtida anpassningar eller nyanskaffningar av komponenter.

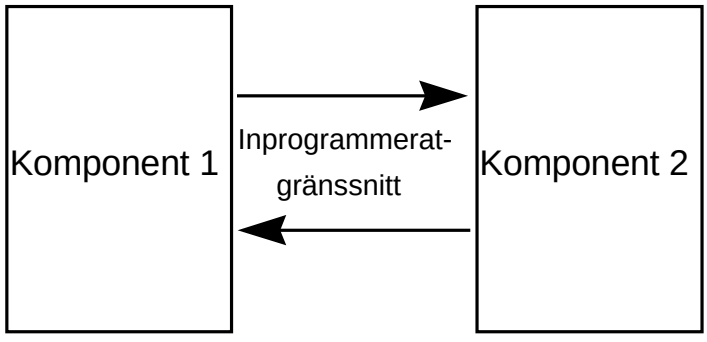
”Arkitekturen definierar ett systems komponenter och deras samverkan samt systemets struktur och väsentliga konstruktionsprinciper.” (Asker, 1995, p. 13)

Då ett DBIS skall implementeras med mjukvarukomponenter blir sambandet mellan arkitektur och standard mycket viktig. Om komponenter skall köpas in från externa leverantörer måste dessa konstrueras på ett sådant sätt att de kan implementeras ihop med redan existerande komponenter. Detta kan lösas genom att man har en öppen arkitektur, vilket innebär att ‘vem som helst’ kan konstruera komponenter som följer arkitekturen, dessutom är det viktigt att arkitekturen bygger på välkända, erkända och använda standarder.

”Exploitation of these component concepts within the software domain will be aided by open standards for specification of component syntax and semantics; the existence of a marketplace (suppliers and customers) for reusable software components; and tools for the construction of components and consequent assembly of components into solutions.” (Digre, 1996, p. 5)

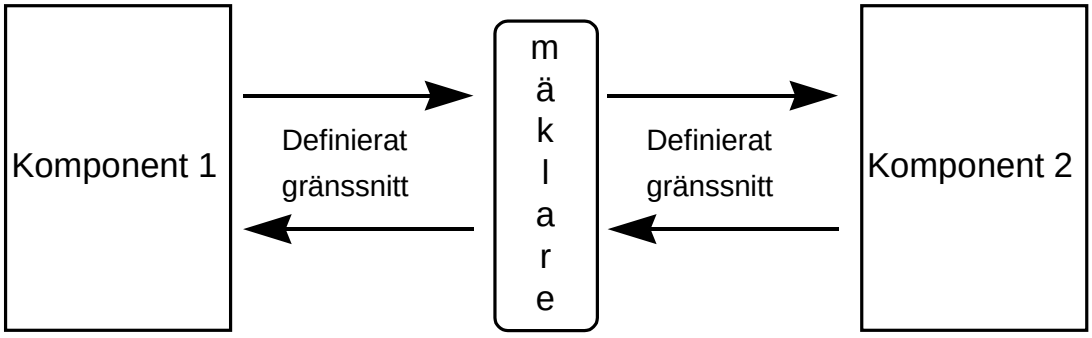
Jag har specialiserat mig på de ansatser som använder mjukvarukomponenter i kombination med koncept från objektorienteringen. De ansatser som finns idag och kan betraktas som välkända, erkända och använda bygger på en av följande strategier:

- Hård bindning, komponenterna kommunicerar med varandra genom direkta inprogrammerade anrop, se figur 2.5. För att hård bindning skall kunna åstadkommas måste man ha tillgång till komponentens implementationsnivå. Denna strategi leder till en hög grad av beroende mellan komponenter och minskar komponenternas utbytbarhet och utveckling.



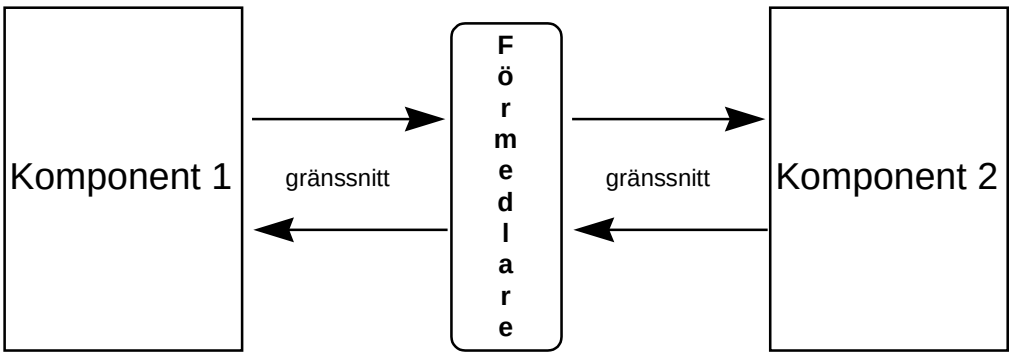
Figur 2.5. Komponenternas kommunikationsgränssnitt kodas in i komponenterna.

- Mäklare, komponenterna kommunicerar inte direkt utan all kommunikation förmedlas (mäklas) av en mellanmjukvara (object request broker), se figur 2.6 för en illustration. Denna strategi ökar komponenternas utbytbarhet och minskar graden av beroende mellan komponenter. De OKS som använder mäklare kan följa en av två idag existerande standarder CORBA (OMG group, 1995) eller COM (Microsoft, 1996a).



Figur 2.6 En mäklare med ett givet gränssnitt förmedlar kommunikationen mellan komponenterna.

- Intelligent förmedlare, mäklaren ersätts av en förmedlare med en högre grad av kunskap avseende komponenternas kommunikationsgränssnitt. En förmedlare kan känna av komponentens gränssnitt och anpassa kommunikationen efter det, se figur 2.7. Denna strategi ger en ännu lägre grad av beroende mellan komponenter, jämfört med mäklare. En förmedlare är inte beroende av ett gränssnitt utan kan anpassa sig efter komponentens (inte tvärtom som är fallet med mäklare). Denna strategi existerar i dag enbart på experimentnivå.



Figur 2.7 En förmedlare förmedlar kommunikationen mellan komponenterna med komponentens gränssnitt.

2.3 Objektorientering

Objektorientering som modelleringsteknik startade som en ren programmeringsteknik. Första skepnaden av tekniken skapades av två norrmän Ole-Johan Dahl och Kristen Nygaard genom sitt objekt-orienterade programmeringsspråk SIMULA. Detta skedde 1962 vid Norsk Regnesentral i Oslo. SIMULA skulle användas för datorbaserade simuleringar. Tanken var att låta programmet innehålla motsvarigheten till de fysiska objekt som fanns i verkligheten i form av programkod. Detta har visat sig vara en bra grund för att bygga mjukvara. Tekniken bygger på att verkligheten kan representeras med ett objektkoncept där ingående objekt har relationer till varandra. Dessa relationer motsvarar de relationer som finns mellan de koncept i verkligheten som objekten representerar. Objektorienteringen bygger på följande tre grund-principer för att beskriva/strukturera verkligheten:

1. Uppdelningen av erfarenheter i olika objekt på grund av deras attribut, genom våra erfarenheter kan vi dra slutsatser om objekt, ex. om jag sett ett träd kan jag nästa gång jag ser ett träd dra slutsatsen att det är ett träd.

2. Skillnaden mellan hela objekt och dess delar, ex. vi kan se ett träd som ett helt objekt men också dess delar, stam, grenar, löv osv.

3. Formationen och skillnaden mellan olika objektklasser, ex. vi kan se att ett träd tillhör objektklassen träd trots stora variationer mellan träd, vi kan också se att en sten inte tillhör objektklassen träd utan istället tillhör objektklassen sten osv.

För att en komponentansats skall vara objektorienterad måste ansatsen implementera (delar av) de teorier och tekniker som finns inom objekt-orienteringen. För en närmare beskrivning av objektorientering hänvisar jag till Coad & Yourdon (1991), Eriksson (1992) och Fagerström (1995).

De delar av objektorienteringen som jag anser måste vara implementerade för att en komponentansats skall vara objektorienterad är:

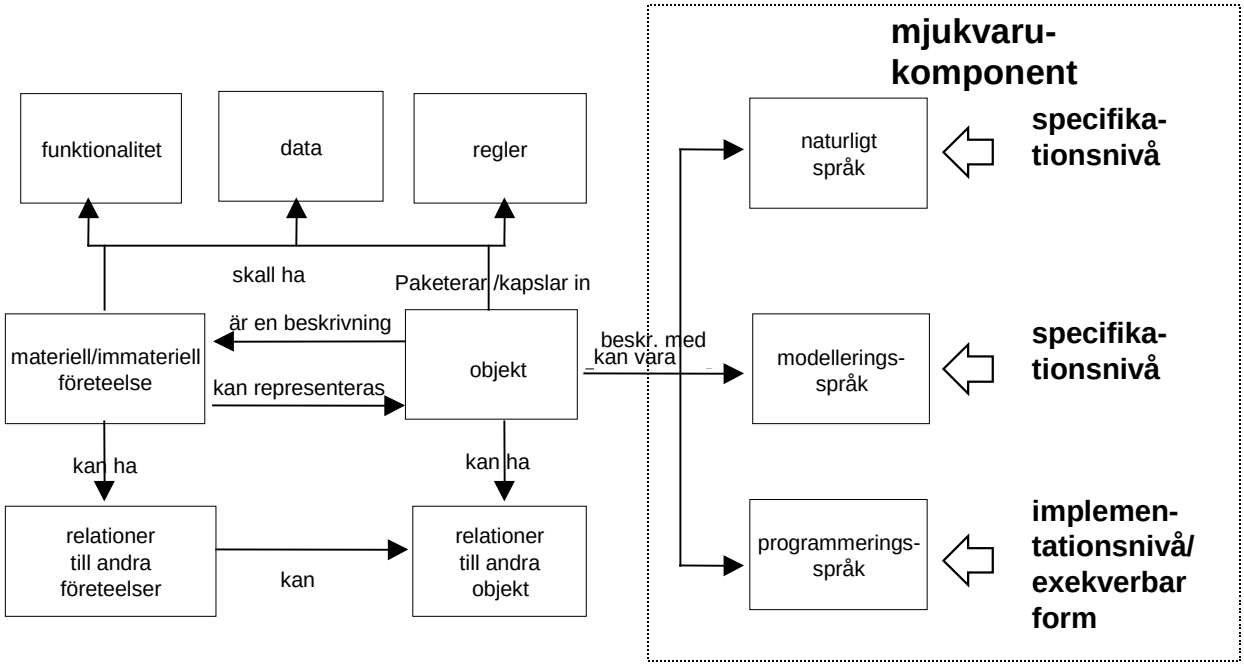
- **Inkapsling:** Det *inre beteendet* hos ett objekt är dolt för andra objekt, ett objekt ses därför som en helhet som har ett yttre beteende som andra objekt kommer i kontakt med. Objekten kommunicerar med varandra via meddelanden. Ett objekt kan inte hämta information om eller påverka det inre uppträdandet hos ett annat objekt utan måste få informationen av objektet via meddelanden. Inkapsling möjliggör att komponenter kan konstrueras så att funktionalitet kapslas in och ett yttre kommunikationsgränssnitt kan definieras.

- **Kommunikation med meddelanden:** All kommunikation mellan objekt sker i form av specificerade meddelanden. Även denna teknik möjliggör definitionen av ett kommunikationsgränssnitt mellan mjukvarukomponenter.

Inom objektorienteringen finns det fler användbara koncept som med fördel kan implementeras men de koncepten är inte av vital karaktär för komponentansatsens användbarhet.

2.3.1 Skillnaden mellan objekt och mjukvarukomponent

Ett objekt är en representation av en materiell eller immateriell företeelse. Ett objekt paketerar och kapslar in de(n) data, funktionalitet och regler som är relaterade till den företeelse som objektet representerar. Ett objekt kan representeras på flera olika beskrivningsnivåer: naturligt språk, modelleringsspråk eller programmeringsspråk. Ett objekt kan vara en mjukvarukomponent under förutsättning att objektet uppfyller den definition av mjukvarukomponent som jag redogjort för. En mjukvarukomponent kan ses som en specialiserad typ av objekt, se figur 2.8 för begreppsgraf.



Figur 2.8 Begreppsgraf över objekt samt dess relation till mjukvarukomponent.

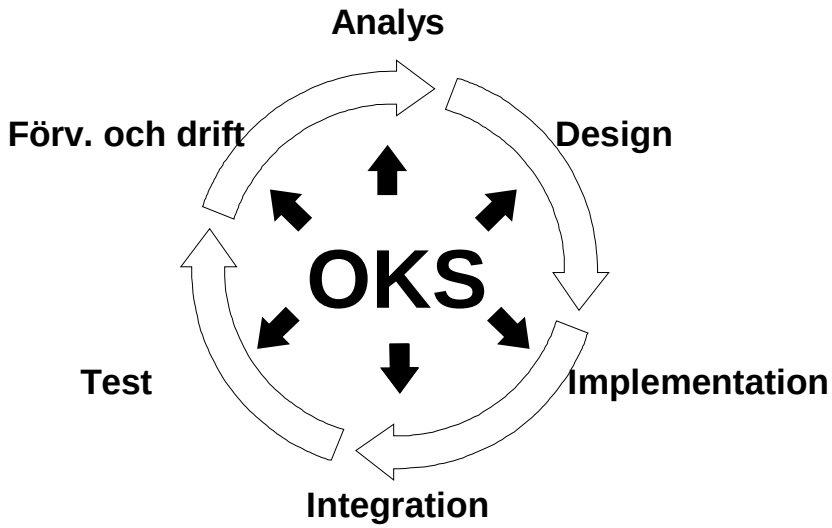
2.4 Ett OKS täckningsgrad

Det finns ett flertal olika modeller för att beskriva ett DBIS livscykel. Det är dock viktigt att vara medveten om att varje DBIS har en unik livscykel. En modell för att beskriva ett DBIS livscykel är en generalisering som inte alltid kan användas eller är lämplig för att beskriva en specifik livscykel. Ett DBIS livscykel kan beskrivas utifrån olika perspektiv beroende på vem som gör beskrivningen. Ur ett konsument-perspektiv används ett DBIS för att stödja den löpande verksamheten. Ur ett producent-perspektiv är det skapandet av DBIS som är den löpande verksamheten, detta ger ett annat perspektiv på ett DBIS livscykel i förhållande till konsument-perspektivet. Det är mycket viktigt att vara medveten om sambandet mellan producent och konsument. En konsument kan vara beroende av var en producent befinner sig i sin livscykel och tvärtom.

Generellt kan dock sägas ett DBIS livscykel på något sätt (sekventiellt, iterativt, rekursivt eller parallellt) genomgår följande faser/processer:

Analys	förståelse för problemen som DBIS skall lösa
Design	utveckla en arkitektur och detaljer för lösningen
Implementation	uttrycka designen på ett för datorn exekverbart sätt
Integration	implementationen skall anpassas till existerande miljö och system
Test	identifiera och eliminera felaktigheter i DBIS
Förvaltning och drift	det implementerade systemet skall underhållas och förändras allt efter verksamhetens behov och krav

Man kan tänka sig en situation där man använder ett OKS vid någon, några eller alla faser ett DBIS genomlöper i sin livscykel. Detta kallar jag för ett OKS täckningsgrad av ett DBIS livscykel. För att ansatsen skall ha täckning för en fas/process måste det finnas stöd och tekniker för representation och användning under fasen/processen. Genom att sedan beskriva alla de faser som ansatsen har täckning för, kan vi identifiera ansatsens täckningsgrad, se figur 2.9.

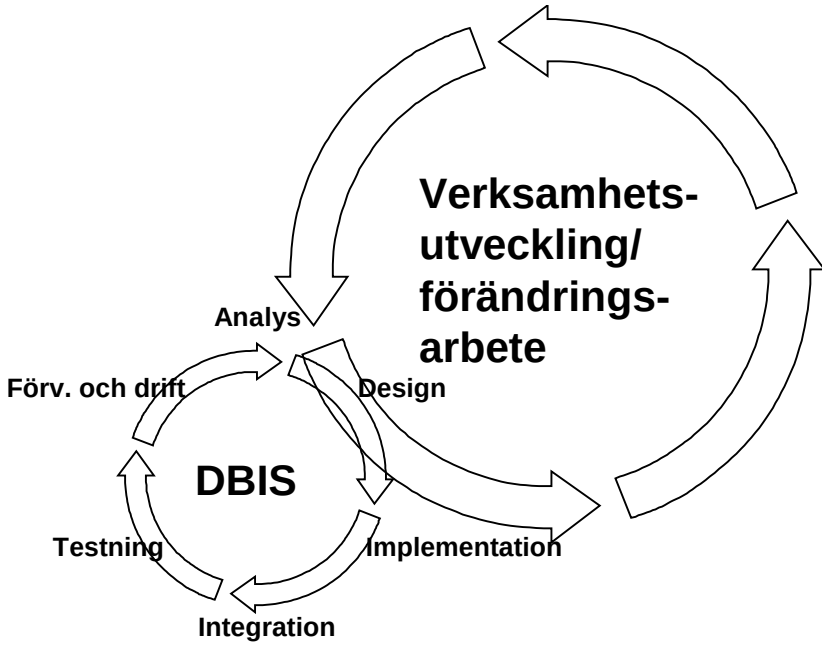


Figur 2.9 Ett OKS täckningsgrad i ett DBIS livscykel.

Om vi använder ett OKS som bara stödjer realisering så måste de övriga faserna utföras med någon annan ansats. Detta innebär att under informationssystemets livscykel måste man byta ansats (synsätt, notation, representationsteknik och arbetsformer). Varje gång man måste byta tankesätt uppstår risken för ett glapp mellan verklighet och modell samt en eventuell avtrubbning av beskrivningen. Ett byte av ansats innebär oftast att vi använder den nya ansatsen på resultatet från den föregående ansatsen. Ett exempel kan vara en funktionell utformningsfas som övergår i en objektorienterad realiseringsfas. Vi kommer då att 'objektorientera' den funktionella utformningen.

2.5 Verksamhetsutveckling

Ett DBIS existerar för att lösa ett eller flera problem åt en övergripande verksamhet. Det är fenomen i denna verksamhet som initierar skapandet av ett DBIS med dess livscykel. Det är under arbetet med att utveckla/analysera en verksamhet som behovet av ett DBIS uppstår ,se figur 2.10. Det bedrivs alltså ett utvecklings/analysarbete på verksamhetsnivå. Ett stort problem med verksamhetsutveckling och skapandet av DBIS är att de är till naturen mycket olika varandra. Detta har medfört att det finns ett 'glapp' mellan verksamhetsutveckling och skapandet av DBIS. Det blir därför mycket intressant att undersöka om ett OKS innehåller stöd även för verksamhetsutveckling och i så fall på vilket sätt.



Figur 2.10 Verksamhetsutveckling i förhållande till skapandet av ett DBIS.

2.6 Infologiskt eller datalogiskt fokus

Langefors (1993) identifierade två fundamentala delar av ett DBIS, den infologiska och den datalogiska delen.

”The ”Infological” problem of how to define the information to be made available to the information system user, and how to design data that may represent the information to the user…The ”datalogical” problem of how to organize the set of data and hardware so as to implement the information system.” (Langefors, 1993, p.148)

För att identifiera den infologiska delen av ett DBIS måste man identifiera den information som systemet skall hantera, samt beskriva hur informationen skall representeras i systemet. Den datalogiska delen innebär att identifiera och utforma den IT-infrastruktur (hårdvara-mjukvara) som krävs för att kunna skapa systemet enligt den infologiska delen. Enligt Langefors (1993) innehåller utvecklingen av ett DBIS två stora problem, att identifiera och representera information (vad?) samt organisera och implementera densamma (hur?). Se figur 2.11 för en illustration av ett DBIS två huvudproblem.

Figur 2.11 Ett DBIS kan beskrivas som bestående av två delar.

Ett heltäckande OKS bör därför hantera både den infologiska samt den datalogiska delen av ett informationssystem. Det är i vilket fall som helst mycket viktigt att vara medveten om vilken del (om nu inte ansatsen hanterar båda) som strategin hanterar. Båda delarna måste hanteras och om ansatsen bara hanterar den ena delen måste en annan ansats användas för att hantera den andra delen. Det är dessutom viktigt att identifiera hur (på vilket sätt) ansatsen hanterar delarna.

2.7 Gångbarhet

Syftet med att använda ett OKS är möjligheten att kunna återanvända, köpa och utveckla mjukvarukomponenter för att konstruera DBIS. Detta innebär att ett OKS spridning, användning och livskraft är central för dess användbarhet. Ett OKS som bara används av enstaka verksamheter kommer alltid att förbli en intern strategi som enbart kommer att möjliggöra återanvändning och inte leda till möjlighet att köpa standardkomponenter. Det krävs att det finns en volym av producenter och konsumenter som är intresserade av samma OKS (utbud-efterfrågan). Ett exempel på detta fenomen kan vara Microsofts operativsystem DOS. Vem som helst får bygga och sälja applikationer till det. Detta innebär att många använder DOS därför att det finns många applikationer till det. Då det finns många användare som är presumtiva konsumenter blir det intressant för producenter att skapa ännu fler applikationer till operativsystemet som i sin tur lockar ännu fler användare. DOS genomgick en positiv marknadsspiral. Detta fenomen är något som måste iakttas då ett OKS skall användas (Asker, 1995). Är ansatsen etablerad? Finns det en marknad? Sker det en rimlig produktion av komponenter med ansatsen? Vilken marknadspotential kommer ansatsen att ha i framtiden?

Ytterligare en viktig aspekt att ta hänsyn till är ansatsens teknikberoenden. Med teknikberoende menar jag ansatsens beroende av viss hårdvara eller mjukvara för att kunna användas. En tydligare uppdelning av teknikberoenden kan vara:

- Plattformsberoende, är ansatsen bunden till ett visst operativ-system?
- Maskinberoende, är ansatsen bunden till en viss typ av datorer?
- Mellanmjukvaruberoende, är ansatsen beroende av någon annan programvara bortsett från operativsystemet?
- Leverantörsberoende, finns det oberoende leverantörer av komponenter?

En analys av en ansats etablering i dagens och framtida mjukvaruindustri i kombination med en analys av ansatsens teknikberoenden ger ett mått på ansatsens gångbarhet.

3. Objektorienterade komponentansatser

I detta kapitel skall jag försöka kartlägga och beskriva de ansatser som är välkända, erkända och använda idag. Jag tänker försöka beskriva varje ansats utifrån de delar/koncept som jag identifierat som tillhörande ett OKS:

- Sättet att implementera mjukvarukomponenter
- Sättet att representera komponenter (beskrivningsnivåer)
- Hur ansatsen möjliggör återanvändning och anpassning av mjukvarukomponenter
- Ansatsens täckningsgrad av ett DBIS livscykel
- Hur (om) ansatsen ger stöd i verksamhetsutvecklingsarbete samt hanterar infologiska och datalogiska problem
- Ansatsens gångbarhet

3.1 VBX (Visual Basic) och ActiveX komponenter

Implementation

Visual Basics VBX är en binär standard för hur underprogram kan kommunicera med varandra. Komponenterna finns representerade i binär form och kommunikationsgränssnittet skapas genom ett permanent run-time system som alltid måste användas. Komponenterna kommunicerar via hård bindning, all kommunikation mellan komponenter måste programmeras in i komponenten. Denna standard är numera utvecklad till ActiveX, som används för att konstruera komponenter som kan användas via Internet (WWW). Run-time modulen som krävs för att köra ActiveX komponenter är Java Virtual Machine (Microsoft, 1996b, Microsoft, 1996c).

Representation

En komponent utvecklad med ActiveX eller VBX existerar på minst binär form och implementationsnivå. Det finns inget krav på en specifikationsnivå och inget stöd för hur en sådan specifikation bör se ut enligt ansatsen. Implementationsnivån existerar där komponenten skapades men behöver dock inte existera där komponenten används (jämför applikation - källkod). En leverantör kan om så önskas alltså bara leverera den binära nivån och komponenten kan sedan användas förutsatt att kunden har rätt anpassad systemarkitektur (run-time system). Om en ny komponent skall integreras i miljön måste de komponenter som skall kommunicera med denna vara antingen förberedda (redan inprogrammerade) med kommunikationsanropen eller förändras (omprog-

rammeras) så att kommunikationen blir möjlig. Detta kan endast göras på implementationsnivån.

Återanvändning/anpassning

För utvecklaren av en ActiveX-komponent är komponenten en 'genomskinlig låda' då utvecklaren har tillgång till implementationsnivån av komponenten. Om distribution enbart sker på den binära nivån kan däremot kunden bara återanvända komponenten som en 'svart låda'.

Täckningsgrad

Ansatsen ger ett direkt stöd i följande faser (se figur 3.1):

- Implementering, om ett programutvecklingsverktyg (ex Visual Basic eller Borlands Delphi) används under utformningsfasen av ett DBIS (prototyping) .
- Integration, komponenter integreras via ett givet run-time system som måste användas.

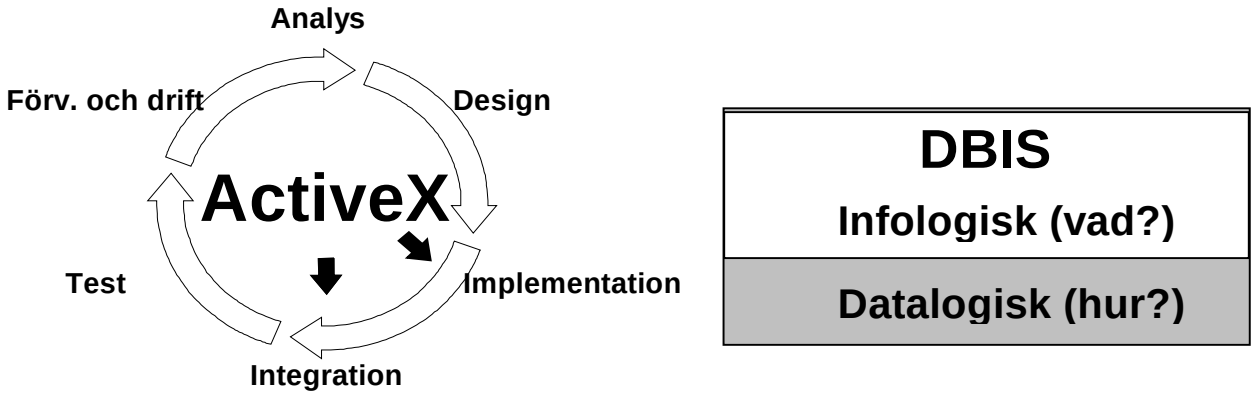
Ansatsen kan ge ett visst stöd vid analys och design, dock ej uttalat, om man använder prototyping under faserna.

Stöd i verksamhetsutvecklingsarbete samt Infologisk eller datalogisk fokus

ActiveX som OKS fokuserar i huvudsak den datalogiska delen av ett DBIS, tyngdpunkten ligger vid aspekter såsom:

- kommunikationsgränssnitt
- run-time system
- plattformsaspekter
- konstruktion av komponenters implementationsnivå samt exekverbar form

Det finns inget uttalat stöd för infologiska problem eller verksamhets-utvecklingsarbete.



Figur 3.1 ActiveX (VBX) täckningsgrad respektive problemfokus (grått indikerar fokusområde i den högra figuren).

3.2 Java applets och applications

Implementation

Java är ett objektorienterat programmeringsspråk som bygger på C++. Språket är kompilerande. Komponenterna genereras utifrån källkod som kompileras och då skapas den binära nivån av komponenten. Komponenterna kommunicerar endast via ett permanent run-time system som alltid måste användas, all kommunikation mellan komponenter måste programmeras in i komponenten (hård bindning). Run-time systemet som krävs för att köra komponenter utvecklade med Java är Java Virtual Machine, se Javasoft (1996). Det finns två typer av komponenter, applets och applications. Skillnaden är att en applet enbart existerar i en WWW-miljö och förflyttar sig till anropande maskin på begäran. Den kompilerade komponenten förflyttas från anropsmiljön till målmiljön där Java Virtual Machine exekverar den kompilerade enheten (applet) via en kombinerad Just-In-Time kompilator och en interpretator, se Kramer (1996) och Javasoft (1995) för ytterligare detaljer.

Representation

En komponent utvecklad med Java existerar på minst binär form och implementationsnivå. Det finns inget krav på en specifikationsnivå och inget stöd för hur en sådan specifikation bör se ut i strategin. Implementationsnivån existerar där komponenten skapades men behöver dock inte existera där komponenten används (jämför applikation - källkod). En leverantör kan om så önskas alltså bara leverera den binära formen och komponenten kan sedan användas förutsatt att kunden har rätt anpassad systemarkitektur (run-time system). Om en ny komponent skall integreras i miljön måste de komponenter som skall kommunicera med denna vara antingen förberedda (redan inprogrammerade) med kommuni-

kationsanropen eller förändras (omprogrammeras) så att kommunikationen blir möjlig. Detta kan endast göras på implementationsnivån.

Återanvändning/anpassning

För utvecklaren av en Java-komponent är komponenten en 'genomskinlig låda' då utvecklaren har tillgång till implementationsnivån av komponenten. Om distribution enbart sker på den binära nivån kan däremot kunden bara återanvända komponenten som en 'svart låda'.

Täckningsgrad, stöd i verksamhetsutvecklingsarbete samt infologisk eller datalogisk fokus

Ansatsen har samma fokusområde och ger samma typ av stöd som ActiveX (VBX) komponenter (se figur 3.1).

Gångbarhet

Java är en ansats med hög gångbarhet idag. Det finns ett flertal leverantörer och användare av ansatsen. Ett problem kan dock vara att Microsofts ActiveX implementerar Java komponenter men inte tvärtom. Komponenterna används till största delen vid konstruktion av websidor. Ansatsen har skapats av Sun Microsystems och är leverantörsoberoende. Det krävs dock ett run-time system (Java Virtual Machine) för att kunna använda komponenterna.. Java Virtual Machine är en mjukvara som kan existera på de flesta operativsystem och maskinplattformar, se Kramer (1996).

3.3 Ansatser som bygger på CORBA standarden

Implementation

De komponentansatser som bygger på CORBA-standarden använder en mäklarstrategi för kommunikation mellan komponenter. Komponenterna implementeras via någon form av källkod som kompileras till binär nivå via en speciell kompilator. Komponenterna kommunicerar via en mäklare som representeras av en run-time miljö (kallas även mellanmjukvara) som måste användas. Komponenterna kan kommunicera via mäklaren och har därför en lös bindning avseende kommunikation mellan komponenter.

Representation

En komponent utvecklad med en CORBA-ansats existerar på minst binär och implementationsnivå. Det finns inget krav på en specifikationsnivå och inget stöd för hur en sådan specifikation bör se ut i strategin förutom att komponenten måste följa CORBA standarden avseende kommunikation mellan komponenter. Implementationsnivån existerar där komponenten skapades men behöver dock inte existera där komponenten används (jämför applikation -källkod). En leverantör kan om så önskas alltså bara leverera den binära nivån och komponenten kan sedan användas förutsatt att kunden har rätt anpassad systemarkitektur (run-time system). Nya komponenter kan införas och användas utan att gamla komponenters binära nivå påverkas.

Återanvändning/anpassning

För utvecklaren av komponenter med CORBA-standarden är komponenten en 'genomskinlig låda' då utvecklaren har tillgång till implementationsnivån av komponenten. Om distribution enbart sker på den binära nivån kan däremot kunden bara återanvända komponenten som en 'svart låda'.

Täckningsgrad

Ansatsen ger stöd i följande faser (se figur 3.2):

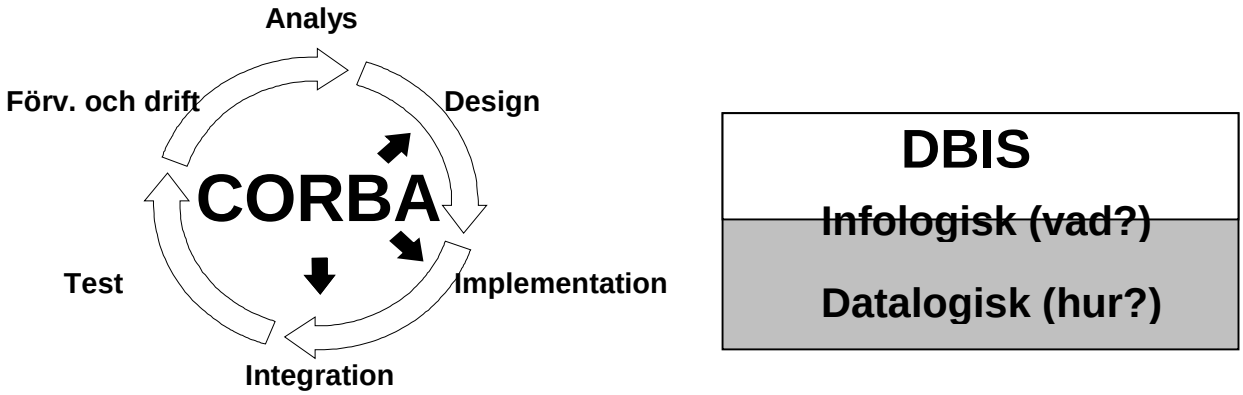
- Design, det finns beskrivningar av hur komponenterna skall se ut (kommunikationsgränssnitt) och kommunicera.
- Implementation, komponenter kan implementeras genom ett flertal utvecklingsverktyg som kan vara beroende av specifik CORBA-implementation.
- Integration, komponenter integreras via ett givet run-time system (mellanmjukvara) som måste användas, och som måste vara anpassad till CORBA-standarden.

Stöd i verksamhetsutvecklingsarbete samt infologisk eller datalogisk fokus

Ett CORBA-baserad OKS fokuserar i huvudsak den datalogiska delen av ett DBIS, tyngdpunkten ligger vid aspekter såsom:

- kommunikationsgränssnitt
- run-time system
- plattformsaspekter
- konstruktion av komponenters implementationsnivå samt exekverbar form

Det finns ett visst stöd för att hantera infologiska problem framförallt avseende en komponents kommunikationsgränssnitt på specifikationsnivå. Detta kan då även användas under verksamhetsutvecklingsarbetet.



Figur 3.2 En CORBA-implementations täckningsgrad respektive problemfokus (grått indikerar fokusområde i den högra figuren).

Gångbarhet

För att kunna diskutera gångbarhet måste varje implementation hanteras för sig. Nedan följer en lista över implementationer som följer CORBA-standarden. För varje implementation har jag försökt kartlägga dess gångbarhet.

IONA Orbix

Ansatsen är framtagen av IONA Technologies Ltd., dess största användningsområde är för närvarande framförallt ett mobiltelefonisystem som Motorola Inc. äger (Fagerström, 1995). Marknaden för komponenter byggda med ansatsen är begränsad och inte spridd i mjukvaruindustrin i allmänhet. IONA har dock utvecklat standarden numera för att även implementera komponenter utvecklade med Microsofts COM standard (se kap 3.4) och Java-komponenter via verktyget Orbix desktop (IONA Technologies Ltd. 1996). Som användare är man beroende av IONA som leverantör av objektmäklare och kompilatorer. Jag anser gångbarheten i allmänhet för IONA Orbix vara låg.

SOM/DSOM och OpenDoc

Strategin SOM/DSOM (System Object Model, Distributed SOM) är framtagen av IBM och utvecklades först för OS/2. Ansatsen i sig har inte spridit sig speciellt långt utanför IBM's egen mjukvara. Däremot går det bra för OpenDoc som använder SOM som ORB, OpenDoc har en ganska

stor spridning. OpenDoc är en ansats framtagen främst för att slutanvändaren skall kunna bygga egna DBIS med mjukvarukomponenter. Ansatsen kräver run-time system som är leverantörsberoende och komponenter kan tillverkas ur ett flertal utvecklingsverktyg. Dessutom går det att kommunicera med OLE 2.0 komponenter.

ORB plus

ORB plus är skapad av Hewlett-Packard och utvecklad för HP-UX, Sunsoft Solaris och Microsoft NT (Hewlett-Packard, 1996). Ansatsen är användbar då den följer CORBA- standarden men dess spridning är relativt liten. Det kan dock tilläggas att Hewlett-Packard har som målsättning att dominera komponentmarknaden och satsar mycket resurser för att bli en större aktör på komponentmarknaden. Komponenterna skapas utifrån C++ kod.

ObjectBroker

Digital Equipment Corporation har en egen implementering av CORBA ObjectBroker. Digital har dessutom ett samarbete med Microsoft för att även möjliggöra användning av komponenter utvecklade enligt COM (se kap 3.4).

Övriga implementationer:

- **Xshell** (Expersoft)
- **HP-DOMS** (Hyperdesk)
- **ORBeline** (PostModern Computing)
- **DOE** (SUN microsystems)
- **ObjectManager/FOS** (Fujitsu)
- **DAIS** (ICL)
- **ORBitize** (NetLinks)

3.4 COM-komponenter

Implementation

COM är Microsofts Component Model och skall uppfattas som en standard för hur mjukvarukomponenter skall kommunicera med varandra. Komponenterna implementeras via OLE 2 (Object Linking and Embedding) som är implementationen av COM-standarden. Komponenter som är utvecklade med OLE 2 behöver inget run-time system för att kunna användas. Kommunikationen hanteras via operativsystemet och det kommunikationsgränssnitt som COM definierar. Komponenterna kommunicerar via lös bindning, då all kommunikation mellan komponenter sker via operativsystemet och COM standarden (Microsoft, 1996a).

Representation

En komponent utvecklad med OLE 2 existerar på minst binär form och implementationsnivå. Det finns inget krav på en specifikationsnivå och inget stöd för hur en sådan specifikation bör se ut. Implementationsnivån existerar där komponenten skapades men behöver dock inte existera där komponenten används (jämför applikation - källkod). En leverantör kan om så önskas alltså bara leverera den binära nivån och komponenten kan sedan användas förutsatt att kunden har ett operativsystem som följer COM-standarden. Nya komponenter kan införas och användas utan att gamla komponenters binära nivå behöver påverkas.

Återanvändning/anpassning

För utvecklaren av komponenter med OLE 2 är komponenten en 'genomskinlig låda' då utvecklaren har tillgång till implementationsnivån av komponenten. Om distribution enbart sker på den binära nivån kan däremot kunden bara återanvända komponenten som en 'svart låda'.

Täckningsgrad

Strategin ger ett direkt stöd i följande faser (se figur 3.3):

- Implementering, komponenter kan konstrueras genom ett flertal applikationer och utvecklingsverktyg.
- Integration, komponenter integreras via ett givet operativsystem som följer COM- standarden och måste användas.

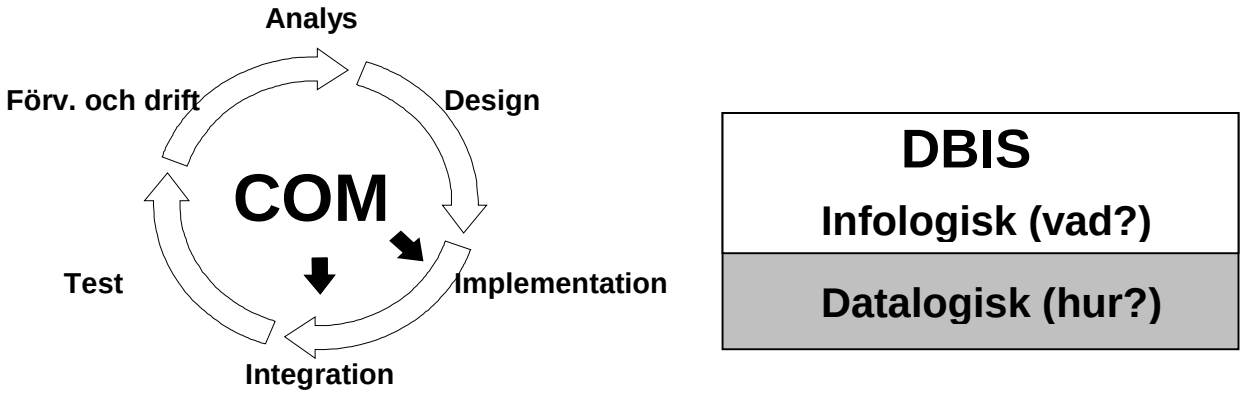
Ansatsen kan ge ett visst stöd vid analys och design, dock ej uttalat, om man använder prototyping under faserna.

Stöd i verksamhetsutvecklingsarbete samt infologisk eller datalogisk fokus

COM som OKS fokuserar i huvudsak den datalogiska delen av ett DBIS, tyngdpunkten ligger vid aspekter såsom:

- kommunikationsgränssnitt
- plattformsaspekter
- konstruktion av komponenters implementationsnivå samt exekverbar form

Det finns inget uttalat stöd för infologiska problem eller verksamhets-utveckling.



Figur 3.3 COM-komponenters täckningsgrad respektive problemfokus (grått indikerar fokusområde i den högra figuren).

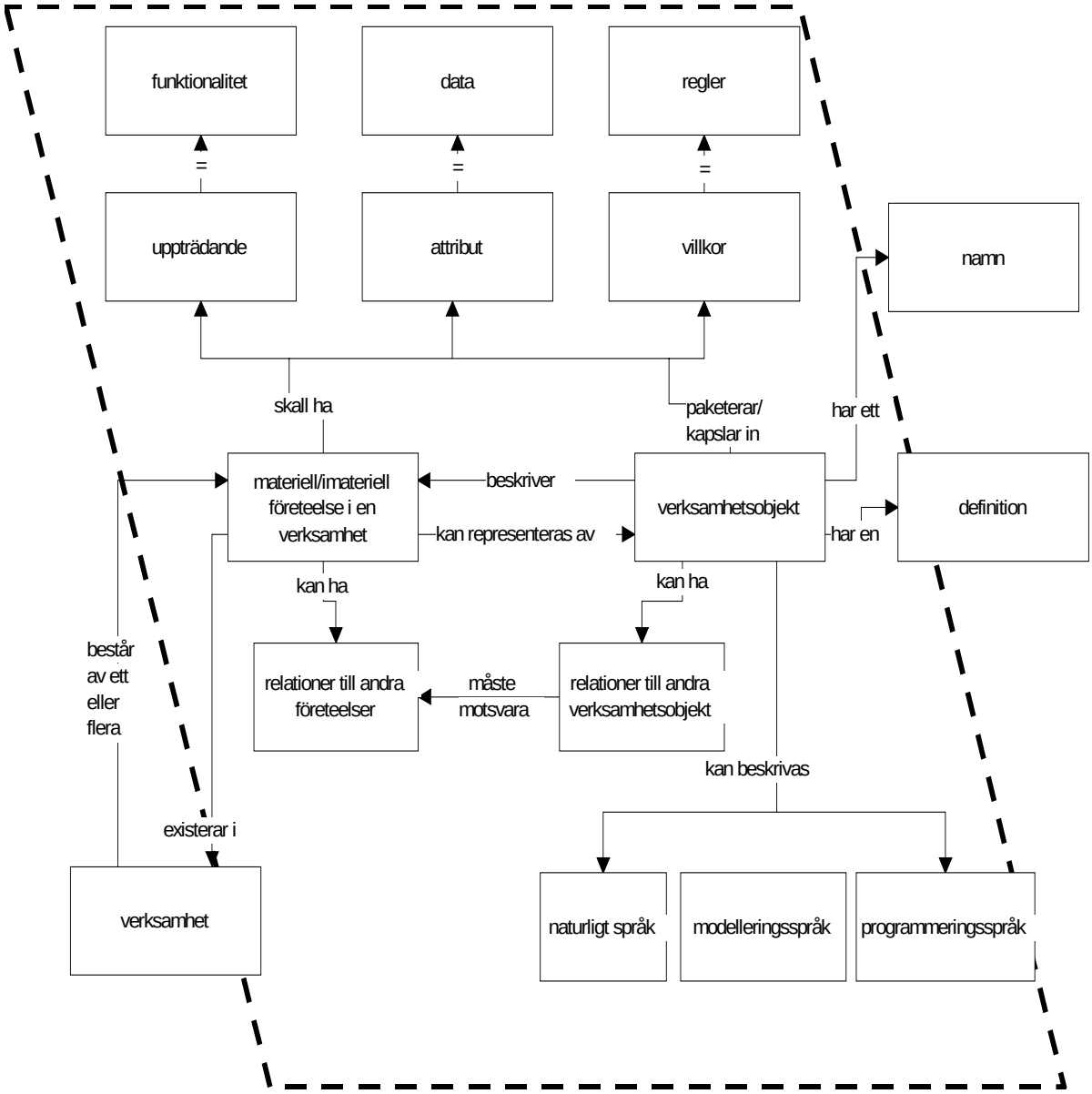
Gångbarhet

Att använda OLE 2 är en av de mest gångbara ansatserna idag. OLE 2 är framtaget av Microsoft och har en stor spridning och användning. Ansatsen är leverantörsoberoende då 'vem som helst' kan bygga en applikation som använder, används av eller genererar OLE 2 komponenter. Den kräver dock ett operativsystem som följer COM- standarden för att kunna använda komponenterna, dessa operativsystem kommer till största del från Microsoft. Ett problem med COM är att den inte är kompatibel med CORBA. Det finns dock implementationer av CORBA som kan hantera COM exempelvis OpenDoc och Object-broker.

3.5 Business Objects (Verksamhetsobjekt)

Implementation

Ansatsen specificerar inte hur mjukvarukomponenter skall implementeras utan påpekar endast att realisering skall ske med mjukvarukomponenter (ansatsen förvaltas och förädlas av OMG så ett viss fokusering mot CORBA förekommer). Ett verksamhetsobjekt är en representation av en materiell- eller immateriell företeelse som ingår i en verksamhet. Representationen skall minst innehålla namn, definition, attribut, uppträdande, villkor och relationer till andra verksamhetsobjekt. Representationen kan vara beskriven i naturligt språk, modelleringsspråk eller programmeringsspråk. Ett verksamhetsobjekt är en utveckling av objektorienteringens objekt. Skillnaden är att ett verksamhetsobjekt måste representera en företeelse ur en verksamhet, se figur 3.4 för begreppsgraf.



Figur 3.4 Begreppsgraf över verksamhetsobjekt, den streckade delen motsvarar definitionen av ett objekt.

Representation

En komponent beskriven med business objects som ansats har alltid minst en specifikationsnivå, de övriga nivåerna finns det inget stöd för. Hur komponenten skall implementeras i en systemarkitektur finns ännu inte specificerat.

Återanvändning/anpassning

En mjukvarukomponent utvecklad med verksamhetsobjekt som strategi har enbart stöd för återanvändning av specifikationsnivån av mjukvarukomponenter.

Täckningsgrad

Ansatsen ger stöd i följande faser (se figur 3.5)

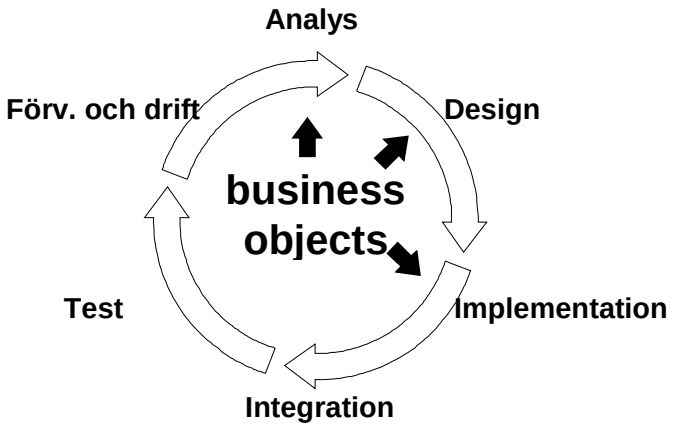
- Analys, verksamhetsobjekt ger ett tydligt och starkt stöd avseende analys och modellering, både avseende tankesätt, notation och modelleringsteknik.
- Implementation ger stöd avseende vad som skall representeras i ett DBIS samt ingående komponenters specifikationsnivå.

Stöd i verksamhetsutvecklingsarbete samt infologisk eller datalogisk fokus

Att använda business objects som OKS ger det största stödet avseende den infologiska delen dessutom ger strategin stöd i verksamhetsutvecklingsarbetet, strategin stödjer bland annat (se Casanve, 1996):

- Kommunikation, ansatsen bidrar med en terminologi och detaljeringsprinciper för att möjliggöra kommunikation mellan användare och utvecklare och skapa förståelse för verksamheten med verksamhetens eget språk.
- Modellering, genom att använda verksamhetsobjekt kan man med fördel modellera och beskriva en verksamhet genom dess affärsprocesser.

Det finns ett visst stöd för att hantera datalogisk problem genom implementering framförallt via CORBA-standarden. Enligt OMG kommer det att utvecklas ett fullständigt stöd även för datalogiska problem under 1997.



Figur 3.5 Verksamhetsobjekts täckningsgrad respektive problemfokus (grått indikerar fokusområde)

Gångbarhet

Verksamhetsobjekt som OKS har hög gångbarhet. Ansatsen är till största del en verksamhetsutvecklingsteknik. Ansatsen fokuserar i första hand infologiska problem vilket är ganska unikt för de ansatser jag identifierat och beskrivit. Ansatsen är skapad i sin första skepnad av Oliver Sims (1994) på IBM, därefter har OMG tagit över utvecklingsarbetet och håller på att komplettera med stöd även för datalogiska problem. Inom OMG har det bildats två underavdelningar som arbetar med verksamhetsobjekt, BOMSIG (Business Objects Special Interest Group) samt BODTF (Business Objects Domain Task Force) som båda arbetar med att vidareutveckla ansatsen.

Ansatser som använder verksamhetsobjekt:

EIM (Enterprise Integration Model)

EIM (Digre, 1996) är en ansats som använder verksamhetsobjekt för infologiska problem samt en mäklarstrategi (CORBA) för datalogiska problem.

OOBE (Object Oriented Business Engineering)

OOBE (Shelton, 1995b, 1995c) är en ansats som använder affärsprocesser och verksamhetsobjekt för infologiska problem samt en mäklarstrategi (CORBA) för datalogiska problem (Shelton 1995c).

BAA (Business-Application Architecture)

BAA (Casanve, 1996) är en ansats som använder verksamhetsobjekt för datalogiska problem.

”The business-application architecture does not attempt to specify the correct or best method for implementing business objects. Any combination of computer languages, 4GLs, design tools, frameworks, rule-based systems, and expert systems may be employed to implement a business object” (Casanve, 1996, p.6)

4. Slutsatser

Ett objektorienterat komponentsynsätt innebär att under hela eller delar av ett DBIS livscykel använda mjukvarukomponenter som implementerar delar av det objektorienterade tankesättet. Då ett OKS används kan ett DBIS bestå av köpta, återanvända eller egentillverkade mjukvarukomponenter.

Langefors (1993) identifierade två fundamentala delar av ett DBIS, den infologiska-(identifiera och beskriva den information som systemet skall hantera) och den datalogiska delen (beskriva hur IT-infrastrukturen för systemet skall se ut). Ett OKS kan ta hänsyn till båda dessa delar, eller fokusera på en av dem.

Det är viktigt att fokusera på vad som utlöser skapandet av ett DBIS, det föregås av någon form av verksamhetsanalys/utveckling. Behovet av ett DBIS skapas alltså då vi utvecklar en verksamhet. Idag finns det ett glapp mellan verksamhetsutvecklingsarbetet och skapandet av DBIS. Det blir därför mycket intressant att identifiera om ett OKS ger stöd för, och i så fall hur den ger stöd för, verksamhetsutvecklingsarbete.

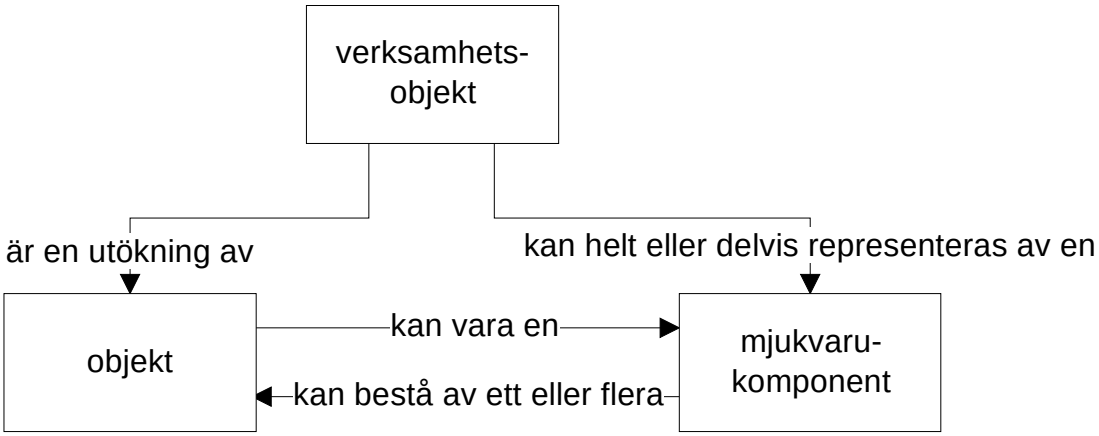
Ett OKS skall hantera infologiska och/eller datalogiska problem avseende DBIS genom mjukvarukomponenter samt ha ett objektorienterat synsätt på dessa. Ansatsen bör bygga på koncepten inkapsling och kommunikation med meddelanden från det objektorienterade synsättet. Ansatsen skall vidare ha en tydlig beskrivning av hur komponenter skall återanvändas och anpassas. För att uppnå återanvändning måste verksamheten ha en tydlig organisation för hantering av återanvändbara resurser.

En mjukvarukomponent är en självständig, återanvändbar exekverbar enhet som erbjuder definierad funktionalitet via ett specificerat kommunikationsgränssnitt. En mjukvarukomponent kan påverka/-påverkas av andra mjukvarukomponenter. En mjukvarukomponent måste ha en specifikation, kan ha flera implementationer samt exekverbara (binära) former. En mjukvarukomponent kan betraktas som en specialiserad form av objekt.

De ansatser som jag har kartlagt och beskrivit fokuserar antingen datalogiska eller infologiska problem, det finns ännu ingen ansats som hanterar både och. De ansatser som hanterar infologiska problem använder alla konceptet verksamhetsobjekt för detta, dessa ansatser är OOBE (Object Oriented Business Engineering), EIM (Enterprise Integration Model) samt BAA (Business-Application Arcitecture).

Ett verksamhetsobjekt är en representation av en materiell- eller immateriell företeelse som ingår i en verksamhet. Representationen skall minst innehålla namn, definition, attribut, uppträdande, villkor och relationer till andra verksamhetsobjekt. Representationen kan vara beskriven i naturligt språk, modelleringsspråk eller programmeringsspråk.

Relationerna mellan ett objekt, mjukvarukomponent och verksamhets-objekt kan illustreras genom en begreppsgraf, se figur 4.1.



Figur 4.1 Relationerna mellan objekt, verksamhetsobjekt och mjukvarukomponent

En ansats täckningsgrad är hur stort stöd ansatsen ger i de faser/processer ett DBIS genomgår i sin livscykel, detta är en viktig aspekt att fokusera hos ett OKS.

En annan viktig egenskap är ansatsens spridning och användning i mjukvaruindustrin samt framtida livskraft. Är ansatsen leverantörs-, maskin-, operativsystems- eller mjukvaruberoende. Dessa egenskaper tillsammans kallar jag ansatsens gångbarhet. De mest gångbara ansatserna idag är de som bygger på Microsofts ActiveX eller COM teknik. En annan viktig egenskap som utmärker de mest gångbara ansatserna är att de har en helt öppen systemarkitektur (‘vem som helst’ kan konstruera komponenter med ansatsen). De fokuserar dock enbart den datalogiska delen av ett DBIS.

5. Omnämnanden

Denna rapport är framtagen inom forskningsprojektet KOMPASS (komponentbaserade informationsystem), Högskolan i Karlstad,

institutionen för informatik, matematik och statistik. Följande personer starkt bidragit till denna rapport med synpunkter, ideér och kritik: seniorkonsult Gösta Steneskog (Handelshögskolan i Stockholm, Institut V), adj. prof. Anders G. Nilsson (Handelshögskolan i Stockholm, Linköpings Universitet), doktorand Marie-Therese Lundmark (Högskolan i Karlstad), prof Bo Sundgren (Handelshögskolan i Stockholm, SCB), seniorkonsult Lars Wiktorin (ITplan) samt seniorkonsult Claes-Göran Lindström (ITplan).

6. Referenser

Andersen E S. (1994*) Systemutveckling - principer, metoder och tekniker.* Studentlitteratur, Lund.

Asker B. (1994*) Att bygga produkter med köpt programvara. Sveriges verkstadsindustrier*, Stockholm.

Asker B. (1995) *Arkitektur och systembygge för programvara . Sveriges verkstadsindustrier*, Stockholm.

Asker B, Nilsson M, Söderström P & Wiktorin L. (1996) *Återanvändning i verkligheten.* Studentlitteratur, Lund.

Basili V R. (1994) *Facts and Myths affecting Software Reuse.* ICSE. 16: 269.

Business Object Management Special Interest Group. (1995) *OMG Business Application Architecture.* OMG (www.omg.org),

Casanve C. (1995) *Business-Object Architectures and Standards.* Data Access Coorporation, Miami, USA.

Christiansson B. (1996) *Kunskapsprojektering -Effekter av en objektorienterad komponentstrategi vid utveckling och förvaltning av datorbaserade informationssystem.,* Högskolan i Karlstad

Coad P & Yourdon E. (1991) *Object-Oriented Analysis.* Yourdon Press, USA.

Digital Equipment Corporation. (1996) *Objectbroker.*

Digre T. (1995*) Business Application Components.* Texas Instruments, Inc, Plano, USA.

Digre T & Business Object Architecture team. (1996) *Business Object Facility.* Texas Instruments Incorporated, USA.

Eriksson H-E. (1994) *Objektorienterad programutveckling med C++.* Studentlitteratur, Lund.

Fagerström J. (1995) *Objektorienterad analys och design -en andra generationens metod*. Studentlitteratur, Lund.

Foody M. (1995) *Connecting today's islands of objects, Cross platform solutions -OpenDoc Parts Framework*. Discovery publishing Group, USA.

Garlan D, Allen R & Ockerbloom J. (1995) *Architectural Mismatch or Why it's hard to build systems out of existing parts.* ICSE. 17: 179-185.

Goldberg A & Rubin K S. (1995) *Succeeding with Objects, Decision Frameworks for Project Management.* Addison-Wesley Publishing Company, UK.

Graham I. (1994) *Migrating to Object Technology.* Addison-Wesley Publishing Company, Inc., UK.

Griss M L. (1994) *Software Reuse Experience at Hewlett-Packard.* ICSE. 16: 270.

Grotehen T & Schwarb R. (1995) *Implementing Business Objects: CORBA interfaces for legacy systems.* University of Zurich, Switzerland.

Hertha W F, Bennett J E, Post F J & Page I M. (1995) *An Architecture Framework: From Business Strategies to Implementation.* Canadian Imperial Bank of Commerce, Toronto, Canada.

Hung K. (1996) *A dynamic Business Object Architecture in an Iterative Life-cycle Environment for Information System Development.* www.scism.sbu.ac.uk/cios/hungks

IONA Technologies Ltd. (1996) ORBIX *Desktop for Windows*

Jacobson I. (1992*) Object-oriented Software Engineering a Use Case Driven Approach.* Addison Wesley Publishing Company, UK.

Kramer D. (1996) *The Java platform*. JavaSoft, Sun Microsystems, Inc., USA.

Kramer J. (1994) *Distributed software engineering.* ICSE. 16: 253-263.

Langefors, B. (1993) *Essays on Infology. Department of Information Systems*, University of Göteborg

Lindvall M. (1994) *A Study of Traceability in Object-Oriented Systems Development*. Linköpings universitet

Lotus Notes: *Lotus Components: White Paper. OLE Technology and Lotus Components*. http://components.lotus.com

Maciaszek L E, Getta J R & Bosdriesz J. (1996) *Restraining Complexity of Object Oriented System Development the "AD-HOC" Approach*. ISD proceedings. 5: 425-435.

Maring B. (1996) *Object-Oiented development of Large Applications*. IEEE Software. 33-40.

Mattison R & Sipolt M J. (1994) *The Object-Oriented Enterprice Making Corporate Information Systems Work*. MCGraw Hill, Inc., UK.

Meyer B. (1988) *Object-oriented Software Construction*. Prentice Hall, UK.

Microsoft. (1996a) *The Microsoft Object Technology Strategy: Component Software*. www.microsoft.com

Microsoft (1996b). *What is ActiveX?*, www.microsoft.com

Microsoft (1996c). *Java and ActiveX*, www.microsoft.com

OMG Group (1995*). The Common Object Request Broker: Architecture and Specification Revision 2.0*, www.omg.org

OpenDoc for Macintosh. *An Overview for Developers*.

Parnas D L. (1994) *Software aging*. ICSE. 16: 279-287.

Ramackers G & Clegg D. (1995) *Object Business Modelling, requirements and approach*. Oracle Corporation, UK.

Rogal D. (1995) *Cross Platform Solutions -OpenDoc Parts Framework, The New Cross-Platform development Alternative*. Cross Platform Solutions: Discovery Publishing Group, USA.

Rosenbaum S & du Castel B. (1995) *Managing Software Reuse -- An Experience Report*. ICSE. 17: 105-111.

Schmidt D C & Vinoski S. (1995) *Object Interconnections Comparing Alternative Client-side Distributed Programming Techniques* (Column 3). SIGS C++ Report Magazine.

Shelton R E. (1994a) November, *Business Objects What is a Business Object?*, Data Management Review, www.openeng.com.

Shelton R E. (1994b) December, *Business Objects BPR with Objects*, Data Management Review, www.openeng.com.

Shelton, Robert E. (1995a) February, *Business Objects Business Object Management,* Data Management Review, www.openeng.com

Shelton, Robert E. (1995b) March, *Business Objects, OOBE Framework & Reference Model,* Data Management Review, www.openeng.com

Shelton, Robert E. (1995c) April, *Business Objects, OOBE vs. OO A&D Methods,* Data Management Review , www.openeng.com

Shelton, Robert E. (1995d) July, *Business Objects, Data Warehouse,* Data Management Review, www.openeng.com

Shelton R E. (1996) *Business Objects.* Open engineering Inc. www.openeng.com,

Sims O. (1994) *Business Objects Delivering Cooperative Objects for Client-Server.* Mcgraw-Hill Book Company, UK.

Steel J. (1996a) *Component Technology Part I.* International Data Corporation

Steel J. (1996b) *Component Technology Part II.* International Data Corporation

Stewart S L & St. Pierre J A. (1995) *Experiences with a Manufacturing Framework.* U.S Department of commerce, USA.

Sun Microsystems Inc. (1995) *The Java Language: An Overview*. USA.

Sun Microsystems Inc. (1996) *Java(tm) Beans: A Component Architecture for Java.* USA.

Sutherland J. (1996) *The Object Technology architecture: Business Objects for Corporate Information Systems*. OOPSLA'95 Workshop on Business Object Design and Implementation, Springer-Verlag, Berlin.

Taylor R N, Medvidovic N & Anderson K M. (1995) *A Component- and Message Based Architectural Style for GUI Software.* ICSE. 17: 295-304.

Texas instruments & Microsoft. (1995) *Re-Engineering Application Development.* USA.

Torres J, Troyano J A & Toro M. (1994) *An Object Oriented Technique for Systems Specification.* ISD proceedings. 4: 538-547.

Tracz W. (1994) *Software Reuse Myths Revisited*. ICSE. 16: 271-272.

Vaughn T. (1990) *Issues in Standards for Reusable Software Components.* http://ruff.cs.umbc.edu,

Wasmund M. (1994) *Reuse Facts and Myths*. ICSE. 16: 273.

Zweben S H, Edwards S H Weide B W & Hollingsworth J E. (1995) *The Effects of Layering and Encapsulation on Software Development Cost and Quality.* IEEE transactions on SOFTWARE ENGINEERING. 21: num. 3 200-207.