TRIAD Newsletter K 31

Repository
State-of-the-art

Stig Berild

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

Innehåll

1 Inledning

1.1 Kort historia

Begreppet "repository" har en relativt kort men intensiv historia. Det blev ett "innebegrepp" (buzzword) att svänga sig med i samband med att kraven på effektiv utveckling och underhåll av system alltmer kom i fokus. Högkvalitativ information om den egna verksamheten och dess informationssystem skulle ses som en resurs i sig, ja på sikt som ett bedömt överlevnadsvillkor i en alltmer konkurrensutsatt och internationaliserad värld. Repository som företeelse, eller snarare som idé, kom som en befriande fågel fenix, för alla att samlas kring. I repositoryt skulle all viktig beskrivningsinformation läggas på ett för verksamheten som helhet samordnat sätt. Produkter började utvecklas. Många artiklar blev skrivna om detta i datamedia. Repository kom att appellera till något positivt och viktigt. Följden blev att alla som kunde åberopa att deras produkter kunde lagra någon typ av beskrivningsinformation, må vara Case-verktyg, DBMS, dokumenthanterare, datakataloger, CAD-program, ... valde att konstatera att de jobbade med ett repository. Standardiseringsaktiviteter sköt fart för att lägga sin ordnande, styrande hand över det hela.

Produkterna blev klara och började användas, dock inte alls i den omfattning som förväntats. De som vågade och hade råd kom sällan att använda sina repositorier på det integrerade och fullständiga sätt som visionen hade målat upp. Produkterna svarade dessutom i allmänhet inte heller mot ställda förväntningar. Verksamhetsintegrerad information betydde också med osviklig automatisk hantering i stordatorbaserade produkter.

Inte heller gick Case-verktygen, de som skulle föda repositoryt med uppgifter, någon guldkantad tid till mötes. De högt ställda förväntningarna infriades ofta inte, de förföriska lockropen i produktprospekten till trots.

Det kanske största bekymret visade sig vara att användarna hade svårt att hitta information av värde att stoppa in i repositoryt, framför allt information som var avsedd att på något sett samordnas mellan olika intressenter i verksamheten. Även om man kunde stoppa in uppgifter fanns sällan någon som fann det mödan värt att hämta fram dem. Inte heller fanns utvecklad hanteringsmetodik, än mindre någon naturlig förankring i eller samordning med de vanliga, etablerade rutinerna och ansvarsförhållandena i den existerande verksamheten.

Med andra ord behoven och bekymren fanns, men varken kunder, produkter eller processen som sådan visade sig vara mogna.

Artiklar sågs allmer sällan, deltagarantalet på konferenser krympte.

Självrannsakan gav vid handen att de tidigare styrande målsättningarna var orealistiska. Den totala samordningen var varken genomförbar eller ens efter-

strävansvärd. Den information som skulle hanteras var för differentierad, den berörde i ett mer ambitiöst perspektiv i stort sett allt och alla i en verksamhet. Beskrivningsinformation genererades på många platser, för många olika syften, under många olika förutsättningar, med hjälp av många olika typer av datorstöd som befann sig på olika så kallade tekniska plattformar (operativsystem, m m).

Behoven av rimlig samordning var det däremot ingen som egentligen tvekade om. Krav på ingenjörsmässighet, på återanvändning av informationskomponenter växte sig hela tiden allt starkare. Kanske kunde man på sikt ställa allt större krav på samordning. För stunden var det viktigt att börja i någon ände, d v s med mer begränsade målsättningar där man kände sig vara på stabil mark och på rimliga grunder och med rimlig trovärdighet i förväg kunde bedöma nyttan. Dessutom vann man automatiskt värdefull erfarenhet inför en repositorybaserad framtid.

Objektorientering, client/server-arkitekturer, nya principer för samverkan mellan fristående system gav en pånyttfödelse och nytt hopp.

Produkterna började närmast undantagslöst att anpassas till en client/server-miljö. Standardiseringsaktiviteter började få mer genomslag eller åtminstone uppmärksamhet. Kunskap och vilja att hantera beskrivningsinformation växte återigen.

Dämed är vi framme i nutid.

Nå, hur många decennier omfattar denna kompakta historiebeskrivning? Ett halvt decennium, ungefär. Visst fanns det datakataloger tidigare, ansatser till produkter likaså, men full fart blev det först (i och med att IBM "mutade in" området med sitt AD/Cycle-koncept) i slutet på 1980-talet. Det vi nu börjar se växa fram är en andra generationens system och användare. De storvulna visionerna har ersatts med mer begränsade ambitioner. Vi kommer säkerligen att genomlöpa ett antal ytterligare generationer innan vi med fog kan säga att vi med kvalitet och konsekvens hanterar information som en reell resurs och nyttjar den effektivt för såväl utveckling, drift som underhåll av såväl verksamheten som dess stödjande system.

1.2 Rapportens disposition

Föreliggande rapport har till syfte att belysa några av frågeställningarna kring begreppet repository. Bland dessa hör:

Vad brukar normalt innefattas i begreppet? Vilka är de typiska egenskaperna? Vilka användningsområden? Förhållande till andra system/verktyg? Finns oklarheter och frågetecken?

Rapporten bygger på material från tidskriftsartiklar, konferenser och ett par böcker. Som inom de flesta nya områden förekommer mycket tyckande, baserad på liten erfarenhet. Några etablerade definitioner, avgränsningar et c existerar inte. Ofta menar man olika saker med samma begrepp eller samma sak med olika

begrepp. Repositoryområdet intar därvidlag ingen särställning. Inte heller denna rapport. Synpunkter och åsikter är antingen författarens egna eller hämtade från läst litteratur.

Avsnitt 2 beskriver den problemsituation som varit upprinnelsen till framväxten av repository som idé. Den kanske viktigaste repository-egenskapen ligger i dess syfte, nämligen att samordna och integrera information. Denna samordning kan eftersträvas med olika grad av ambition. Olika ambitionsnivåer diskuteras i avsnitt 3 med referens till de olika faser som idéutvecklingen hunnit genomlöpa. Avsnittet indikerar indirekt avsikter med och krav på ett repository. Avsnitt 4 innehåller en mer explicit och renodlad diskussion på samma tema. Den som har för avsikt att beträda repositoryområdets förgyllda men minst sagt gropiga mark ges några råd i avsnitt 5. Standardisering pågår inom området under beteckningen IRDS (Information Resource Dictionary System). Vad som hittills åstadkommits berörs i avsnitt 6. Några synpunkter och frågetecken levereras i avsnitt 7 varefter avsnitt 8 avrundar det hela.

2 Problem

Detta avsnitt är relativt utförligt eftersom problemen indirekt även motiverar och formulerar behov och målsättningar för dess lösning, d v s för de egenskaper som repository-begreppet representerar.

2.1 Systemutveckling

Trenden inom systemutvecklingsområdet kännetecknas bland annat av

- större projekt – många inblandade människor med olika kvalifikationer
- komplexare system
- friare, mer heltäckande systemutvecklingsmetoder
- ökade kvalitetskrav på systemen
- krav på utbyggbara system
- krav på att systemen ska vara förståeliga och överblickbara för analys och kontroll
- krav på underhållsvänliga system.

Samtidigt konstateras allmänt att

- systemutveckling är dyrt, behöver effektiviseras
- systemutveckling tar alltför lång tid
- systemen sällan riktigt väl svarar upp mot behoven
- återanvändning av kunskap, gamla specifikationer och kod är näst intill obefintlig.

Motverkande åtgärder kan exempelvis vara att

- precisera metoder och modeller
- ersätta synen på utvecklingsprocessen som individuellt hantverk med inriktning mot gruppbaserad ingenjörsmässighet
- kräva kvalitativt godtagbar dokumentation av både beslut, problem och lösningar
- använda datorbaserade hjälpmedel typ Case-verktyg.

Trots detta kvarstår en hel del bekymmer:

- Inom en organisation eller verksamhet används många alternativa metoder, modeller och verktyg.
- Case-verktygen har olika sätt att se på och hantera information inom de olika faserna av ett systems livscykel.
- Case-verktygen stödjer ofta bara vissa faser, de behöver alltså samarbeta.
- Intresset för de tidiga faserna är ofta synnerligen måttligt. De anses flummiga eller onödiga. Duktiga programmerare anser sig kunna utveckla tillämpningar mer eller mindre direkt bara de får lite hum om vad behovet är. Den egna kreativiteten anses fylla eventuella luckor. Ändå talar statistiken sitt klara språk. De allra flesta felen kan upptäckas i de tidiga faserna om dessa genomförs. Där orsakar de bara en bråkdel av de konsekvenser och kostnader som en upptäckt under implementering eller vid test ger upphov till.

- Komplexiteten växer snabbare än verksamheten har möjlighet att lära sig begripa, utnyttja och hantera.
- Arvet i form av alla system i produktion kräver alla tillgängliga resurser för underhåll och anpassningar. Bara i USA lär det finnas mer än 70 miljarder rader använd Cobol-kod representerande en uppskattad investering på ca 350 miljarder dollar. Vem orkar då tänka på spännande, avancerade datorstöd för nyutveckling, på datorstöd som ännu inte bevisat sig hamna på ett otvetydigt plus vid en kostnads/intäktsanalys?

Den som ändå gör ett försök kan mycket väl hamna i följande förtvivlade, antagligen inte osannolika, senario:

- Företaget har inga verktyg och ingen fastlagd systemutvecklingmetod (som används konsekvent).
- 90% av energin går åt till underhåll och akuta problem.
- Hoppet ställs till Case-verktyg, som köps. Användbar utvecklingsmetod hoppas man så att säga få på köpet.
- På grund av arbetsbelastning enligt ovan finns ingen tid att lära sig verk-tyget, i alla fall inte ordentligt.
- Någon avsätts att testa och allmänt "ansvara" för verktyget. Antagligen är det samma person som nyss argumenterat för ett inköp efter att ha gått en fri introduktionskurs.
- Ingen i ansvarig ställning orkar/vågar ta strategiskt införandebeslut. Alternativt fattas ett sådant i desperation efter effektiv övertalning av förespråkaren (som därvid förväntar sig få en guru-roll, den alla frågar, den som fixar). Alternativt testas verktyget i ett pilotprojekt.
- Verkligheten kommer som en kalldusch. Det som alla innerst inne visste kommer i öppen dager. Verktyget löser inte i sig problemen, det är bara ett stöd. Användning kräver bred utbildning i verktyg och metoder, innebär stringens och anpassning av arbetet till verktygets metod (vem vill skrota sina inarbetade rutiner?). Förespråkar verktyget ingen direkt metod inträder istället den stora osäkerheten. Verktyget kanske inte motsvarar alla vackra löften. Som förstaprojekt tar dessutom saker och ting lite längre tid, något som inte kalkylerats in i tidplanen. Snart inträder tids/resursbrist, tillika brist på intresse. Resultatet blir ett misch-masch utan egentligt värde.
- Utvärderingen blir den väntade. Många "vad-var-det-jag-sa" personer dyker plötsligt upp.
- Underhållskontraktet sägs upp och allt återgår till det normala.
- Det tar år innan någon vågar ta upp motsvarande förslag igen.

Nåväl, Case- och SEE-verktyg används i realiteten för diverse ändamål inom systemutveckling, inte minst i större företag. (SEE=Software Engineering Environment.) Verktygen blir allt bättre, förmågan att använda dem likaså. Låt oss titta lite på Case-verktygens möjligheter och begränsningar i nästa avsnitt. Motiveringen till genomgången är att en del av de redovisade begränsningarna motiverar just behovet av ett repository. Andra begränsningar och bekymmer är snarare av en karaktär som istället accentueras om repositorystöd införs som komplement till Case-verktyg innan bekymren avlägsnas.

2.2 Case-verktyg

2.2.1 Olika typer

Case-verktyg brukar grovt sett indelas under I-Case respektive C-Case.

I-Case (Integrated Case)

Dessa kännetecknas av att de
- täcker hela livscykeln
- är verksamhetsorienterade
- ofta ger processtyrningstöd.

C-Case (Component Case)

Dessa kännetecknas av att de täcker en specifik "nisch".

Andra indelningar

Upper-CASE (inriktar sig mot analys- och designfaserna)
Lower-CASE (inriktar sig mot realiserings-, test- och förvaltningsfaserna)

eller

Nyutveckling
Reverse engineering

Som synes har begreppet Case en mycket lös och vid uttolkning! Det lär dessutom finnas flera hundra Case-leverantörer!

2.2.2 Bekymmer vid Case-användning

Litteraturen påtalar olika typer av bekymmer kopplade till Case-användning. Ett antal av dem redovisas nedan under tre rubriker, dock utan någon som helst rangordning.

(Det finns givetvis även glädjeämnen, men eftersom dessa, så att säga definitionsmässigt, inte motiverar komplettering med repository-funktionalitet berörs de inte i denna uppställning.)

Bekymmer av allmän karaktär

- Många verktyg. Svårt att välja rätt för aktuellt behov.
- Ingen standardisering. Alltså ingen stabilitet.
- Många systemutvecklingsmetoder finns. Vissa verktyg har metodanpassning, andra inte. Några påstår sig stödja flera.
- Få använder Case-verktyg regelmässigt. Hur många inköpta verktyg ligger inte och "dammar" i någon skrubb. De blir snabbt föråldrade, värdelösa.
- Orealistiska förväntningar eller helt enkelt ett dåligt verktyg.
- Fixering vid verktyg istället för ett helhetsperspektiv ("a computerized mess is still a mess").

Verktygsrelaterat

- Olika funktionalitet, olika "look-and-feel". Tar tid att lära sig flera. Omställningsproblem och irritation.
- Varierande teknik-plattformar används. Anpassning till den eller de som används för andra ändamål i verksamheten önskvärd.
- Verktygen under snabb utveckling/förändring.
- Samverkan med andra verktyg normalt inte deras starkaste sida.

Strategiskt

- Verktygskostnad och grundinvestering hög, inte minst för utbildning och anpassning.
- Stora, kostsamma omställningar på grund av "paradigmskifte". Lätt att köpa verktyg, svårt att inordna i verksamheten. Här behövs kompetens, ansvar, strategi. Viktigt ta ett helhetsgrepp.
- Kostnaderna omedelbara, intäkterna långsiktiga. Intäkterna dessutom delvis svåra att kvantifiera i pengar. Vem kan sätta pris på kvalitetshöjning. Hur värderas en möjlig återanvändning av någon systemkomponents-design om x år? Hur övertyga de ansvariga?
- Systemutvecklare, programmerare med flera ser sin kompetens eller arbetsroll hotad. Alla uppskattar inte att behöva lära om, att anpassa sig till nya förutsättningar.
- Beslutsfattare är ofta konservativa, inom informationsteknologiområdet ofta på goda grunder. Case-verktygen anses i folkmun vara överreklamerade.
- Man orkar helt enkelt inte ta beslut på grund av alla kortsiktiga problem.
- Verktyg tillåts styra metodval istället för tvärtom. Inte sällan införs systemutvecklingsmetod samtidigt med verktyg. Skapar antagligen mer vilsenhet och oro än stöd.
- Start på full "bredd" direkt istället för i pilotprojekt. Även om så inte är fallet tenderar man att starta med vital istället för okritisk applikation, eftersom det i allmänhet är den förra typen som har de akutaste behoven.
- Man glömmer lätt det tunga arbetet med datainsamling. Eventuellt existerande data är sällan av känd kvalitet och därmed oanvändbar. Alternativt inte semantiskt entydigt tolkbar.

Alltså, Case-verktyg kan säkert, rätt använda, vara produktiva redskap. Men vi har också indikerat ett antal faror och problem.

Case-verktyg ska ge lämpligt stöd vid hantering av system under deras livscykler. Anlägger vi ett något generellare perspektiv talas allt oftare om behovet att se information som en resurs i verksamheten, att kunna hantera informationen som en integrerad helhetsbeskrivning av såväl verksamheten som dess informationssystem och på olika detaljeringsgrader.

2.3 Information som resurs

Beskrivningsinformation (metainformation) har i huvudsak kopplats till de behov som utveckling och underhåll av informationssystem har krävt. Med hjälp av olika typer av SEE- och Case-verktyg dokumenteras och analyseras system under dess olika stadier av sin livscykel. Alla är medvetna om att denna information är synnerligen komplex till sin struktur och semantik.

Men dessa informationssystem lever inte sina separata, isolerade liv. De är till för att ge informationsstöd till verksamheten. Det ligger därmed nära till hands att inkludera även verksamheten och hur den samspelar med informationssystemen i beskrivningen.

Centralt i det sammanhanget är givetvis att veta vilken information som hanteras av vilket system, hur, av vilka, enligt vilka regler, o s v. Hur flödar informationen mellan användare och system, mellan systemen direkt och varför? Var sker förädlingar? Både detaljuppgifter och helhetsperspektiv behövs.

Datorstöd av olika slag har traditionsenligt koncentrerat sig på att stödja livscykeln för nya system, system som kan vakas över från början till slut. I detta finns en naturlig lockelse för både produktutvecklare och användare (systemutvecklare m fl) eftersom datorstöden ligger i utvecklingens framkant och inkluderar moderna metoder, språk, stringens, avancerade modeller, ofta packeterade i moderna miljöer och gränssnitt.

Den historiska barlasten i form av s k legacy systems är i själva verket närmast ogripbart tung. Ofta är det mycket tunga stordatorbaserade tillämpningar som fortfarande i många fall står för den alldeles avgörande delen av verksamhetens informationsbehandling. Dokumentation är inaktuell, bristfällig eller svårtolkad. Underhållsarbetet är synnerligen resurskrävande. Datorstöd för reengineering eller reverse engineering blir allt fler och förhoppningsvis allt bättre. Dokumentation av legacy systems måste rimligtvis vara en viktig ingrediens i den totala beskrivningsinformationen.

Verksamheter är mångfacetterade organismer. De tenderar att bli alltmer komplexa till sin struktur, uppbyggnad och dynamiska beteende. Där finns uttalade eller outtalade drivkrafter, strategier och styrande affärsregler. Där finns komplexa och avgörande viktiga relateringar till andra externa verksamheter. Fler processer styrs mer och mer efter fördefinierade, entydigt definierade regler. Ingredienser av automatiserad produktion och informationshantering blir allt vanligare. Listan kan göras lång.

Vem i en organisation har idag tillräcklig och tillräckligt säker blick över allt detta? Står det inom rimligheternas gräns att åstadkomma det? Behövs totalöverblicken?

Allt oftare ser man åsikten att det kommer att vara helt utslagsgivande för företags överlevnad och utvecklingsmöjligheter att i framtiden ha fullt grepp om denna kunskap och förmåga att använda den för sina syften. I det perspektivet kommer det att vara en avgörande fördel att kunna hantera information som en resurs samt att ha insikten att se och uppleva information som en värdefull reell tillgång.

Nästa avsnitt skissar några ansatser till samordning av beskrivningsinformation med olika ambitionsnivåer. Resonemanget placeras in i ett kort "historiskt" tidsperspektiv.

3 Samordning

3.1 Förutsättningar

Användaren av beskrivningsinformation kan vara något datorstöd eller någon människa. De mänskliga användarna antas i normalfallet arbeta mot informationsmassan via olika typer av datorstöd, d v s inte direkt genom något frågespråk eller dylikt. Anledningen är att datorstöden dels kan ge mer specifikt anpassade användargränssnitt för olika ändamål, dels ofta behöver bearbeta informationen på ett mer eller mindre avancerat sätt till/från användaren. Vi har redan nämnt att den service som ska ges ska svara mot en mångfald olika behov och förutsättningar.

I det ena extremläget har vi en uppsättning verktyg som utvecklats vid olika tidpunkter, av olika leverantörer, för olika ändamål. En inte alltför vildsint gissning pekar på olika funktionaliteter (även för samma syften), enligt olika ambitionsnivåer, olika användargränssnitt, olika syn på och hantering av informationen. En typisk, osamordnad C-Case-situation. Denna förutsättning erbjuder ingen automatisk integrering. Om ambitionsnivån är sådan att man känner sig nöjd med respektive verktygs separata funktionalitet är situationen tillfyllest. Detta kan ju exempelvis vara fallet i en höggradigt decentraliserad organisation. Man väljer att satsa på väl fungerande lösningar för väl avgränsade behov. I de flesta fall är det antagligen en övergångslösning inför en högre ambitionsnivå.

I det andra extremläget har vi ett antal produkter som utvecklats med syfte att vara entydigt integrerbara moduler. Tillsammans erbjuder de en välordnad helhetsservice. En typisk I-Case-situation.

Bland fördelarna kan nämnas:
- Enhetlig, integrerad metod och begreppsapparat.
- Enhetlig informationsbas.
- Uppmuntrar till återanvändning.
- Enhetliga användargränssnitt.
- Fungerande gränssnitt mellan olika faser.
- Enklare upplärning.
- Enklare drift- och underhållssituation av verktyg.
- Möjliggör full projektkontroll.
- En motpart vid klagomål, önskemål, m m.

Låter inte detta som alla bekymmers lösning? För vissa verksamheter kan så mycket väl vara fallet. Det finns ju produkter på marknaden med just denna målsättning. Före ett ställningstagande kan det ändå finnas anledning att fundera över följande problemställningar:

- Behöver man, kan man dra nytta av alla I-Case-verktygets funktioner? Betalar man för "onyttigheter"?

- Att låsa sig till en leverantör är att etablera ett långsiktigt, förtroendefullt samarbete. Ansenliga initialkostnader i inköp, utbildning, anpassningar av olika slag omöjliggör snabba byten. Är verktyget uppbyggt för flexibel anpassning till den egna organisationen? Vågar man lita på att leverantören tillhandahåller både det man behöver idag och det man kommer att behöva om ett antal år? Går det överhuvudtaget att svara på den frågan?

- Har det integrerade verktyget svaga länkar? Hur hanteras dessa? Osannolikt att hitta ett verktyg som är bäst i alla avseenden eller som svarar mot alla aktuella och tänkta behov. Å andra sidan kanske verktyget är tillräckligt bra i tillräckligt många avseenden?

- Ofta finns olika typer av förutsättningar inbyggda i ett I-Case-verktyg. Exempelvis kanske det representerar en viss utvecklingsmetodik. Kanske sker valideringar av informationen vid vissa förutbestämda lägen, o s v. Det gäller att bedöma att verktygets funktionalitet svarar upp mot de egna kraven och inte tvärtom, d v s att verksamheten måste anpassa sig till verktyget. Att samtidigt behöva införa både ett nytt verktyg och och en ny utvecklingsmetodik kan vara vanskligt.

- Använder sig verksamheten redan av olika typer av datorstöd? Ska dessa "skrotas"? Om så är fallet, går det och i så fall hur kan fortfarande relevant information i dessa överföras från de existerande till det nya? Om svaret är nekande är vi ju snabbt tillbaka till den andra extremförutsättningen (ett I-Case, många C-Case).

Tänk om det gick att arbeta genom "plug-in-moduler". Då kunde varje verktyg väljas utifrån sina egna förtjänster och genom givna gränssnitt integreras i helhetsfunktionaliteten. Jämför uppbyggnaden av en modern dator, bil eller liknande. Detta utesluter givetvis inte I-Case-verktyg. De behöver bara inte nödvändigtvis ta på sig rollen som totallösning utan inordnas som ett synnerligen omfattande C-Case-verktyg.

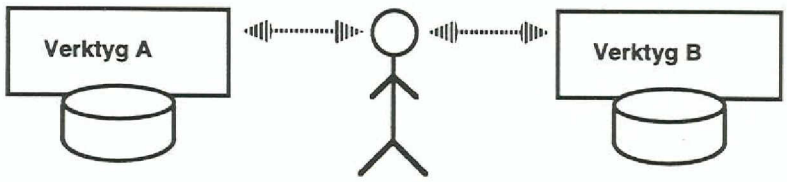
Varje verksamhet måste göra bedömningen utifrån sina egna förutsättningar. Med tanke på

- informationssystems allt större komplexitet och integreringsbehov
- kravet på heltäckande beskrivning över dess fulla livscykel
- den alltmer expanderade rollen information generellt kommer att spela som en viktig resurs
- alltfler användares alltmer diversifierade behov av att arbeta med beskrivningsinformationen
- alltfler användares alltmer diversifierade önskemål avseende hur behoven kan tillgodoses
- den allmänna trenden mot decentralisering och ansvarsdelegering tillsammans med spridningen av datorkraften
- en allt snabbare utveckling av teknik, metoder, modeller och tätare "paradigmskiften"

är det förmodligen ett rimligt antagande att vi i normalläget behöver tillgång till flera/många olika datorstöd för olika ändamål och att vi då och då behöver kunna byta ut och lägga till. Konstaterar vi samtidigt att de behöver kunna hantera delar av den totala beskrivningsinformationen på ett samordnat sätt, uppstår ett integreringsbehov. Integrering kan åstadkommas med olika ambitionsnivåer.

3.2 Olika grad av integrering

I sin enklaste form är människan/användaren integratorn. Denne tolkar information från olika verktyg och översätter vid behov "efter eget huvud och bästa förstånd" mellan dem vid de lägen där verktygen hanterar överlappande information. Även om inte informationen är överlappande, hänger den antagligen ihop på något sätt. Om och hur kopplingen existerar ligger också i användarens huvud. Olika människor gör givetvis olika antaganden.



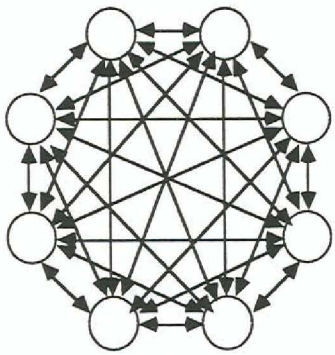
Figur 1

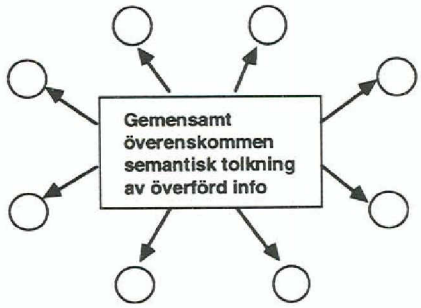
Många gånger önskar man direkt överföra uppgifter mellan verktyg i samband med sammanhängande bearbetningar. Att då behöva gå över en "mänsklig översättare" är både tungt och riskfyllt. Man definierar en översättningsprincip mellan de båda verktygens information och realiserar denna i form av en "brygga", ett slags översättningsprogram. Förutom att programmen alltid översätter enligt samma regler (till skillnad från människor), är de mycket snabbare. Bryggor kan vara enkel- eller dubbelriktade. En given förutsättning är att bryggan kan tolka såväl syntax som semantik hos det inkommande samt kan översätta vidare enligt syntax och semantik som det mottagande verktyget kan hantera. Många verktyg har standardiserade in- och utgångar för extern kommunikation, så kallade import/export-faciliteter. Andra får tillverka i samband med det uppkomna behovet.

Framförallt den semantiska aspekten kan behöva diskuteras ingående mellan företrädare för de båda verktygen och brygg-tillverkaren. Semantiska feltolkningar kan ge allvarliga följdverkningar. Svårare att göra något åt är sådana kvalitetsförsämringar som exempelvis blir följden vid översättning mellan verktyg vars information uttrycks genom modeller med olika semantisk uttryckskraft. Det är i realiteten svårt att utväxla information med garanti för semantisk överensstämmelse. Hur lämnas förresten en sådan garanti av de verktygs- eller bryggleverantörer som inte arbetar med öppna gränssnitt eller brygg-specifikationer?

Figur 2

Om många verktyg blir inblandade behövs det parvisa bryggor mellan varje kombination (figur 3 a). Det kan bli en tuff uppgift för ett verktyg att tillgodose för många externa behov. För att inte tala om det kontinuerliga underhållet av bryggorna! Ett sätt att underlätta denna situation är att för varje verktyg översätta till och från en "neutral form". Bryggorna ersätts med två översättningsmekanismer per verktyg, en till den neutrala formen och en från (figur 3 b).





Figur 3 a

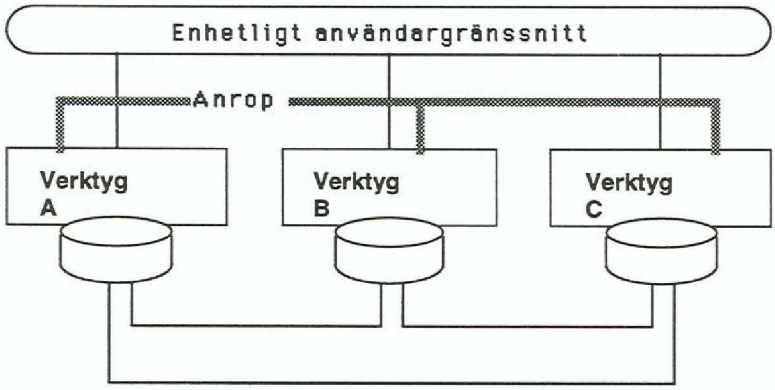
Figur 3 b

Beroende på lösning och ambitionsnivå formuleras semantiken som en del av överföringen, alternativt utnyttjas fördefinierade överföringsformat. Idén kring denna ambitionsnivå är enkel och tilltalande, inte minst i situationer där informationsutbyte behövs mer sällan eller där man inte känner sig mogen till eller tilltalad av en fastare samordning enligt något av alternativen, nedan. Problematiken ligger i möjligheten att ta fram en gemensamt överenskommen "neutral form", en öppen översättningsmodell. En översättningsmodell är å andra sidan bara bindande i och för en överföring. Hur respektive verktyg internt upplever och hanterar sina uppgifter är deras privata ensak. Inte heller är översättningsmodellen bindande för något realiserat datalager, så som blir fallet med en renodlad repository-lösning.

Flera standardiseringssträvanden arbetar enligt denna princip. Dit hör CDIF (CASE Data Interchange Format), se Triad rapport K 21 och GDMO (Guidelines for Definition of Managed Objects), se Triad rapport K 25.

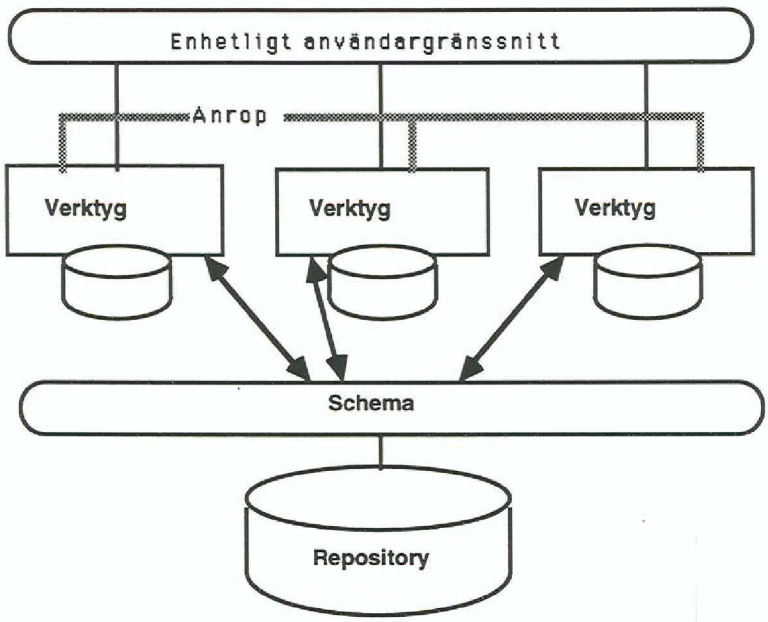
Användare kan behöva arbeta med flera verktyg. Användare byter arbetsuppgifter inom en verksamhet. Oavsett vilket, vore det en given fördel om samtliga verktyg erbjuder ett gemensamt definierat användargränssnitt för hantering av beskrivningsinformation. Gränssnittet styr upp både presentation och olika typer av generella handgrepp. Här anpassar man sig lämpligen efter de standarder som numer etablerats. Därtill ska gränssnittet om möjligt reglera annan enhetlighet av mer tillämpningsorienterad karaktär (exempelvis formulär-uppbyggnad, grafiska primitiver, m m).

Förutom att verktygen vid behov utbyter information enligt något av alternativen ovan, kan de behöva samarbeta för att utföra en sammansatt, mer komplex operation. Detta sker genom någon form av anrop. Figur 4 visar denna något mer sammanflätade arkitektur. De tunna linjerna representerar bryggor.



Figur 4

Ju mer överlappande information, ju mer de olika verktygen behöver samarbeta, ju större krav som ställs på informationens aktualitet, ju mer man behöver bearbeta helhetsinformation, desto angelägnare blir behovet av ett gemensamt informationslager, en slags gemensam databas för beskrivningsinformation. Vi är framme vid behovet av ett repository, figur 5.



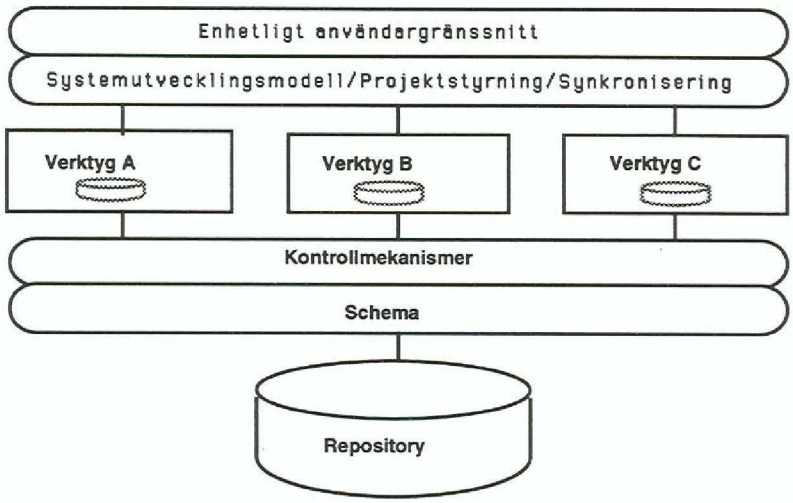
Figur 5

Att man valt att lägga en distinktion mellan begreppet databas och repository har flera skäl. Dels vill man notera att det är fråga om ett mycket specifikt och komplext tillämpningsområde både vad gäller informationens struktur och vad den representerar. Dels ser man en relativt avancerad arkitektur av samverkande men annars ganska fristående system. Dels ställer tillämpningsområdet i sig specifika och avancerade krav på databashanterarens servicerepertoar. Mer om detta i avsnitt 4.

På samma sätt som en databastillämpning presenterar sin syn på den hanterade informationen i form av ett schema, formulerar även ett repository den gemensamt överenskomna synen på innehållet genom ett schema. I databassammanhang formuleras numer huvudparten av scheman för nyutvecklade tillämpningar med hjälp av relationsmodellens begreppsapparat. Den har inte ansetts erbjuda tillräcklig semantisk kraft för repositorybehov. Som tidigare nämnts kommer i normalfallet många olika verktyg och ännu fler användare av olika kategorier, olika bakgrund och med olika behov att arbeta mot ett repository. Därmed följer i första hand höga krav på klar, entydig och enkelt tolkbar innebörd av schemat. Schemat ska snarare fylla rollen som en semantiskt entydig beskrivning än en logisk struktur för implementering. De flesta repository-ansatser tillämpar någon form av objektorienterad begreppsapparat för att formulera sitt schema.

I figur 5 har respektive verktyg fortfarande sin lokala databas som komplement till ett repository. I repositoryt läggs enbart den information som är av gemensamt intresse och som bedöms vara i ett tillräckligt väl genomarbetat skick för

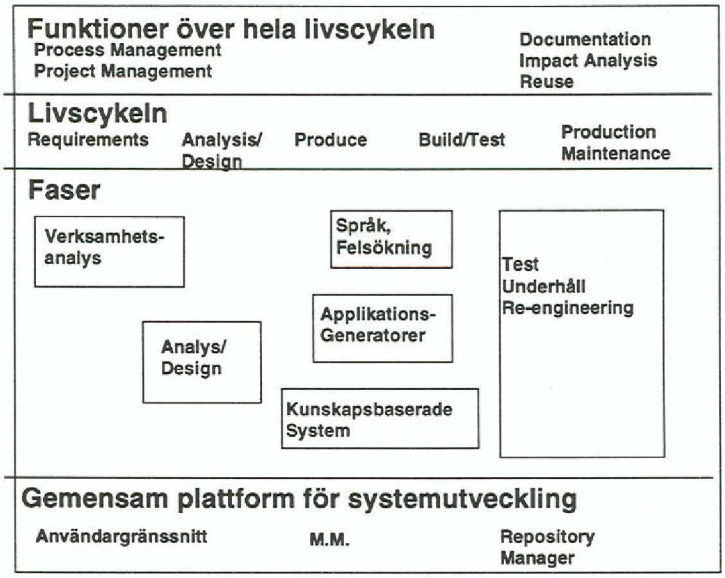
"omvärldens" ögon. Under den första repositorygenerationens "glansålder" sågs dessa separata databaser som en olycklig kompromiss. Den fulla samordningen och kontrollen saknades och orsakade brister i helhetssynen. Lokala behov kunde åstadkommas genom olika typer av versionshantering inom det totalgemensamma repositoryts ram. Vi är framme vid figur 6. De lokala databaserna borde endast vara temporära lagringsplatser. Integrationen manifesteras ytterligare genom komplettering av ett arbetsstyrningsskikt baserat på en lämplig systemutvecklingsmodell. Dessutom finns ett skikt "kontrollmekanismer" avsett att indikera en intelligens och vaksamhet på informationsströmmarna till och från repositoryt i och för lämpliga åtgärder när vissa specifika tillstånd, felaktigheter eller andra förutsättningar inträffar.



Figur 6

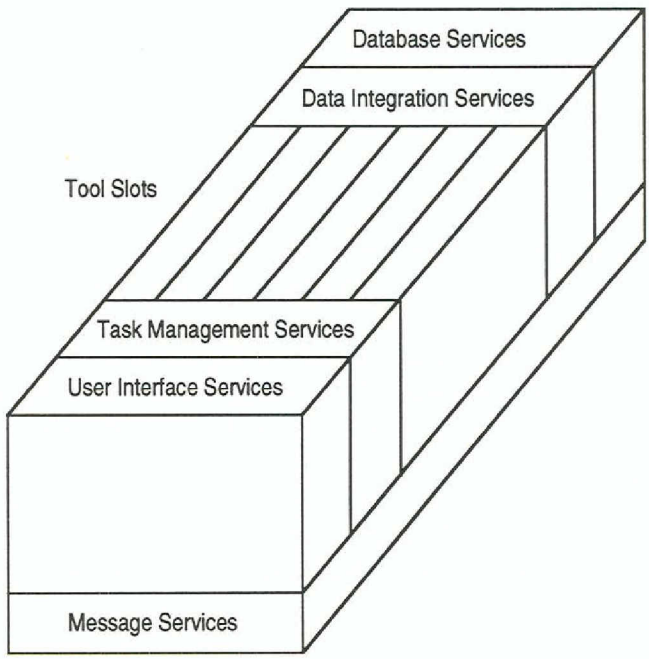
Givet den skissade kringmiljön ska de verktyg som ställer upp på förutsättningarna enkelt och riskfritt kunna "pluggas" in och ut efter tycke och behov.

Därmed är vi framme vid de rådande tankegångarna vi 1990-talets början. Visionen var klar. Repositoryt skulle ligga i stordatormiljö medan verktygen kunde ligga på arbetsstationer. Lokala repositoryn kunde tänkas, om de enligt givna regler checkade ut och in sin information till och från det centrala repositoryt. De lokala skulle alltså främst omfatta från det centrala repositoryt kopierade data i och för något i allmänhet tidsavgränsat behov. IBMs AD/Cycle-koncept var helt tongivande för diskussionen. Se figur 7.



Figur 7

Även den av ECMA framtagna referensmodellen innehöll en mycket kopierad idéskiss i form av en "toaster model". Se figur 8.



Figur 8

3.3 Mot en distribuerad lösning

I teorin tycktes lösningen vara enkel, renodlad, tilltalande. Mycket lite kritik riktades i artiklar och på konferenser. Det var inte fråga om om, utan möjligen om när. Varför visade sig då teori och verklighet inte gå hand i hand? Jo, bland annat av följande anledningar.

- Mycket information var i realiteten endast lokalt, affärsområdes- eller projekt-intressant, åtminstone innan det nått stabilt stadium.

- Ingen schema-ansvarig organisation kunde hävda att de företrädde någon de facto-standard.

- Schemat till ett totalt repository visade sig bli enormt stort. Både schemat och repositoryts innehåll blev övermäktigt att hantera, till och med för den flinkaste databasadministratör. För att inte tala om alla de effekter de mer eller mindre kontinuerligt behövliga utvidgningarna och modifieringarna orsakade.

- Knappast några företag var mogna att hantera sina uppgifter på det ambitiösa sätt som repositoryidén förespråkade.

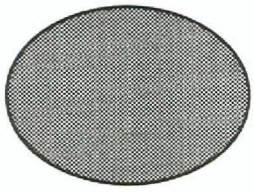
- Stordatorerna ersattes i allt snabbare takt av olika typer av nätverks-lösningar, client/server o s v.

- De repository-produkter som kom fram, svarade inte mot behoven varken i prestanda eller funktionalitet.

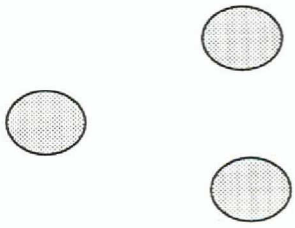
- Ett repository är i sig ganska ointressant. Det är i kombination med en uppsättning verktyg som en produktiv miljö kan åstadkommas. Alltför få produkter orkade/lyckades/ville anpassa sig mot existerande repository produkter.

Till att börja med var det naturligt att backa till figur 5. Lokala repositorier fick större tyngd. Centralt lagrade behövde endast de mest företagsgemensamma och vitala uppgifterna ligga, exv datadefinitioner. Eftersom det inte var fråga om information av någon extraordinär komplexitet eller information under intensiv förändring kunde satsvisa uppdateringar och uttag accepteras. Enkla bryggor kunde tillverkas, alternativt överföring enligt CDIF-principer eller liknande.

Istället för figur 5's uppdelning per verktyg kan en intressantare indelning vara gruppering per projekt, per affärsområde eller dylikt där samordningsbehovet sannolikt är mera påtagligt. Centralt repository, symboliserat med figur 9 ersätts med ett antal lokala, symboliserade enligt figur 10.

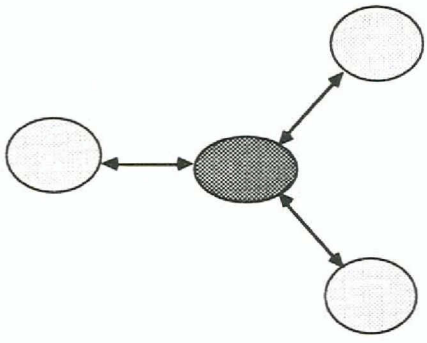


Figur 9



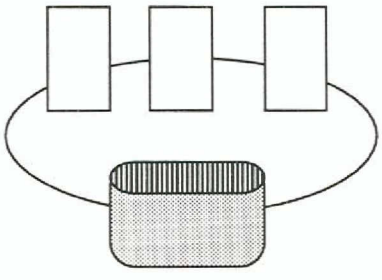
Figur 10

Fortfarande kan en central, integrerad kärna vara motiverad på samma grunder som i figur 5 ovan. En balanserad uppdelning av central och lokal repository-information visas i figur 11.



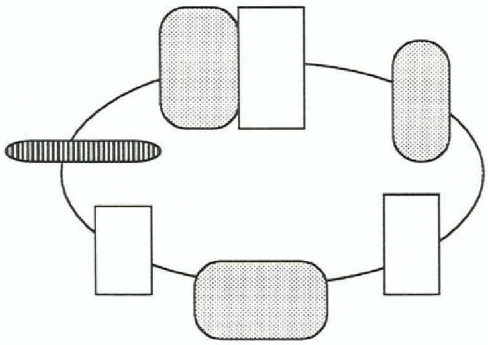
Figur 11

Återigen gäller då att flera verktyg kan behövas för det gemensamma syftet. Detta gäller både ett lokalt och ett centralt repository. Vi har en mikro-situation av figur 5 i varje punkt. En client/server-lösning ligger nära till hands, där repositoryt utgör servern. Sträckad del utgör schemat. Se figur 12.



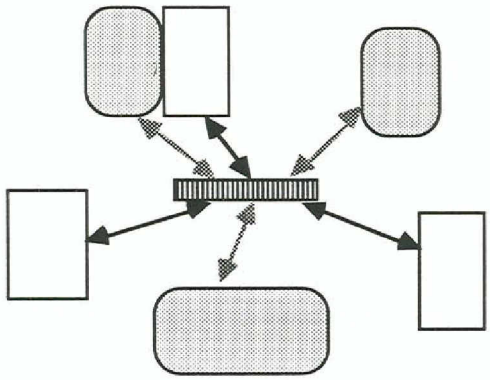
Figur 12

Av prestandaskäl, säkerhetsskäl, behörighetsskäl m m kan repositoryts innehåll behöva delas upp på flera platser, flera servrar eller i serverfunktion hos någon client. Kravet är dock att informationen måste kunna sammanhållas som en logisk enhet, snarlikt en distribuerad databaslösning, figur 13.



Figur 13

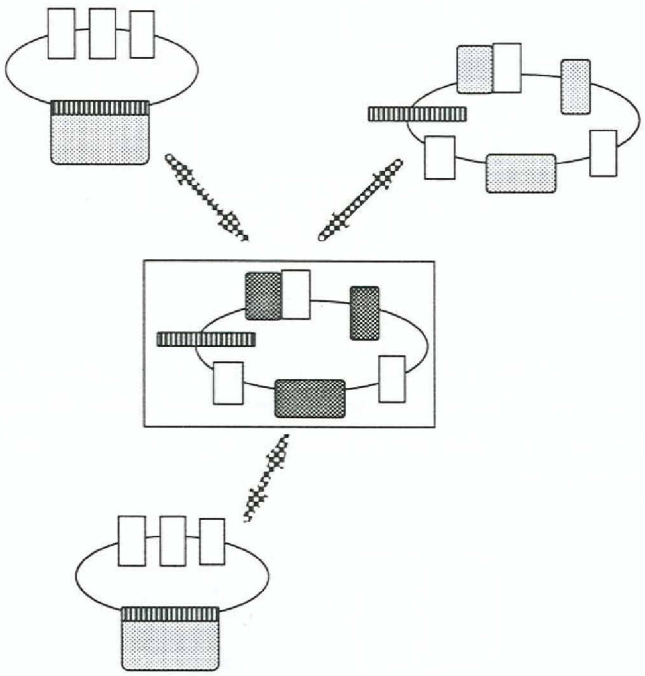
Repositoryhanteraren tar emot begäranden från verktygen, slår i schemat upp vilken delinformation som finns var, sänder ut begäranden till respektive del, tar emot och sammanställer svaren/konfirmeringarna samt meddelar verktyget resultatet, figur 14.



Figur 14

Nu börjar det bli komplicerat. Här gäller det med andra ord att hålla reda på gällande konfiguration och att hålla reda på vilken information som finns var. Vid komplexa frågor och uppdateringar kan fler servrar vara inblandade. Eventuellt behöver man kunna hantera duplikat. Om så, hur utnyttjar man denna kunskap för snabbaste utsökning. Hur sker uppdateringar av dessa? Hur upprätthålls konsistens, realiseras roll-back, rätt sekvens mellan hanteringssteg?

Verktyg eller annan programkod som behöver samverka ligger kanske på olika ställen över nätet. Kanske finns även där duplikat. Hur åstadkoms denna samverkan på ett korrekt sätt? Eftersom man önskar se informationen som en logisk enhet måste antagligen olika versioner kunna hanteras av samma informationselement. Finns strikta mekanismer för detta? Kraven är många. Figur 15 visar en mer komplex bild av grundtemat i figur 11.



Figur 15

Observera, att även schemat kan behöva distribueras, dupliceras m m.

Databashanterare för distribuerad information klarar en del av kraven men inte alla, exempelvis inte processintegration. I det generella fallet kan dessutom de olika servrarna betjänas av olika databashanterare. Då måste samordningen flyttas till en annan implementeringsoberoende logisk nivå, till en övergripande lokal dataintegration. Till skillnad från ett vanligt schema som definitionsmässigt ger enhetlighet och totalintegration redovisar vi här för integrering bara det vi önskar och kan hantera. Den enhetliga logiska nivån ska endast tolkas som en beskrivning över vad som finns var. Önskad samordning måste explicit anges. Eventuella namnförbistringar, strukturella inkonsistenser m m ligger utanför ansvarsramen.

Visst kan repositoryleverantörer och andra bygga upp en avancerad miljö enligt ovan. Problemet är att det då sannolikt byggts enligt egna förutsättningar och lösningsbedömningar. Vi är tillbaka till ett leverantörsberoende. Vad som önskas är en till standard upphöjd specifikation, en specifikation som leverantörer i gemen är villiga att rätta sig efter. Då först får vi en öppen, leverantörsobunden miljö. Mer om detta under standardiseringsaktiviteter, speciellt PCTE (Portable Common Tool Environment), avsnitt 6.

3.4 Dagsläget

AD/Cycle som idéskiss har försvunnit bakom kulisserna tillsammans med IBMs stordatorversion av repositoryt. De flesta leverantörer och användare tycks nu vara övertygade om att vägen till integration av beskrivningsinformation går över distribuerade repositoryn för olika lokala behov med inslag av en kärna av verksamhetsintegrerad information i ett centralt repository. Produkter finns, fler är på gång. De strävar efter att kunna åstadkomma semantisk och i vissa fall funktionell samverkan med Case-verktyg – en förutsättning för deras existensberättigande på marknaden.

Case-leverantörerna strävar å sin sida mot att göra sina verktyg mer I-Case-orienterade, mot att inkludera olika former av 4G-funktionalitet och modellexekvering och att erbjuda moduler för (i första hand) import av information från andra verktyg.

Även om verktygen varit ett bekymmer, ligger antagligen den huvudsakliga problematiken på användarsidan. Där saknas normalt såväl uppgifter att stoppa in i repositoryt som metod- och organisatoriska förutsättningar för en konsekvent användning. Där repository används, är det för mycket avgränsade ändamål, d v s med en begränsad komplexitet i repositoryts schema.

Därmed går vi över från en tidsorienterad syn på repository som företeelse till att titta närmare på vad repository står för som företeelse.

4 Repository; principer, egenskaper, användning

4.1 Syfte

Ett repository ska kunna fungera som lagringsplats för mer eller mindre integrerad information om en verksamhet och verksamhetens informationssystem under olika faser av dessas livscykler.

Något mer utförligt uttryckt:
Ett repository är en plats för att lagra flera versioner av information om
– en verksamhet, dess mål, organisation, aktiviteter, informationsflöden
– verksamhetens begrepp, språk och regler
– tillämpningar, dess funktioner och data
– sambanden mellan dessa delar

i ett språk och en form som tillåter entydig utsökning och sammanställning av dessa uppgifter för många behov till behöriga mottagare.

Visionen är alltså en sammanhållen beskrivning av en organisations verksamhet, dess data och applikationer samt hur de är relaterade till varandra. Denna beskrivningsinformation går ofta under benämningen "metainformation". (Meta- och beskrivningsinformation används fortsättningsvis som synonymer.) De som arbetar med denna typ av information är verksamhetsutvecklare, strateger, systemutvecklare, databas/systemadministratörer, programmerare, driftspersonal m fl.

I visionen ligger även perspektivet att kunna uttrycka metainformationen under
- olika grad av färdigställande
- olika grad av integrering
- olika detaljeringsgrad
- olika tidsperspektiv och versioner.

Med en högre ambitionsnivå bör informationen även innefatta olika aspekter på en verksamhet i sig, dess syfte, regler, bedrivande och resurser.

I engelsk-svenska lexikon betyder repository, något oglamoröst, en" förvaringsplats" och till och med ett "sista vilorum". Låt oss hoppas att det inte är en profetia. Svenska motsvarigheter är bl a **resurskatalog** eller **avancerad datakatalog**. Terminologin är långtifrån stabil. Synen på ett repository som en integrerad lagringsplats för metainformation, uttryckt i termer av en enhetlig datamodell och över ett enhetligt gränssnitt, tycks dock allmänt accepterad. Kanske är det så att begreppets i grunden generella innebörd bidragit till alla vitt skilda uppfattningar om dess mer precisa tolkning. Vi väljer att försvenska och använda begreppet "repository" i brist på stabil svensk terminologi.

Inom internationella standardiseringssammanhang används den mer klargörande beteckningen IRD (Information Resource Dictionary).

Ambitionsnivå och behov styr vilken samordning, d v s vilken repository-profil, som eftersträvas. Än så länge befinner vi oss i alla händelser långt från den indikerade visionen.

Om informationen i ett repository ligger distribuerad eller ej är ur ett användarperspektiv ointressant så länge som den styrs av ett gemensamt logiskt schema och för övrigt beter sig likvärdigt avseende prestanda m m. Distributionsaspekten får däremot en alldeles påtaglig relevans om beskrivnings- eller metainformationen ligger utplacerad under ett antal fristående, logiska scheman. Antingen kan detta ha realiserats genom en uppdelning per verktyg enligt figur 5 och 6 eller uppdelat per projekt, affärsområde eller annat behov enligt figur 12 och 13. Så länge som dessa enligt etablerade förutsättningar tillåts leva sina separata liv, är de helt enkelt att ses som ett antal separata repositoryn. Så snart de enligt någon ambitionsnivå ska integreras, kan knepiga situationer uppstå av både teknisk och semantisk natur. Se exempelvis figur 15 ovan.

4.2 Typer av repositories

Repositoryn brukar indelas i undergrupper med hänsyn till olika egenskaper, användningsområden et c.

Vi har redan nämnt uppdelning i I-Case respektive C-Case beroende på hur mycket av den fulla livscykeln de täcker in.

Ofta försöker man argumentera att det finns en skillnad mellan **repository, encyclopedia** och **data dictionary**. I allmänhet tycks man anse att repository används för den mest generella definitionen, encyclopedia för det repository som finns under visst Case-verktyg och data dictionary för en mycket begränsad funktionalitet, exempelvis som ren datakatalog.

En annan indelning baseras på hur repositoryt används:

Passiva

Används av människor för att lagra modeller av olika slag. Fungerar som ett slags kunskapsstöd på samma sätt som vanliga databaser. Ingen direktkontakt mellan dokumentation och det som dokumenteras. Hanterar i allmänhet Case-verktygens specifikationer. Sannolikheten för inaktuell eller felaktig information är påtaglig och bevisad av mången erfarenhet.

Aktiva under utveckling

All information produceras och hämtas tvingande via förutbestämda "mekanismer" (exempelvis programs datastrukturer). Generering av kod är ett exempel.

Aktiva under produktion (Dynamiska)

Informationen utnyttjas som specifikation för exekveringar av olika slag. Kan innehålla hela exekveringsmiljöer i form av laddmoduler, realiserade databasscheman, behörighetsspecifikationer och annat som efterfrågas under exekvering.

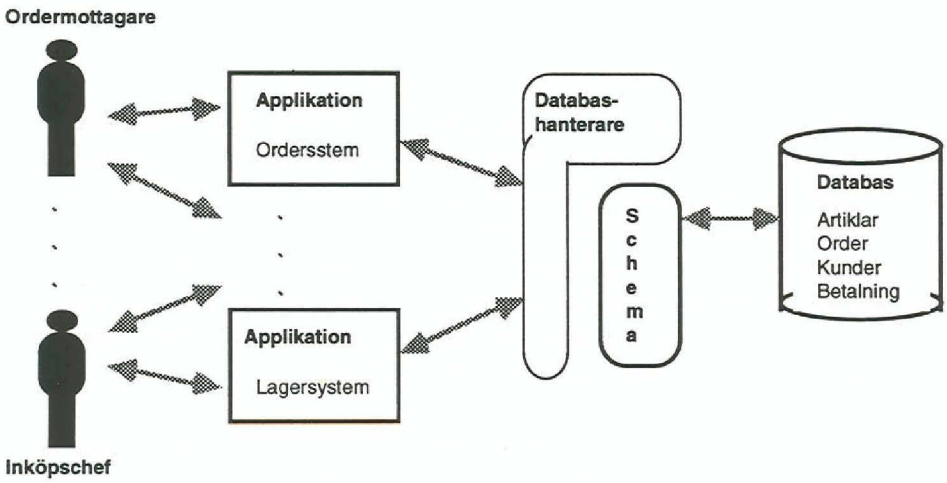
Observera att en repository "mode" inte behöver gälla permanent och över hela datamassan. Vissa tillämpningar kan nyttja repository som aktivt under produktion, andra dokumenterar endast passivt o s v.

Ett dynamiskt repository borde kunna vara det långsiktigt eftersträvansvärda. I ett sådant finns ingen diskrepans mellan beskrivning av ett system i ett repository och systemet självt, vilket automatiskt ökar värdet på beskrivningen och hela repositoryfunktionaliteten. Å andra sidan ställer det extraordinära krav på informationens precision och kvalitet.

Den databaskunnige eller mer kritiskt lagde frågar sig säkert vad det är för speciellt med ett repository jämfört med en vanlig databasfunktionalitet. Frågan är i ett långsiktigt perspektiv i högsta grad relevant. I närperspektivet brukar man argumentera enligt diskussionen i de två följande avsnitten.

4.3 Vad är skillnaden mot en vanlig databas?

Vanliga databastillämpningar innefattar vanligtvis ett antal olika funktioner för att svara mot många olika behov av informationsbearbetning. En förutsättning för detta är en gemensam syn på data, redovisad i databasens schema. Data hanteras av en databashanterare i enlighet med innehållet i schemat. Se figur 16.

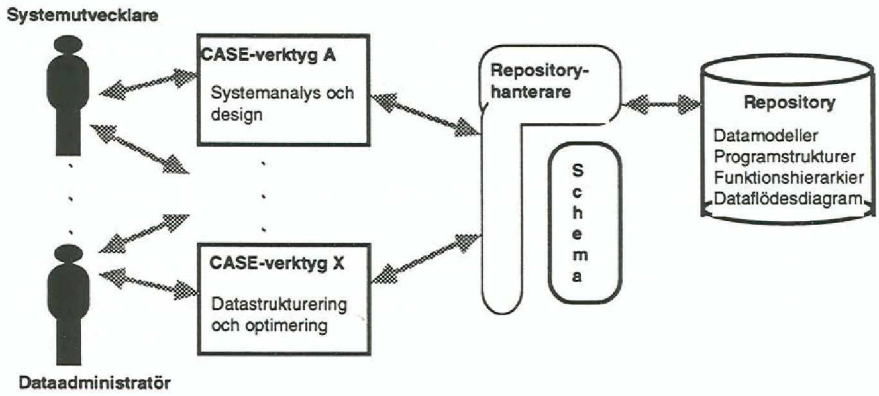


Figur 16

Samma förutsättningar gäller i princip för en repositoryhanterare. De som har behov av att nyttja metainformationen i repositoryt gör det i allmänhet via olika typer av CASE-verktyg. Verktygens funktioner kan i det perspektivet ses som tillämpningsfunktioner. Informationen ska kunna hanteras på ett enhetligt sätt. För ändamålet finns ett överenskommet, gemensamt schema. Självfallet får varje verktyg sedan lokalt mot sina användare formulera denna information enligt principer som svarar mot verktygets profil och användningsområden.

Den mest påtagliga skillnaden är att vi i repositoryt hanterar metainformation och inte "vanliga" uppgifter om verksamheten i form av artiklar, order, personal o s v. Se figur 17.

Där återfinns data om data, mjukvaror, nätverk, applikationer, system under utveckling, verksamhetsmål, organisation, personer, ja det mesta som beskriver ett företags verksamhet och dess informationssystem.



Figur 17

Jämfört med en "vanlig" databastillämpning blir schemat för ett repository sannolikt mycket mer komplext, både vad gäller antalet objekttyper och antalet sambandstyper dem emellan. Det schema IBM definierade under beteckningen Information Model för sin Repository Manager-produkt omfattade flera hundra objekttyper och över tusen sambandstyper innan projektet lades ner. Observera, att det ändå bara täckte in en liten del av de modeller som beskriver ett system under dess livscykel. För mer detaljerad information om några av delmodellerna hänvisas till Triad-rapporterna K6, K7, K12, K13, K14, K15.

Det finns en tendens att se Information Model som en del av den nedlagda IBM-produkten Repository Manager och av den anledningen ointressant. Inget kan vara felaktigare. Ingående delmodeller är i högsta grad aktuella och relevanta. Jämför med en normal databastillämpning. Ett schema för en artikeldatabas blir inte inaktuell bara för att man byter mellan exempelvis databashanterarna Ingres och Oracle. Även om man skulle byta mellan databas-

hanterare representerande olika begreppsapparater är ju den kunskap om tillämpningsområdet som representeras genom det existerande schemat av synnerligen stort värde vid formuleringen av det nya schemat. Inte minst gäller detta om det existerande schemat uttrycks i ett näst intill implementeringsoberoende språk, vilket är fallet med Information Model. Ett alldeles påtagligt belägg för resonemanget finns i det faktum att modellerna nu tagits över för revidering, vidareutveckling och standardisering inom CDIF (se Triad-rapport K21).

4.4 Specifika repositoryegenskaper

Förutom likheterna finns även specifika krav på en repositoryhanterare, som dagens databashanterare normalt inte svarar upp mot (men som mycket väl kan komma att inkluderas så småningom).

- Versionshantering. Ett utvecklingsprojekt genomlöper ett antal faser. Samma sak gäller producerade resultat. Vad ett projekt producerat kanske kopieras av ett annat projekt. De båda projekten arbetar sedan parallellt fram nya versioner utifrån egna villkor och behov. Vid något senare tillfälle kanske de separata versionerna integreras igen o s v. En specifikation existerar alltså i ett antal versioner. Det kan finnas behov av att kunna gå tillbaka och titta på eller starta nya aktiviteter baserade på tidigare versioner. Varje version och dess beståndsdelar måste kunna sparas och hanteras som en enhet.

- Konfigurationshantering. En viss konfiguration är en sammansättning av komponenter i repositoryt. Den kan avse programkod, datamodell, sammansatt funktion eller dylikt. För varje komponent är endast en viss version av densamma aktuell. Ändras en sådan version av en komponent får det återverkningar på de konfigurationer som baseras på just denna version av komponenten. Konfigurationen kanske måste omgenereras. Konfigurationshantering håller reda på en konfigurationskomponenter och versioner samt kan utnyttja denna kunskap för att skapa en konfiguration. En vanlig funktionalitet vid programkodsframställning.

- Långa transaktioner. En design av något slag kan ta lång tid i anspråk. Den måste kanske förankras, justeras, testas, m m innan den anses färdig för allmän insyn via databasen. Under arbetet, som kan ta dagar eller veckor, vill man kunna jobba i fred med dess data och de grunduppgifter den baserar sig på. Ordinarie låsningsrutiner är ett för trubbigt redskap.

- Dataegenskaper. De ingående beståndsdelarna (elementen) i ett repository behöver beskrivas med hjälp av ett antal attribut för att man rätt ska kunna tolka dess valör och användbarhet. Till attributen hör vem som definierat elementet, dess ursprung, när det skapades, bedömd kvalitet eller noggrannhet på valören, etc. (En slags information om beskrivningsinformationen eller metametainformation.)

- Beslutshistorik. Olika typer av beslut fattas under arbetet med elementen i ett repository. Ett historiskt perspektiv kan vara ett viktigt underlag för att bedöma hur och varför elementet har fått aktuell definition. Informationen omfattar vem som beslutade om vilken ändring, varför och när.

- Utvecklingsmetoder och projektstyrning. Häri innefattas information i form av metodstöd, projektlägen, projektkoordination och villkor m m. Stödet kan ges i form av rekommendationer, direkt styrning, automatiskt utförda operationer, o s v beroende på ambitionsnivå. Observera att vi här egentligen utnyttjar repository-schemat som databas, kompletterat med metodbeskrivningar et c och med metametamodellen som schema.

4.5 Teknik idag

Varje CASE-leverantör kan sägas ha ett repository kopplat till sin produkt eller produktfamilj, dock inte med syftet att åstadkomma en generell plattform.

Bland genuina repository-leverantörer kan nämnas
- Digital, CDD/Repository
- R&O, Rochade
- Reltech, DB Excel
- Brownstone, DataDictionary/Solution och Client/Server Repository
- Manager Software Products (MSP), MethodManager
- IBM, AD/Platform
- Infospan, IRMS

Vi kan allmänt konstatera att samtliga numer strävar efter att ta fram funktionalitet för att ge stöd åt mer lokala behov (projekt, affärsområde, etc). Vissa erbjuder därutöver kopplingar till verksamhetscentrala repositoryn. Det ligger utanför syftet med denna rapport att närmare gå in på respektive produkts egenskaper. Dessa är dessutom i högsta grad föränderliga.

5 Införande

5.1 Nödvändiga förutsättningar

Repository som företeelse låter både rätt och riktigt. Än så länge är det dock mer vision än verklighet. Där finns bekymmer av både teknisk och hanteringsmässig natur. Existerande produkter är nya. De finns endast tillgängliga under begränsade dator/OS-miljöer. Erfarenheterna är begränsade.

Det största hindret är nog annars att ytterst få företag idag kan nyttiggöra den kapacitet som ett repository erbjuder. För en produktiv användning krävs bl a:

- Fungerande, etablerad systemutvecklingsmodell.
- Fungerande, etablerad informationsadministration.
- Fungerande, etablerad projektadministration och styrning.
- God helhetssyn på verksamheten och dess beslutsprocesser.
- Acceptans och förståelse från ledningen avseende konsekvenser för verksamheten som helhet.
- Realistisk uppfattning om investeringsbehov i människor och teknik.

I detta perspektiv ter sig produktval och produktegenskaper som klart mindre bekymmer, även om den problematiken också behöver beaktas.

5.2 Val av repository

Val av repository bör baseras på noga genomtänkta överväganden. Viktiga faktorer är bl a

- Vilka Case-verktyg kommer vi att använda?
- Vilken information behöver hanteras?
- Vilka metoder ska kunna stödjas?
- Projektbaserad eller verksamhetsbaserad information?
- Krav på informationskontroll (behörighet, spridning, ..)?
- Behöver vi kunna expandera schema för egna behov? I så fall hur och hur mycket?
- Krav på integration, plattformar?
- Kommer det tänkta repositoryts finesser att användas?
- Leverantörens stabilitet, m m.

Sist men inte minst bör frågan "Behöver vi ett repository?" ställas.

Även om företagen inte känner sig mogna för ett repository just nu, är det viktigt att avdela viss tid för allmänbevakning av områdets utveckling. Att etablera rimliga förutsättningar för ett införande tar tid. Att slå sig till ro kan innebära tempoförlust. Att rusa iväg utan eftertanke i tron att repositoryt mer eller mindre automatiskt löser alla problem är lika farligt. Det är lätt att känna sig hetsad av alla rappa budskap kring nya företeelser och begrepp. En god blandning av kunskap, realism och långsiktigt perspektiv behövs.

5.3 Införandestrategi

I artiklar och rapporter gavs tidigt förslag på lämpliga tillvägagångssätt för ett repository-införande. Eftersom varje verksamhet är unik är det också sannolikt att ett införande bör hanteras med lyhördhet för den egna organisationens förutsättningar. En "syntes" av diverse noterade råd i artiklar av olika slag kan tjäna som vägledning:

- Inventera de verktyg som verksamheten använder idag. Används inte Case-verktyg idag är det lämpligt att först utnyttja ett sådant för att vinna erfarenheter kring hantering av beskrivningsinformation.

- Etablera ekonomiskt och ansvarsmässigt stöd. Utan stöd från ledning och budgetansvariga kommer projektet sannolikt att misslyckas.

- Klargör målsättning och ambitionsnivå. Vilken beskrivningsinformation ska exempelvis hanteras i första fasen? Få datadefinitioner och namngivningsregler att stämma överens. Uppgifter ska inte bara samlas i repositoryt, det ska komma till användning och underhållas! Inventera existerande och tillgänglig information. Kan den användas? Fokusera på vad som är viktigt, d v s strategisk eller återanvändbar information.

- Avsätt ordentligt med tid för utbildning av samtliga inblandade personer. Repositoryidén och även dess realisering upplevs lätt abstrakt. Färdigställ därför ett antal mindre men påtagliga scenarier som konkret visar vilken information som hanteras i en viss situation, varför och vilka effekter som uppnås.

- Räkna med att det kostar. Upprätta kortsiktiga och långsiktiga kalkyler. Realistiska!

- Etablera en enhetlig systemutvecklingsmetodik. Se till att den blir använd (kan i sig vara en mycket svår uppgift och omvälvande för företaget).

- Starta försiktigt med begränsade ambitioner. Försök inte med någon totalintegrering, snarare användning inom projekt eller verksamhetsområde. Se dock till att etablera införandet som en del av den ordinarie verksamheten, annars skapas inte realistiska förutsättningar beträffande insatser, engagemang, behov m m. En normal projektorganisation ska givetvis finnas.

- Bestäm tidigt kriterier för en utvärdering och uppföljning. Se till att den genomförs vid en tidigt överenskommen tidpunkt.

6 Standardiserings-strävanden

6.1 Allmänt

Standardiseringsaktiviteter har sedan ett antal år aktivt bedrivits både inom det internationella standardiseringsorganet ISO (International Standards Organization) och inom USA av ANSI (American National Standards Institute).

Intressant att standardisera kan bland annat vara
- idéer, referensmodeller och ramverk
- repositoryinnehåll
- språk för att formulera repositoryinnehåll
- repositorygränssnitt.

ISO och ANSI har tagit fram ett antal standarder. De baseras på samma grundfilosofi och formuleras alla under det gemensamt accepterade begreppet IRDS (Information Resource Dictionary System). Därmed upphör likheterna. ISO har hämtat sitt beskrivningsspråk för innehåll och gränssnitt från relationsmodellen medan ANSI hämtat det från en variant av en Entity-Relationship-modell. Möjligen kan, lite elakt, en annan likhet vara värd att nämnas: Standardernas genomslag i produkter och användning har i stort sett varit obefintligt. Följande två avsnitt redogör kortfattat för existerande standarder. Avsnitt 6.4 ger en indikation om vad som kan förväntas härnäst.

6.2 ISO/IRDS

Arbetet bedrivs inom ISO/SC21/WG3. På grund av den internationella sammansättningen av gruppen framskrider arbetet relativt långsamt och med mycket kompromissande. Följande har åstadkommits:

IRDS Framework, ISO/IEC 10027, 1991.

Denna standard fastlägger ett ramverk kring IRDS-problematiken. En viktig ingrediens är den beskrivna distinktionen mellan olika typer av modeller som är aktuella och som måste kunna hållas isär. Triad-rapporterna K1 och K2 ger utförliga redogörelser. Denna standard har en aktuell och intressant allmängiltighet som referensdokument.

IRDS Services Interface, ISO/IEC 10728, 1992.

Presenterar en gränssnittsspecifikation som knappast kan förväntas få något nämnvärt intresse eller genomslag.

Pågående aktiviteter är primärt inriktade mot revideringar och utvidgningar av nämnda standarder. För en utförligare redogörelse hänvisas till Triad Rapporterna K 9 och K10.

6.3 ANSI/IRDS

Arbetet bedrivs inom en kommitté med beteckningen ANSI/X3H4. Den har varit aktiv under ett tiotal år och under denna period åstadkommit bland annat nedanstående. Det bör nämnas att arbetsinsatserna i allmänhet varit både mer omfattande och intensivare än motsvarande under ISO/SC21/WG3.

IRDS, X3.138, 1988

Detta var den först framtagna standarden inom IRDS-området. Som sådan blev den ett referensdokument. Standarden definierar grundläggande egenskaper hos ett tänkt IRDS. Bl a definieras den rekommenderade, E-R-baserade begreppsapparaten för formulering av metainformation.

IRDS Services Interface, X3.185, 1992

Innehåller gränssnittsspecifikationer och fyller på så vis samma syfte som ISO/IEC 10728 men baserat på E-R-modellens förutsättningar.

IRDS Export/Import File Format, X3.195, 1991

Definierar struktur och syntax för överföring av beskrivningsinformation, exempelvis mellan Case-verktyg och repository. Standarden har samma syfte som motsvarande CDIF-standard.

Dessa tre standarder anses ha bidragit till idéutvecklingen inom IRDS-området. Däremot har inte leverantörer inspirerats till anpassning (förutom Infospan).

Till senare resultat hör
- en specificerad referensmodell
- formulering av ett ramverk för representation, integrering och översättning av och mellan olika fristående representationsspråk och scheman
- definition av en alternativ, logikbaserad begreppsapparat
- definition av ett nytt gränssnitt baserat på en objektorienterad syn

samtliga beskrivna i Technical Reports.

Det intressanta med närmandet till objektorienterad specifikation är att det innebär en påtaglig modernisering av det sätt på vilket IRDS-problematiken attackeras, något som också bidragit till de beslut ANSI/X3H4 nyligen tagit. Mer om detta i nästa avsnitt.

6.4 Dagsläge, närmaste framtiden

Förutom aktiviteterna inom ISO och ANSI arbetar flera andra organisationer inom eller tangerar repositoryområdet.

OMG (Object Management Group) arbetar med lösningar för processintegration (processobjekt) i sina CORBA-specifikationer (CORBA = Common Object Request Broker Architecture). Se vidare Triad rapport K28.

Inom CDIF definieras överföringsprotokoll och metamodeller (det tillämpningsspecifika schemat).

Standardiseringssträvanden inom dbms-området täcker fler och fler av de tidigare repository-specifika egenskaperna. Dit hör behörighet, vyer, kanske versioner, ja det mesta faktiskt.

Den enda reella kandidaten till en integrerad helhetsspecifikation av en repositorymiljö är i dagsläget PCTE (Portable Common Tools Environment). PCTE är en gränssnittsspecifikation för kombinerad data- och processintegration med syfte att erbjuda en öppen miljö. Se Triad rapport K 22.

PCTE, som har sitt ursprung i ett Esprit-projekt för ett tiotal år sedan, har också på senare tid blivit alltmer uppmärksammat och accepterat. Bland annat finns nu ett konkret samarbete mellan OMG och PCTE. Den befintliga PCTE-specifikationen genomlöper en så kallad "fast-track" (snabb-behandling) till internationell standard inom ISO. Välbesökta konferenser anordnas. USAs Department of Defense har rekommenderat PCTE-baserade produkter. IBMs nya satsning på ett distribuerat repository är PCTE-baserat o s v. Produkter kan i framtiden i stor utsträckning antas bli PCTE-orienterade.

Många anser att PCTE dock behöver moderniseras för att vinna allmän acceptans. Bland annat hoppas man att det nyligen initierade samarbetet med OMG ska ge PCTE en mer objektorienterad profil. I samma anda orienteras nu ANSI/X3H4 helt mot PCTE. De har nyligen beslutat att inte längre ta fram egna IRDS-standarder. Istället kommer man att satsa all energi på att stödja framtagning av nästa objektorienterade version av PCTE inom ISO i syfte att åstadkomma en modern standard med bredast möjliga förankring.

Vi kan alltså konstatera att de olika drivkrafterna inom området börjar bli mer medvetna om varandras existens. Viljan till samarbete ökar vilket brukar vara ett tecken på att området håller på att "sätta sig" och att marknaden börjar ställa krav. Detta kräver i sin tur att den högst begränsade tillgången på kunniga personer inte onödigtvis splittras upp under mer eller mindre överlappande insatser. Just nu verkar de flesta samlas kring vidareutveckling av PCTE.

7 Frågetecken

Repositiryidén omges fortfarande av många oklarheter. Några av dessa är

- Vilken information ska hanteras i repositoryt? Vem bedömer och efter vilka kriterier? Informationen i repositoryt ska ju både samlas in och underhållas. Vem bedömer lönsamheten och hur? Går det att göra enkla kostnads/intäktskalkyler för att övertyga sig själv, beslutsfattare, ekonomiansvariga eller måste vi lita till vår intuition?

- Är det alldeles givet att det centrala, samordnade repositoryt är realistiskt eller ens produktivt? Vem tar ansvar för innehållet? Klarar vi av att hantera komplexiteten hos innehållet? Hur samordnas uppdateringar? Hur sprids kunskap om innehållet och dess uppdateringar? Hur löses intressekonflikter? Kanske är kompromisser i form av självständiga lokala repositories eventuellt kompletterade med endast den mest vitala eller uppenbart gemensamma informationen samlad i ett centralt repository, en mer realistisk ansats? Hur bör i så fall denna samverkan mellan lokalt och centralt repository lösas? Erfarenhet saknas. Idéutveckling kvarstår.

- Hur kan vi vara säkra på att informationen i repositoryt hanteras rätt när kommunikationen alltid går via Case-verktyg? Vilka krav bör vi ställa på "varudeklaration" av verktygen?

- Beroendet till respektive CASE-verktygs schema har upphört. Istället tenderar vi få ett beroende till repository-leverantörernas mer generella scheman. Är beroendeproblemet löst eller har det bara flyttats ett steg längre från användaren?

- Har det smakliga begreppet "återanvändbarhet" analyserats tillräckligt? Hur avgränsa repository-element för återanvändbarhet? Hur beskriva dess egenskaper? Hur finna dem? Ansvar? Ägarskap? Ska återanvändning ske genom kopiering eller referens? I det senare fallet skapas beroendeförhållanden som måste kunna hanteras. I förra fallet tappar man samordningselementet så fort kopieringen skett.

- Kommer repositoryn att bli sannt aktiva? Vilka förutsättningar krävs? Vilka blir konsekvenserna?

- Hur undvika att data i ett repository degraderas, aldrig tas bort. Innehållet måste alltid hållas som en god avspegling av den verklighet som modelleras.

- Ska kopior kunna hanteras, exempelvis för lokala projekt. Hur kontrolleras? Hur återföras? Låsning?

- Hur sprida och underhålla en nödvändig kunskap om hantering, effekter, regler, ... för alla berörda?

- Klarar vi av att hantera komplexiteten hos repositoryt? Vem har överblicken, kan se alla konsekvenser och beroenden?

- Blir vi slavar under repository-kunskapen? Hamnar makten hos den repositorykunnige istället för hos verksamhetskunnige?

- Ett repositoryschema är ju under konstant utveckling. Antag att ett standardiseringsorgans rekommenderade schema modifieras. Vilka blir konsekvenserna hos verktyg och leverantörer? Vilka datorstöd behövs för att stödja anpassningen? Antag motsatsen, d v s att scheman standardiseras så långsamt att en organisation känner sig tvingad att göra egna anpassningar. Vad blir konsekvenserna i existerande och framtida informationsutbyten?

- I och med fokuseringen på återanvändbarhet och stringenta utvecklingsmodeller ökar också kravet på utvecklare att ändra sina attityder från "konst" till "ingenjörsmässighet". Etablerar man inte rimlig och genomtänkt arbetsmetodik kommer sannolikt repository som idé inte att kunna förverkligas.

8 Slutsatser

Stordatorlösningens bristande framgång får inte tolkas som repositoryts generella oförmåga eller grava fel i grundidén. Repositoryt behövs och kommer att behövas än mer framöver, men antagligen i mer verksamhetsanpassade former. Erfarenheterna av LAN-baserad utveckling och pågående intensiva utveckling av teknik och funktion kring heterogena miljöer (OMG bl a) talar för distribuerade lösningar.

Den tekniska aspekten är en sida av problemet, den hanterade informationen en annan. Det verkar vara mycket svårt att få acceptans för ett standardiserat repositoryschema. Ännu svårare är det att nå rimlig täckning för många behov. Höggradig komplexitet och svåröverblickbarhet bidrar. Verkligheten tycks dessutom röra sig snabbare än modellen orkar hinna med. Dessa bekymmer kommer vi aldrig ifrån. Kanske är därför mer lokala, projekt- eller affärsområdesbaserade repositoryn det enda realistiska, möjligtvis kombinerade med viss samordning av verksamhetsvital information. Trender inom området "business objects" kan komma att skapa nya möjligheter.

Kanske förblir därmed integrerade C-Case-verktyg en vision?

Skillnaden mellan ett dbms och ett repository kommer att bli allt mindre markerat.

Håll ett vakande öga på händelseutvecklingen kring PCTE.

I alla händelser befinner vi oss i ett startskede. Samtliga inblandade, såväl repository-leverantörer som brukare behöver lära och anpassa. Helhetsperspektivet på hanterad information är synnerligen komplext. Vi bör bereda oss på en lång upplärnings- och mognadsprocess – på ett spännande 90-tal.

TIDIGARE UTGIVNA PUBLIKATIONER AV TRIADGRUPPEN

Verksamhetskrav på informationsadministration

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

Modellering

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

Kunskapsförmedling

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

Uttagssystem

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

Katalogprinciper

K 1: IRDS
K 2: IRDS Modeller och modellnivåer
K 3: Koppling begreppsmodell – relationsmodell
K 4: IBM:s Repository Manager – en Introduktion
K 5: IBM:s Repository Manager: Datamodelleringsbegreppen
K 6: IBM:s Repository Manager: Begreppsmodellering i Information Model
K 7: IBM Repository Manager: Attribut- och värdemodellering i Enterprise Submodel
K 8: Navigering i Repository
K 9: TRIAD Newsletter – IRDS inom ISO. Dagsläget
K 10: TRIAD Newsletter – ISO/IRDS. Händelseutvecklingen 91/92
K 11: Samverkan mellan resurskataloger – visioner eller behov
K 12: AD/Cycle I Information Model – Processer och och informationsflöden mellan processer
K 13: AD/Cycle I Information Model – Info Flows inom Processmodellen
K 14: AD/Cycle I Information Model – Relationsdatabasmodellering
K 15: AD/Cycle I Information Model – Härledningsspecifikationer i begreppsmodellen
K 16: IA-prototyp
K 17: Repository AD/Cycle – International Users Group
K 18: RAD-konferensen i Chicago, 1992
K 19: Vad händer inom ANSI-IRDS?
K 20: Information Warehouse – vad är det?
K 21: CDIF – en översikt
K 22: PCTE – en översikt
K 23: XLII – en öppen och flexibel utvecklingsmiljö
K 24: Hybris IA/DA – En IA-prototyp vid Telia
K 25: Introduktion till GDMO-standarderna
K 26: OpenODB
K 27: ANSI/X3H7 "Object Information Management"
K 28: Object Management Group
K 29: Översättning av modelldata
K 30: Objektorienterade ansatser inom ANSI/IRDS

KORT OM TRIAD

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

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

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

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