Institutionen för Programvaruteknik och datavetenskap
DVC 001 Examensarbete 10 poäng C-nivå
Informationsteknologi programmet våren 2002

Applikationsintegrering

-En analys av metoder och teknik

Författare: Noradin Amini
Leif Erixon

Handledare: Bengt Carlsson

Abstract

In the contemporary world of information technology you find a multitude of applications and systems covering a broad spectrum of areas of need in different companies. One effect of this multitude of programs is the difficulty to make them exchange information with each other or to collaborate, since they are developed by different programming languages for different platforms, with different standards and different data formats.

Our aim with this work is to describe how it is possible to tie these programs together to make them actually communicate with each other in order to exchange information, share their native methods and also to become a part of the overall business processes. In this integration task you will, among other things, find different levels of application integration such as data level, method level, application interface level and user interface level integration.

Application integration also involves hardware components, called middleware, that facilitate the physical connection between applications. There is a range of different middleware products offered today on the market. The functionalities of those products varies greatly depending on which technology or technologies they are built around and which vendor they come from.

In order to make a connection to a real life situation we have made up a company with a need of integration. By trying to choose a solution for this company we discuss *what* to integrate, data, methods etc, and the *technical solution,* middleware product, which might be useful to integrate the different kind of applications in our imaginary company.

Finally we have come up with some conclusions in our work. These are both of a more general art and also a conclusion specific to our case study.

Sammanfattning

Inom IT-området finns idag mängder av applikationer och system som täcker ett brett spektrum att behovsområden inom olika företag. En effekt av denna mångfald av programvaror inom ett företag är svårigheten att få dem att att byta information med varandra eller att samarbeta, eftersom de är utvecklade med olika programmeringsspråk för olika plattformar, med olika standards och olika dataformat.

Vårt arbete går ut på att ge en beskrivning av hur man kan få dessa applikationer och system att bryggas samman så att de faktiskt kommunicerar med varandra för att byta information, dela de metoder som finns och även ingå i större integrerade affärsprocesser. Vi talar bl.a. om integrering på olika nivåer i applikationerna som t ex datanivå, metodnivå, applikationsgränssnittsnivå och användargränssnittnivå.

En integrering kräver också en teknisk mellanvara, en s.k. ”middleware”, för att möjliggöra integreringen rent tekniskt. Här finns vitt skilda tekniker och produkter att tillgå. Funktionaliteten i dessa middleware varierar stort beroende på vilken eller vilka olika tekniker de bygger på och vilken tillverkare de kommer ifrån.

För att anknyta till det praktiska integreringsarbetet beskriver vi ett tänkt fall som vi försöker välja en integreringslösning för. Genom denna lösning vill vi välja både *vad* som skall integreras (data, metoder osv.) och *vilken teknik* som kan användas för att integrera det tänkta företagets olika applikationer. Detta beskriver vi i diskussionsdelen.

Slutligen har vi dragit en del slutsatser av vårt arbete. Dessa slutsatser är både allmänna och fallspecfika.

1. Inledning

Vilka IT-projekt är i fokus under det kommande året? Computer Sweden ställde frågan till ett antal svenska IT-chefer i början av 2001. Svaren visade på ett svalt intresse för stora affärssystem, men däremot ett stort intresse för att knyta ihop gamla system med mer moderna. ”Nyckeln är så kallad mellanvara, det vill säga mjukvara som kopplar ihop system och gör informationen åtkomlig genom ett gemensamt gränssnitt”.[1] Kenneth Bolinder, IT-chef på koncernen Assidomän säger: ”Inom koncernen har vi drygt 25 olika affärssystem. Istället för att byta alla system mot ett kopplar vi ihop befintliga system med mellanvara”.

En fördel som nämns är att genom att köpa mindre komponenter och bygga ihop dem till en helhet får man bättre avgränsade och mer hanterbara projekt.

Ett annat exempel är Electrolux IT-solutions. Vd:n Stephan Carlquist räknar i år med att driva ett stort antal projekt i Europa och USA och en majoritet av de projekten handlar om att koppla samman olika system med hjälp av mellanvara. Att införa stora affärssystem tar för lång tid. IT-solutions skulle hjälpa en kund att knyta ihop tre olika lokala affärssystem i tre länder, vilket tog två månader. I normal fall hade det tagit två år att ersätta tre affärssystem med ett enda.

Exemplen ovan belyser ämnesområdet för vårt arbete, nämligen integrering av applikationer och system. EAI, Enterprise Application Integrering, är det begrepp vi använder för att beskriva denna ”nya” IT-sektor. Integrering som sådan är inte nytt. Men så mycket har hänt inom områden under de senaste fem åren att vi kan tala om en ny period inom informationsteknologin.

Tony Brown[2] menar att vi kan dela in IT-historien i tre tydliga vågor. 70-talet präglades av automatiserad databehandling med hjälp av stordatorer. Perioden kallas ”*Centraliserad dataverksamhet*”. Den andra vågen kom under 80-talet med introduktionen av PCn och client/server-teknik, relationsdatabaser och nätverk. Detta var en period av ”*Distribuerad Dataverksamhet*”. Stordatorn var fortfarande viktig för att stödja en snabbt växande informationsbaserad ekonomi vilken krävde mer datorkraft. En mängd olika system utvecklades för att stödja företagens olika behov. Vårt nuvarande IT-landskap växte fram, ett landskap som beskrivs som en skärgård med en mängd isolerade öar, utan förbindelse med varandra.

Vi befinner oss nu i den tredje vågen, vilken började under 90-talet. Erp-systemen, dvs. affärssystemen, introducerades som en följd av det management-tänkande som rådde. Ett radikalt ny process-centrerad styrning skulle göra företaget mer effektiv, då alla i företaget skulle använda samma system och procedurer. Likaså fick Internet sitt genombrott och kopplade samman en mängd människor med en mängd information.

[1] Byttner, 2001

[2] Brown, 2000

Denna tredje våg kallas ”*Inkluderande dataverksamhet*” (Inclusive Computing). Inkluderande dataverksamhet är en rad teknologier som gör relevant information omedelbart tillgänglig för alla (anställda, kunder och leverantörer), genom att använda integrerade affärsprocesser inom och mellan företag. [3]

Enligt IDC Research förväntas denna integreringsmarknad bli den viktigaste och snabbast växande IT-sektorn under de närmaste tre till fem åren. Marknaden för programlicenser kommer enligt deras beräkningar att öka från fem miljarder år 2000 till närmare 21 miljarden år 2005 vilket betyder en årlig tillväxt på över 30%. Jämför detta med den beräknade tillväxten på övriga IT-tjänster på 11% under samma period. [4]

Samtidigt är eai-området komplicerat och svårt att få grepp om för många. Teknikerna är många och utvecklas snabbt, alla leverantörer påstår sig vara marknadsledande, många överord används i beskrivningen av produkterna om vad de kan åstadkomma, olika begrepp används vårdslöst, utan att man definierar vad man menar. Till och med begreppet EAI har så många infallsvinklar att det är en utmaning att ge det en entydig definition.
Den stora utmaningen ligger naturligtvis inte i att definiera och beskriva olika EAI-lösningar på ett bra sätt, utan att praktiskt genomföra lösningarna. En artikel nämner i förbigående att 88% av integreringsprojekten misslyckas, en uppgift som gavs i aug 2001.[5] Det ger en antydan om komplexiteten.

1.1 Metod

Vårt arbete består av en genomgång av aktuell litteratur och material som finns publicerat på Internet. Litteraturlistan kan synas mager, eftersom detta område är nytt och ständigt förändras, vilket gör att det finns få böcker att tillgå. Material på nätet finns det mer av, oftast kortare artiklar, vilket ger en fragmenterad kunskap.

Vårt arbete blir därmed en genomgång och analys av det material vi hittat och en egen diskussion/reflektion över olika lösningar.
För att ha en praktisk tillämpning att luta oss emot beskriver vi ett företag som har ett integreringsbehov, företaget Iridium AB.

1.2 Frågeställningar

Eftersom vi som författare inte hade någon förkunskap om integrering när vi tog oss an ämnet, hade vi många frågor som vi sökt svar på. Några frågor som vi utgår från är:

- Hur kan jag praktiskt länka samman applikationer och system som är byggda som slutna, ”ointagliga” system? Detta är ju något som tidigare varit omöjligt.

[3] Brown, 2000
[4] IDC, 2001
[5] Pollock, 2002

- Finns det någon teknologi som är mer lämplig än övriga för att skapa integrering mellan olika applikationer/system?

1.3 Avgränsningar

En integreringslösning behöver dels hantera hur man tekniskt löser sammanlänkningen av systemen. Tekniken ska möjliggöra överföring av data mellan olika applikationer.
Dels ska lösningen hantera hur man skapar förutsättningar för en rätt integrering av informationen. Informationsintegrering handlar om att bestämma vilken data som ska finnas tillgänglig för de olika applikationerna. [6]

En integreringskonsult berättar att hennes jobb ”är till 20 procent teknik och till 80 procent verksamhet. Det svåra med integrering ligger sällan i tekniken, utan för det mesta i att hitta skillnader i de system som ska integreras”, dvs informationsintegreringen.
Ofta beror skillnaderna på att företagets olika verksamhetsområden under årens lopp har utvecklat egna system och egna affärsprocesser. Dessa måste man ofta förstå i detalj för att kunna föreslå de förändringar som behövs för att företaget ska få sina system att samverka. Som exempel på olikheter nämner hon att begreppet ”kund” definieras på olika sätt i ett ekonomisystem och i ett säljstödsystem.[7]

I vårt arbete koncentrerar vi oss på applikationsintegreringen och avgränsar oss från informationsintegreringen, ett nog så intressant område.

Ett annat intressant område är hur man skapar applikationer som redan i utvecklingsarbetet anpassas för en framtida integrering. Drömmen är att kunna skapa applikationer som i stort sett är ”plug-and-play-färdiga”. Viktiga punkter i det sammanhanget är arkitekturfrågor, att skapa ett separat integreringslager, utnyttjandet av standards osv. Vi låter den infallsvinkeln vara, för någon annan att fördjupa sig i. Vi utgår alltså från redan befintliga applikationer och info-system, med de inbyggda svagheter och begränsningar som finns där sedan tidigare, dvs. applikationer som inte skapats för integrering.

1.4 Mål

Vårt mål är att ge en aktuell belysning av ämnesområdet applikationsintegrering, genom en sammanställning av aktuellt material, samt att utifrån ett praktiskt exempel diskutera olika val av integreringslösningar.

[6] EAI - ett integreringskoncept, 2002
[7] UtbildningsGuide, 2002

1.5 Tänkt scenario

För att göra arbetet mer konkret och, förhoppningsvis, mer lättsmält, ger vi en beskrivning av ett företag som har behov att integrera sina system. Exempelföretaget är en skapelse av oss själva. Den ger en bakgrund till våra diskussioner kring olika integreringslösningar och utgör också en utgångspunkt till vår diskussion i slutet av arbetet.

Iridium AB är ett företag inom livsmedelsbranschen med ett brett sortiment och med ett ganska stort distributionsnät i Mellansverige med omkring 3000 butiker. Varorna kommer från ett 50-tal leverantörer. Huvudkontoret är i Stockholm. Nu har Iridium nyligen köpt upp ett konkursföretag i Göteborg, Olssons HB, ett medelstort produktionsföretag med ett sortiment som passar mycket bra in i Iridiums profil. Tanken är att Olssons HB ska återuppta och utveckla sin produktion. Samtidigt blir Olssons HB bas för försäljning och distribution av Iridiums hela sortiment i Göteborgsregionen.

Målet är nu att utveckla ett väl fungerande IT-system för verksamheten. Vad som finns är följande: På Iridiums kontor i Stockholm finns ett affärssystem, Movex från Intentia, som ligger på en AS400 dator, dvs en minidator från IBM. Systemet har några år på nacken men fungerar bra och är väl inarbetat. Systemet är kopplat till en Oracle databasserver. Det finns inga direkta kopplingar till andra system, förutom att underleverantörer kan gå in i systemet via webben och se aktuellt sortiment, priser och produktnyheter.

Olssons HB installerade i slutet av 80-talet en stordator med ett system för produktionsstyrning, materialplanering och lager. Där finns också ett administrativt system för order, bokföring, löner och kundregister, i Windowsmiljö. Databasserver är en SQL Server7.

Ambitionen är nu att kunna integrera de befintliga systemen hos de båda företagen och likaså göra en närmare integreringskoppling till övriga leverantörer och till kunderna. Är det möjligt och vad skulle det innebära? Hur kan man på bästa sätt utnyttja de befintliga systemen? Att byta ut något eller några system har man redan utvärderat som ointressant i denna situation p.g.a. ekonomiska aspekter. Återanvändning är vad som gäller.

2. Integrering

2.1 Historik

Datasystemen från datorålderns barndom på 60- och 70-talet var enkla i sin uppbyggnad och i sin funktionalitet, med ett primärt syfte att ta över repetitiva uppgifter. ADB-system skapades för automatisering av databehandling. ”Någon tanke på integrering av gemensamma data fanns över huvud taget inte. Det enda målet var att föra över manuella arbetsuppgifter till datorn”.[8]

På 80-talet började många företag förstå värdet i och nödvändigheten av integrering av olika applikationer. Man använde de medel som fanns till buds på den tiden, vilket kunde innebära att man körde batch-uppdateringar på nätter eller helger, skickade FTP-filöverföringar eller ändrade koden i programmen.[9]

Det var en verklig utmaning för IT-personalen att försöka designa om redan implementerade applikationer för att få dem att samverka. Senare lärde man sig att koppla samman system genom att använda traditionella point-to-point middleware.

Point-to-point-integrering innebär att dela information mellan ett enskilt käll-system och ett enskilt mål-system. Integreringen var svårt att hantera, behövde ständigt dyra ändringar men var det enda alternativet innan andra tekniska lösningar kom.

Behovet av integrering växte fram utifrån misslyckandet hos de flesta erp-system-lösningar att ge en fullständig funktionell ersättning till de traditionella systemen, främst stordatorerna. Kostnaden och tiden som krävdes för att göra ett sådant byte ansågs av de flesta vara alldeles för högt för att vara försvarbart. Istället har de flesta företag insett att inom en överskådlig framtid kommer Erp-systemen att finnas sida vid sida med stordatorer och likaså tillsammans med e-business applikationer. Den integreringsteknik som nu växer fram ger en stabil struktur för att ta tillvara funktionaliteten hos stordatorerna och samtidigt använda spjutspetsteknik för affärskritiska samarbets- och kommunikationsbehov.[10]

Förutom affärssystem finns i företag idag en rad applikationer och system införskaffade för speciella avdelningar och verksamhetsområden. Nu inser många företag vinsterna med att länka samman många affärsprocesser och få systemen att samverka.[11]

[8] Inmon, 2002
[9] Linthicum, David S. 2002, e
[10] Holland, 2002
[11] UtbildningsGuide, 2002

2.2 Drivkrafter bakom integrering

Det finns idag många och starka drivkrafter bakom det ökade behovet av integrering. Vi har redan berört några av dem. När ett företag har investerat stora summor och lagt ner mycket tid i ett egenutvecklat system är det inte alltid möjligt eller lämpligt att byta ut det. Det är många gånger inte ekonomiskt försvarbart att byta ut det för att investera i nya system. Därför måste man hitta vägar för att kunna flytta information mellan de olika systemen.

Tidsfaktorn är också viktig. Att skapa en ny systemlösning tar för lång tid. Tiderna för implementering måste minskas. Därför återanvänds nu systemen istället för att man börjar från scratch.

Storföretagen har stora problem med vildvuxna systemfloror. Dels används olika system och dels används lokala varianter av samma system. Dessa måste man se till att de fungerar tillsammans. [12]

Företagen fokuserar ofta på kundbehoven för att öka försäljningen och en bra kundservice/kundinformation kräver en bred samling av affärsprocesser, vilka inkluderar även de slutna systemen såsom stordatorer.

Kundservice inkluderar även e-handeln, att kunderna själva utför enklare tjänster, som ex bank- och postärenden, att kunderna själva söker information om företaget och dess tjänster osv. Detta skapar ett starkt behov av exponering av ett företags system mot webben och denna exponering kräver ofta en integrering av de bakomliggande affärs-systemen.

Uppköp och sammanslagningar. Förmågan att införliva och utnyttja resurser som införskaffas via uppköp och sammanslagningar är mycket viktig och detta kräver massiva integreringsinsatser. Företag ser kombinationen av nya och gamla tillgångar i form av IT-system som ett sätt att bygga konkurrensfördelar och att få avkastning på investerat kapital.

Integrering av leverantörsledet, dvs. business to business-integrering. Det finns ett stort behov av att arbeta nära samman med sina leverantörer i en realtidsintegrering för att få bättre kontroll över leverans av tjänster/produkter till sina kunder.

Kravet att integrera olika system, som många gånger är som isolerade öar inom ett företag, har aldrig varit större. Inte heller har det varit så viktigt med integrering för att företaget ska kunna behålla olika konkurrensfördelar. Integrering har blivit ett måste.
Det finns uppgifter från Gartner Group om att en normal systemavdelning inom ett större företag idag lägger uppåt 40% av sin årliga IT-budget på att bygga och upprätthålla integreringslösningar.

[12] EAI - ett integreringskoncept, 2002

2.3 Fördelar och nackdelar med integrering

Integrering av IT-system krävs om man vill kunna använda data från olika datakällor. Detta ska kunna göras utan dyrbar och oflexibel point-to-point-integrering. En sådan integrering leder till lägre direkta IT-kostnader men även till andra fördelar, [13] t ex :

- Möjlighet att välja den applikation som bäst passar tillämpning och användare (best-of-breed) och slippa ett beroende av en (affärssystems)leverantör.
- Möjlighet att förlänga livslängden på befintliga (och väl fungerande) system
- Möjlighet att eliminera manuellt arbete
- Flexibilitet att snabbt svara på förändringar på marknaden genom att kunna byta ut eller lägga till vissa applikationer
- Kunna dra slutsatser ur den statistik som uppstår av meddelandehanteringen, t ex att kunna följa upp hur många orderavbokningsmeddelanden som skickats
- På sikt få fram en gemensam datamodell
- Underlätta vid uppköp och sammanslagning av företag
- Möjligheter att minimera ”dubbel” inmatning av information

Nackdelar:

- De flesta lösningar som säljs idag är mycket stora och dyra
- Man underskattar grovt det arbete som krävs för att få ihop datamodellen, all integrering kräver en genomtänkt verksamhets- och informationsmodell som kan överföras till en beständig datamodell
- Det är inget engångsarbete utan måste ske kontinuerligt
- Få företag lyckas nå hela vägen fram till en fungerande datamodell

Konsekvenserna av dåligt integrerade system kan bli förödande. Som i de flesta systeminföranden på företagsövergripande nivå är det därför att rekommendera att starta med så få system som möjligt för att successivt bygga på miljön med fler system och applikationer. Det är viktigt att tänka igenom vilka framtida egenskaper företaget ska ha. Ska det gå mot tät integrering mellan enheter eller mot en flexibel organisation för att kunna omstrukturera snabbt efter marknadens villkor? Detta kan bland annat få konsekvenser på hur hårt man väljer att integrera applikationerna i en verksamhet eller hur stor man väljer att göra datamodellen. [14]

[13] Ward, 2000
[14] EAI - ett integreringskoncept, 2002

2.4 Definition av EAI

Förkortningen EAI står för *Enterprise Application Integrering* vilket kan översättas med ”integrering av ett företags applikationer och system”. Linthicum menar att ”det finns lika många definitioner av EAI som det finns analytiker och leverantörer”, vilket kan vara nog så förvirrande. Vi stannar vid Linthicums definition nedan, vilken är kortfattad och övergripande.

”EAI möjliggör delning av data och processer mellan godtycklig applikation eller datakälla i ett företag. Detta ska dessutom kunna göras utan omfattande ändringar i applikationer och datastrukturer”.[15] Utvecklingen av begreppet EAI har gått från en snävare definition med tyngdpunkt på integrering av affärssystem, till en mer allomfattande definition av hela integreringsområdet.

Alla leverantörer av middleware och integreringslösningar sätter numera beteckningen EAI på sina produkter. EAI har blivit ett ”inneord”. Det finns en diskussion om huruvida processtyrning[16] ingår i EAI eller inte. Vi ser det som den högsta abstraktionsnivån inom eai-området, vilket torde vara det vanligaste synsättet. Eftersom EAI är ett så stort område med mängder av produkter, tekniker och infallsvinklar behöver man göra den gripbar genom någon form av gruppering eller strukturering.

Vi tänker oss följande arbetsgång i vårt arbete:

Vad ska integreringen innefatta?
System och applikationer inom företaget eller ska även system utanför brandväggen integreras?

Vad är det som ska integreras?
Data säkerligen, men ska även processer integreras, dvs olika funktioner som nu ligger utspridda i olika applikationer och system? Finns det mer man kan integrera? Vi tar hjälp av Hurwitz Group´s analys av EAI-marknaden.

Hur kommer man åt de data och de processer som ska integreras?
Många, speciellt äldre, system är vad man ibland kallar ”informationssilos”, helt slutna mot yttervärlden. Detta gäller speciellt stordatorer. Men även de flesta affärssystemen och egenutvecklade systemen saknar en naturlig ingång. De är först nu som behovet av integrering med andra system finns med i systemutvecklarnas tänkande när de skapar systemen.
Vi behöver alltså hitta integreringspunkter och för detta ändamål delar vi in applikationerna i integreringsnivåer.

[15] Linthicum, 2002, e
[16] se vidare sid 14 under Affärsprocessintegrering

Vilken teknik kan vi använda för att integrera?
Vi går igenom olika tekniska lösningar, från de mer grundläggande teknikerna till mer omfattande helhetslösningar. Denna marknad av middleware-produkter är stor, ständigt föränderlig. Marknaden är ”hypad”, dvs leverantörerna använder många överord om produkterna. Vi gör en uppdelning av olika grundtyper av middleware.

2.5 Vad ska integreringen omfatta?

Integrering kan appliceras på två olika områden: applikationer och system innanför brandväggarna, eller en integrering som innefattar även andra enheter som finns utanför brandväggarna. [17]

Intra-enterprise integrering

Detta innebär integrering av olika system/applikationer inom ett företag eller en organisation över ett intranät, dvs. innanför brandväggarna.

Inter-enterprise integrering

E-business som det också kallas dvs. integrering av ett affärssammanhang som innefattar ett företag och dess leverantörer (B2B, Business to Business) och/eller kunder (B2C, Business to Customer) över ett internet.

Tidigare B2B använde traditionell teknik som ex EDI (electronic data interchange) som flyttade information mellan företag via stora batch-överföringar. Idag frågar man mer och mer efter realtidslösningar och internetbaserad teknik som ex message brokers och applikation servers som utnyttjar internet, liksom nya standards för utbyte av data som ex XML. Både B2B och B2C exponerar information som finns inom organisationen till människor utanför organisationen. Om ett företag vill skapa en webbhandel behövs först en integrering av systemen internt innan de kan utgöra grund för en webbaserad försäljning eller informationsutbyte.

2.6 Vad är det som ska integreras?

Hurwitz Group försöker svara på den frågan genom att dela upp EAI-marknaden i fem segment, vilka står för olika integreringsbehov. [18]

[17] Linthicum, 2000, a
[18] Ren, 2002

Plattformsintegrering

Detta ger förbindelse mellan olika hårdvaru- och operativsystem. De tekniker som ger plattformsintegrering är messaging (budskapsförmedling), ORBs (Object Request Brokers) och RPCs (Remote Procedure Calls), vilka beskrivs under avsnitt 4.7

Alla komplexa EAI-lösningar kräver olika typer av kommunikation, inklusive en-till-många-messaging (budskapsförmedling) som skapas genom publish and subscribe mekanismer. Detta är något som alla integreringslösningar kräver: messaging, ORBs, RPC, publish & subscribe.

Sedan finns en rad tilläggsfunktioner; ex logik för hur man kopplar in sig på varje applikation, antingen genom kodning eller genom förkodade applikationsadapters.
Dessutom behövs funktioner som tar hand om skillnader i datarepresentation i de olika systemen. Detta kan kodas manuellt eller så kan man använda data-översättnings och omvandlingsprodukter. (dataintegrering & applikationsintegrering).

Du behöver även definiera logiken för att bestämma när och vart budskap ska skickas (routas). Detta kan du göra genom manuell kodning eller med hjälp av en Message Broker (applikationsintegrering). Om affärslösningen kräver integrering av nuvarande funktionalitet med ny funktionalitet, måste du ta hand om integreringen av ny funktionalitet (komponentintegrering).

Dataintegrering

Dataintegreringsprodukter finns i två olika kategorier: Den första inkluderar databas-gateways som ger SQL-access till olika typer av datakällor. Dessa är synkrona dataaccessprodukter och kräver att utvecklarna har kunskap om den underliggande databasens uppbyggnad (schema).

Den andra kategorin produkter innehåller redskap som hämtar ut, förändrar, flyttar och laddar data. (ETML redskap; Extract, Transform, Move, Load). Dessa redskap är vanligtvis batch eller point-in-time lösningar, lämpliga för den initiala inmatningen till ett Datawarehouse eller stora batch-överföringar. Dataintegreringsredskap hämtar och laddar data direkt, och går därmed förbi applikationslogiken.[19] Många av dessa redskap skapades först för att bygga Datawarehouses. Många ETML-leverantörer utökar sina lösningar med messaging-stöd. Dessa menar att deras produkter har utomordentlig kapacitet för metadatahantering och rening av data.

Några menar sig också ha stöd för mer komplexa typer av förändringar. Dessa redskap importerar metadata från många källor, inklusive relationsdatabaser, standardsystem och COBOL copy books, och har ett grafiskt användargränssnitt (GUI) för att man grafiskt

[19] Se avsnitt 3.5 som beskriver olika integreringsnivåer. Här talar vi om integrering på datanivån.

ska kunna dra relationer mellan system. Lösningen är ett antal system-till-system datakartor som redskapet hanterar.

Varje gång ett nytt system läggs till måste den mappas mot alla övriga system som den är integrerad med. Förändringar i någon applikation påverkar alla andra system som den är integrerad med.

För att kunna skapa dataintegrering måste man välja standardformat. Dessa stöder delandet och fördelandet av information och affärsdata. Dessa standards inkluderar COM+/DCOM, CORBA, EDI, JavaRMI, XML. [20]

Komponentintegrering

Komponetintegrering gör att ny funktionalitet på ett lätt sätt kan kombineras med affärssystem (erp-system) och client/server- och stordatorapplikationer. Applikations-servers ger denna funktionalitet, likaså har de funktioner för att ge säkerhet, transaktioner, lastbalansering och failover, förbindelsepool, sessionshantering, dataaccess till relations- och icke-relationsdatakällor. Dessa tjänster är också nödvändiga för integreringslösningar.

Applikationsintegrering

Det är ett ramverk för en samling tekniker som tillsammans ger en nästintill reatidsintegrering. Ramverket inkluderar

- Plattformsintegreringsteknik;
- Händelseintegrering med hjälp av message brokers vilken ger översättning (translation) av data;
- Förändring (transformation) och routing av data utifrån regler; och applikationsinterface-integrering vilket sker genom applikationsadapters till SAP, Peoplesoft och /eller andra ledande Erp-system. [21]

Integreringsramverken skapar mervärde genom att göra komplexiteten mer abstrakt beträffande att skapa, hantera och ändra integreringslösningen.

Några leverantörer levererar allt som nämnts ovan. Andra levererar vissa delar (som ex översättning och ändring eller adapters), vissa gör det genom samarbetspartners. Den största fördelen som applikationsintegrerings-frameworks ger är snabbare tid till marknaden genom färdiga adapters och återanvändbar integrerings-infrastruktur.

---

[20] EAI Overview, 2002

[21] SAPs R/3-system är marknadsledande inom affärssystem-marknaden, övriga stora aktörer är PeopleSoft och Baan.

Dataöversättnings- och förändringsredskapen som finns inom detta segment är annorlunda än de som finns i dataintegreringssegmentet genom att de är byggda för realtid istället för batchkörning. Vissa har också en effektivare, mer anpassningsbar metod för förändring genom att mappa varje applikation till en enda gällande (canonical) form. När en applikation ändras behöver bara mappningen mot den gällande formen ändras, medan alla andra applikationer inte berörs.

Applikationsintegrering används för B2B integrering, implementering av CRM (customer relationship management) system som är integrerade med ett företags back-end-applikationer, webbintegrering, och att bygga webbsites som utnyttjar många affärssystem. Kundintegreringsutveckling kan också bli nödvändigt, speciellt när man integrerar en stordatorapplikation med en ny ERP-applikation.

Affärsprocessintegrering

har den högsta abstraktionsnivån och anpassningsgraden när det gäller EAI. Dessa lösningar ger affärsfolket möjlighet att definiera, titta på och förändra affärsprocesserna genom ett grafiskt interface som ger en modell av hela affärsflödet i företaget.

Affärsprocessintegrering kräver alla de övriga underliggande typerna av integrering. Vissa leverantörer säljer bara processintegreringslösningar, och deras produkter placeras på toppen av andra EAI-ramverk; andra säljer en helhetslösning.[22]

2.7 Hur kommer man åt de data och de processer som ska integreras?

Både intra- och interenterprise integrering kan indelas i följande typer: [23]

Datanivå integrering

Handlar om att dela information på databasnivån, att flytta data mellan olika dataförvaringsplatser och på så sätt integrera applikationer. Det finns två varianter av lösningar här. Den första är en enkel, grundläggande kopieringsfunktion som oftast finns inbyggd i olika databaser för att kunna flytta information mellan databaser så länge som de har samma tabelluppbyggnad på alla databaser som utbyter information.

Den andra varianten innebär förutom kopiering även förändring av data för att passa olika typer av databaser, med olika uppbyggnad av tabeller och för olika leverantörers produkter. Genom att man hanterar applikationsinformation på datanivå behöver man inte

[22] Ren, 2002
[23] Linthicum, 2000, a

ändra någon kod i varken käll- eller målapplikationen. Detta minskar riskerna, förenklar hela processen och gör att metoden är relativt billig.

Det finns ännu en variant på att integrera databaser, vilken kallas för Federated database. Istället för att kopiera data mellan databaser presenterar denna modell en enda ”virtuell” databas. Systemutvecklarna använder denna databas som en integreringspunkt och når genom detta interface vilken databas som helst i de olika systemen.[24]

Applicationsinterface-nivå integrering

Denna typ av EAI är mest användbar för standardsystem som SAP, PeopleSoft, Baan, vilka alla har interface in till sin affärslogik och data, men på mycket olika sätt. Utvecklarna använder dessa interface, tar fram data, placerar den i ett format som kan förstås av målapplikationen och för över informationen.

Genom att använda applikationsinterfacen kan utvecklarna sammanföra många program och låta dem dela både affärslogiken och information. Begränsningarna är de specifika drag och funktioner som interfacen har.

Metodnivå integrering

är att dela affärslogiken inom ett företag eller mellan företag. Exempelvis kan metoden för att uppdatera en kunddatabas kommas åt från ett flertal program genom att använda en gemensam metod, som ex finns på en delad applikationsserver eller genom distribuerade objekt. Genom att dela metoder blir applikationer nära sammankopplade och därmed nära integrerade.

Det finns två sätt att göra det på: Antingen kan man skapa ett antal programmetoder som delas av flera och som finns på en delad fysisk server, tex en applikationsserver eller en TP-monitor (transaction processing monitor). Eller så kan metoder som redan finns inom en applikation delas genom att man använder distribuerad metoddelnings-teknologi såsom distribuerade objekt.

Användarinterfacenivå integrering

Här används användargränssnittet för att skapa integrering (även känd som screen scraping). Tekniken används oftast för att komma åt mainframes (stordatorer), vilka många gånger inte har någon anknytningspunkt till databas- eller logiknivån.

Denna typ av integrering anses av många som ett primitivt och instabilt sätt, men den har använts under många år och man har också löst många frågor kring kapacitet,

[24] Linthicum, 2002, d

tillförlitlighet och skalbarhet. Även om det inte är en önskvärd lösning är detta ibland den enda lösningen och därmed en viktig sådan.

Informationsportal integrering

Mycket populär idag pga webben. Här integreras applikationer genom att man presenterar information från många olika applikationer inom samma användarinterface, vanligtvis en web-browser. Därmed undviker man komplexiteten och kostnaderna med en traditionell back-end integrering.

2.8 Integrering av mainframes med andra applikationer

I över trettio år har det utvecklats mainframes-baserade applikationer. Dessa applikationer har utgjort kärnan i större företags affärsprocesser. Dessa applikationer är beroende av för företagen viktiga data som lagras i mainframes.[25]

Man skulle kunna fråga sig varför inte migrera från mainframes och skaffa modernare affärssystem. Men det finns skäll till att inte göra det. Rent praktiskt vet vi att det fortfarande finns över 20 000 mainframes runt om i världen som innehåller för företagen väldigt viktig data. Vi vet dessutom att mainframes klarar av att utföra över 10 000 transaktioner per sekund. Så man kan svara att det är därför man inte bestämmer att lämna bort dessa framgångsrika arbetshästar.[26]

För att kunna dra nytta av mainframes kraft väljer man idag att integrera dem företagets andra system som t ex erp-system. Meningen är att dra nytta av mainframes stora kapacitet för bearbetning av information. Det finns fem olika sätt att integrera mainframes och dessa fem kategorier kan grupperas ytterligare i två huvudgrupperingar, Invavise och non-invasive.[27] Invasive innebär att man måste gå in på låg nivå i själva host-applikationen och ändra i källkoden. Non-invasive metoden innebär däremot att mainframes hålls totalt utanför och lösningen utvecklas externt och sedan kopplas till mainframes. Följande är de fem sätt som integreringen baseras på:

Invasive

- Direkt åtkomst till mainframes-databasen.
- Använder API för att tillhandahålla åtkomst till mainframes-data.
- Bygga om applikationen för att tillhandahålla data och användargränssnitt i enlighet med företagets behov.

De flesta mainframes är utvecklade sedan länge tillbaka och är ofta Cobol-baserade och kompetenta personal med kunskaper i detta språk är inte lätt att få tag i idag. De flesta

[25] Jacada white paper
[26] Neil McHugh
[27] Propelis white paper

ovannämnda sätt kräver att man ska in i mainframes källkod och göra ändringar där. Det är också känd att de flesta mainframes saknar APIs så man måste utveckla dessa från scratch vilket varken är praktiskt eller ekonomiskt. Därför blir ofta svårt om inte omöjligt att hitta rätt kompetent personal som skulle kunna genomföra dessa. Som vi har sett är det ofta ingen som kan förstå koden strukturen på data i mainframes.[28]

Non-invasive

- **Screen scraping** är det ena av s k non-invasive tillvägagångssättet för åtkomst till mainframes. Detta sätt går ut på att fånga data på användargränssnittet. Det handlar om att definiera hur man ska få det rätt skärmen (screen), lokalisera den rätta informationen på skärmen, läsa den och slutningen bearbeta denna information.[29] Detta innebär att man skapar ett automatiserat program (en adapter) som agerar som en verklig användare, navigerar genom olika skärmar, efterliknar tangenttryckningar och läser skärmar in i minnet där informationen analyseras, omformateras och transporteras till antal lager på middleware för att slutligen skickas till målsystemet. Det finns klara fördelar med detta sätt. För det första behöver man inte ändra i källkoden på mainframes. För det andra är det lätt komma åt data genom s k skärmar. Dessutom är det riskabelt att ändra i mainframes kod vilket kan undvikas genom det här sätt.[30]

- **Kombination av screen scraping med modern, objekt-orienterad server-baserad teknik,** här använder man sig av s k. multi-tier applikation modellering. Enligt detta sätt kan man placera en server som en förmedlare mellan klienten och mainframes. Genom att tillämpa screen scraping kan man då få igång en dataström från mainframes till klintens användargränssnitt. Servern behandlar och modellerar och isolerar data fullständigt från klientapplikationen. Både erp-applikationer och webbaserade applikationer kan genom denna server få tillgång till data från mainframes.

  Fördelarna med detta sätt är klientappliktioner är helt isolerade från mainframes-logiken. Dessutom påverkar inte eventuella förändringar på mainframes klientapplikationerna och alla förändringar sker bara på den mellanliggande servern.

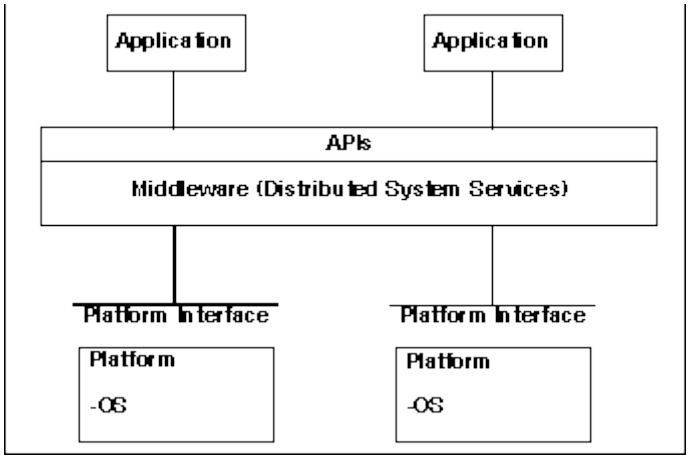
[28] Jacada white paper
[29] D. Linthicum, EAI, s. 81
[30] Propelis white paper

3. Vilken teknik används för integrering?

3.1 Begreppet Middleware

När vi talar om integreringsteknik kommer vi in på begreppet middleware som vanligtvis står för den tekniska lösningen av en integrering. Det finns en stor variation av middleware-produkter, från det enkla till det mycket komplexa.[31] Det kan vara minst sagt förvirrande när det mesta inom integreringsområdet, både ”det enkla” som exempelvis en RPC-funktion och en komplex lösning som Message Broker kallas både för en EAI-lösning och Middleware.

Inom middlewaregruppen finns några ”grundfunktioner” eller ”underliggande mekanismer”, nämligen message passing, RPC och ORBs. Dessa tekniker har funnits länge och utvecklades bl.a. för att möjliggöra flytten från centraliserade stordatorer till distribuerade system med client-server arkitektur. Teknikerna möjliggör också kommunikation mellan applikationer på helt olika plattformar. Teknikerna höjer abstraktionsnivån ett steg och döljer därmed komplexiteten hos det underliggande operativsystemet och nätverket.



Figur 3.1 Middleware ur ett logiskt perspektiv [32]

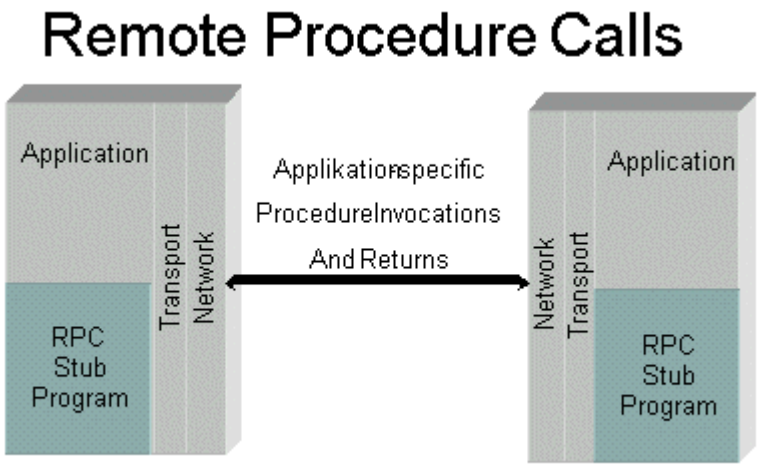
När vi talar om “en typ av EAI-lösning” som exempelvis Application Server eller Message Broker så består den dels av en teknisk hård- och mjukvara som kallas likadant, Applikation Server eller Message Broker, men också av mycket annat, vilket tillsammans skapar en integreringslösning. Detta ”annat” består t ex av adapters, APIs, olika arkitekturlösningar, affärslogik, informationsintegrering och mycket mer.

[31] Stallings, 1998
[32] www.sei.cmu.edu

Vi går i detta kapitel först igenom grundfunktionerna och därefter de grupper av middleware-produkter som är de vanligast förekommande, nämligen RPC:n databas-orienterad middleware, TP monitorer, Application servers, Message Brokers och Distribuerade objekt.

3.2 Remote Procedure Calls

Den första integreringslösningen som utvecklades var RPC som hade en s.k. point-to-point-funktion, vilket innebär att man skapar en direktlänk mellan två applikationer och på det sätt möjliggör utbyte av information.



Figur 3.2 Remote procedure Calls [33]

När programmet kompileras, som det visas i figur 3.2, skapas en lokal ”stubbe” dels hos klienten dels på servern. Dessa anknytningspunkter anropas när en fjärrfunktion behövs.[34]

RPC är en client/server infrastruktur som ger möjligheten att anropa en funktion i ett program och utföra den i ett annat program eller på en annan maskin. Denna anropsfunktion och exekveringen av ett program på en annan maskin är transparant för användarna. RPC har funnits sedan slutet av 70-talet.[35]

Den traditionella RPC:n är synkron vilket betyder att den anropande processen måste vänta tills den anropade processen har returnerat ett värde. Den anropande applikationen måste alltså stoppa exekveringen av programmet under tiden fram till den anropade fjärrapplikationen svarar.

[33] www.sei.cmu.edu
[34] Stallings, 1998
[35] www.sei.cmu.edu

Den synkrona RPC:n är en enkel mekanism som är lätt att förstå och lätt att programmera, eftersom dess beteende är förutsägbart. Men den förmår inte att utnyttja den parallellism som finns i distribuerade applikationer vilket begränsar dem och drar ner kapaciteten hos dem[36]. RPC är därmed inte heller lämpad för distribuerade objekt.
Asynkron RPC finns också men tillhör än så länge undantagen.[37]

En nackdel är att denna lösning bara kan binda samman två applikationer vilket innebär att ett företag med många olika applikationer kan behöva flera sådana point-to-pointlösningar. Detta skulle i sin tur öka komplexiteten inom företagets system betydligt samtidigt som det skulle försvåra kontrollen över systemet. Dessutom är det ett känt faktum att det krävs en hel del ändring i koden hos båda de applikationer som är sammankopplade med RPC.

Vidare är omfattningen av utbytet över nätet ganska mycket för att genomföra ett s k request vilket innebär en hög belastning för nätverket. Ett typiskt RPC-anrop kan kräva så mycket som 24 olika steg för att kunna slutföras.[38]

Samtidigt är det en styrka med RPC att dess synkrona funktion skyddar mot en överbelastning av nätverket, till skillnad mot MOMs asynkrona mekanism. RPC kan kombineras med MOM (Message-Oriented Middleware) och MOM i sin tur kan användas för asynkron processkörning. Därmed kommer man förbi vissa begränsningar.

RPC-infrastrukturer är implementerad inom Distributed Computing Environment, DCE, och inom Open Network Computing, ONC, som är utvecklad av Sunsoft, Inc. Dessa två RPC-implementationer dominerar den nuvarande Middlewaremarknaden.[39]

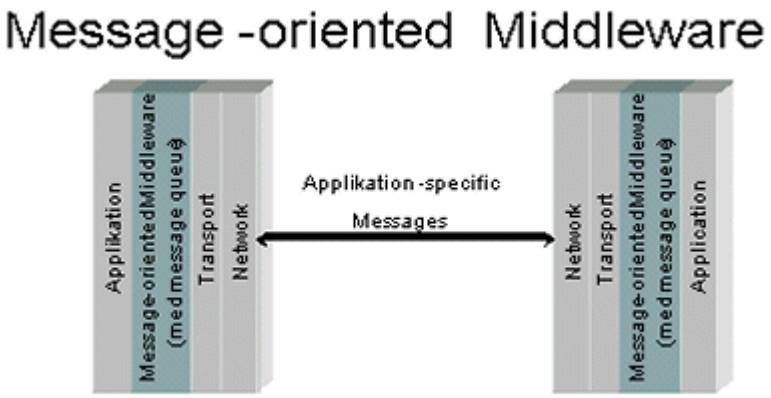
3.3 Message-Oriented Middleware (MOM)

Meddelande-orienterad mellanvara, MOM, möjliggör datautbyte mellan applikationer. Det hanterar leveransen av meddelanden, ger multiplattformstöd osv.

MOM skapades för komma tillrätta med de problem som fanns med RPC, framför allt dess synkrona natur. Även messaging används för att koppla ihop applikationer som kör på olika operativsystem, men här utnyttjas meddelandeköer, message queuing, för att temporärt lagra och hämta information, om destinationsapplikationen är upptagen eller inte uppkopplad just då. Det innebär att själva applikationerna kan fortsätta att med sin körning som vanligt. MOM tekniken är icke-blockerande till sin natur. MOM kan jämföras med email i det avseendet att det är asynkront och kräver att mottagaren av meddelandet ska tolka dess innebörd och på ett rätt sätt agera därefter. [40]

[36] Stallings, 1998
[37] www.sei.cmu.edu
[38] Linthicum, 2000, a
[39] www.sei.cmu.edu
[40] www.sei.cmu.edu

De meddelanden som här används för att kommunicera med andra applikationer är dessutom små vilket gör dem lätta att hantera. En annan fördel med denna middleware är att de kan skicka samma meddelande till flera fjärrapplikationer utan att behöva vänta på att dessa program ska vara igång, s k broad-casting.



Figur 3.3 Message oriented middleware [41]

Meddelandeorienterad middleware (MOM) som visas på figur 3.3 är en programvara som finns på båda sidorna av client/server-arkitekturen.

MOM-produkter började komma från mitten till sent 80-tal. Även fast RPC-baserad teknik kräver att en koppling upprättas mellan klient och server så tillåter messaging-tekniken att infrastrukturen hanterar meddelanden och köer, prioriterar dem osv.

Vissa MOM-produkter stöder transaktioner. De kan också köa och leverera en batch med transaktioner för ökad effektivitet. MOM kan också stödja säker leverans, lastbalansering och synkron och asynkron kommunikation. P.g.a. dessa egenskaper har organisationer som ex banker använt messaging och queuing som sin infrastruktur för att integrera sina applikationer, vare sig de är batch eller transactions.[42]

MOM i sig självt är inte tillräckligt för applikationsintegrering, främst därför att den visserligen låter applikationer utbyta meddelanden men den saknar funktioner för förvandling av data och schema och kan inte heller erbjuda s k ”intelligent routing”.

3.4 ORBs

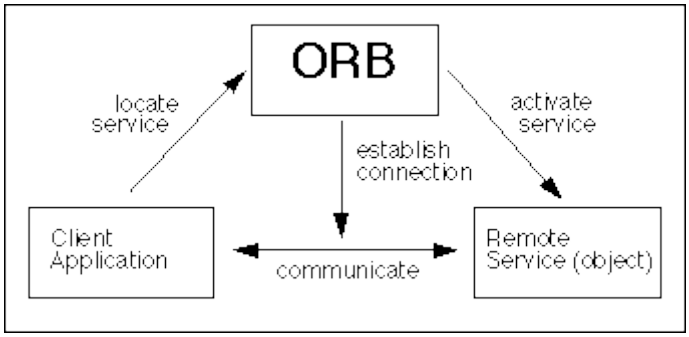
Objekt Request Brokers (ORBs), är middlewareteknik och produkter som hanterar kommunikation och datautbyte mellan objekt. Tekniken gör det möjligt att distribuera de

[41] www.sei.cmu.edu
[42] Zahavi, sid 380

objekt som utgör en applikation över olika typer av nätverk. Funktionerna hos ORB-tekniken är:

- Definition av interface
- Lokalisering och aktivering av fjärrobjekt
- Kommunikation mellan klienter och objekt

En objekt request broker fungerar ungefär som en telefonservice. Det erbjuder en samling tjänster och hjälper sedan till att upprätta en kommunikation, figur 4.4, mellan klienter och dessa tjänster.

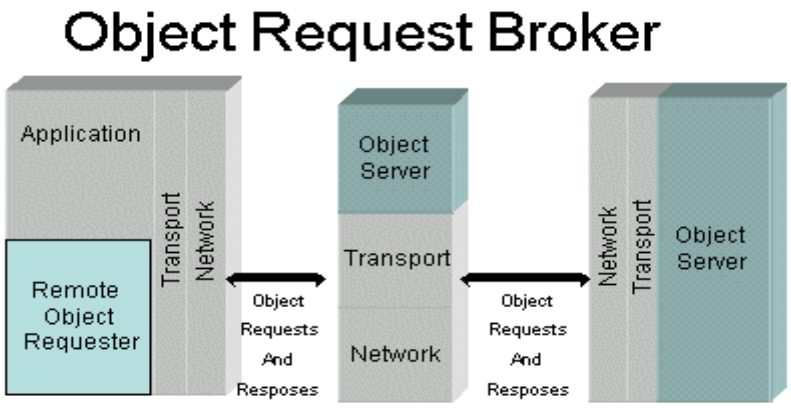


Figur 3.4 Object request broker[43]

ORB-tekniken utgör grunden för Distribuerade Objekt-produkter såsom CORBA och COM (se följande avsnitt).

Vi ser många exempel på hur olika tekniker blandas, eftersom en viss teknik sällan eller aldrig ensam möter alla de behov som finns. En sådan trend är att sammanföra MOM och ORB-teknikerna. Kombinationen av dessa produkter, som oftast kallas message request brokers (MRB), inkluderar funktioner som exempelvis förändring, transformation, av data och asynkron budskapsförmedling. Message Brokers, en MOM-produkt, kommer att dyka upp som en CORBA3-produkt (en ORB-produkt). [44]

[43] www.sei.cmu.edu
[44] Zahavi, 2000



Figur 3.5 Object Request Broker [45]

3.5 Distribuerade objekt

Distribuerade objekt betraktas som middleware eftersom de skapar inter-application kommunikation. Men de är också mekanismer för utveckling av applikationer genom att göra det tekniskt möjligt att dela metoder och de stöder därmed metodnivå-EAI. Distribuerade objekt är egentligen små program som använder standardinterface och standardprotokoll för att kommunicera med varandra.

Exempelvis kan en utvecklare skapa ett CORBA-kompatibelt distribuerat objekt som körs på en Unix-server och ett annat CORBA-kompatibelt objekt som körs på en NT-server. Eftersom båda objekten skapats genom att använda en standard, CORBA, och båda objekten använder ett standard kommunikations-protokoll (Internet Inter-ORB-protokoll, IIOP) kan de utbyta information och köra sina funktioner genom att anropa varandras metoder. [46]

Två typer av Distribuerade Objekt finns på marknaden: CORBA , skapad 1991, mer en standard än en teknologi, och COM, en standard från Microsoft, som inkluderar interface standards och kommunikations-protokoll.

CORBA är ett extra lager uppe på en RPC, vilken är en synkron överföring. Därför har inte CORBA den bästa prestanda vilket anses vara dess stora nackdel. [47]

3.6 Databas-orienterad middleware

Det är en middleware som möjliggör kommunikation med en databas, antingen om

[45] www.sei.cmu.edu
[46] Linthicum, 2000, e
[47] Linthicum, 2000, a

det är från en applikation eller från en annan databas. Denna middleware utvecklades för erbjuda en mekanism för att hämta information från antingen en lokal databas eller en fjärrdatabas. Databas-orienterad middleware fungerar med följande två grundläggande databastyper:

3.6.1Call-level interfaces

Call-level interfaces middleware möjliggör åtkomst till hur många databaser som helst genom ett väldefinierat gränssnitt. Dessa fungerar vanligtvis med relationsdatabaser. Ett exempel på Call-level interfaces är Microsofts Open Database Connectivity (ODBC). ODBC erbjuder också många samtidiga accessmöjligheter till samma gränssnitt. Ett annat exempel är Javasofts JDBC. Detta är ett standardgränssnitt som använder ett antal java-metoder för att underlätta åtkomst till många databaser.

3.6.2 Native database middleware

Denna typ av middleware kommer bara åt en viss databas genom åtkomstmekanismer tillhörande just denna databastyp. Nackdelen är att den är begränsad till bara en databastyp. Fördelen är att den ger ökad prestanda och åtkomst till alla lägre delar av en viss databastyp.

3.6.3 Databas Gateways

En annan databas-orienterad middleware är "gateways". Liksom andra typer i denna kategori av middleware är gateways' främsta uppgift att erbjuda förbindelse mellan *applikationer* och olika databaser. Dessa gateways är mycket användbara för att koppla samman applikationer med gamla och stora system som mainframes. De kan mappa dessa ålderdomliga databaser så att de ser ut mera som traditionella databaser och översätter förfrågningar(Queries) och information som kommer in och ut genom gateways.[48]

Gateways använder en API med ett enda gränssnitt för att komma olika typer av databaser som finns på olika plattformar. Gateways översätter SQL-anrop till ett standardformat som heter Format and Protocol (FAP) och detta blir den ordinarie förbindelsen mellan klienten och servern samt en länk mellan databasen och plattformen.[49] Gateways kan direkt översätta API-anrop till FAP och flytta förfrågningar till måldatabasen och även översätta förfrågningen så att måldatabasen och plattformen kan reagera på den.

[48] D. Linthicum, EAI, s. 195
[49] D. Linthicum EAI , s. 206

Det finns ett antal gateway-produkter på marknaden idag som t ex Information Builders' Enterprise Data Access/SQL (EDA/SQL), IBM's Distributed Relational Data Access (DRDA) och ISO/SAG's Remote Data Access(RDA).

3.7 Transactional middleware

Denna typ av middleware bygger på transaktioner. En transaktion är ett begränsat arbete eller en händelse med ett början och ett slut. Hela processen måste nå sitt slut innan målapplikationen uppdateras, dvs. det gamla ersätts med nya. När inte transaktionen slutet av någon anledning, exempelvis nätverksavbrott eller liknande, så berörs inte målapplikationen och ett nytt försök görs vid senare tillfälle.

Detta är den första av fyra egenskaper som en transaktion har. Man talar om ACID-test av transaktioner. Bokstäverna står för Atomic, Consistancy, Isolation, Durability.
Atomic betyder ”allt eller inget”. Transaktionen fullföljs helt eller inte alls. **Consistancy** betyder att systemet alltid befinner sig i ett stabilt tillstånd, vare sig den fullföljer en transaktion eller inte. **Isolation** är en transaktions förmåga att arbeta isolerat från andra transaktioner som körs på samma TP-monitor eller Applikationsserver. **Durability** betyder att transaktionen, när den har fullföljts och är färdig, överlever systemfel, datorhaverier och andra fel. Sammantaget är en transaktion säker. Även när saker och ting går snett tillåter en TP-monitor eller Applikationsserver inte att detta påverkar andra transaktioner, applikationer eller data.

Dessa middleware, som består av TP-monitorer och applikationsservers, gör ett bra jobb med att koordinera förflyttningen av informationen och metoddelningen mellan många olika resurser.
Trots att denna middleware erbjuder utmärkta möjligheter för metoddelning är den inte så bra på att dela information, vilket är det verkliga målet med applikationsintegreringen. Dessutom måste målapplikationernas källa ändras för att dra nytta av Transactional middleware.

3.7.1 TP-monitorer

TP-monitorer har funnits ett bra tag. Den äldsta, Tuxedo, är över 20 år gammal. De var en del av stordatormiljön, där de hanterade interaktionen mellan ointelligenta fjärrterminaler och applikationer på en stordator.

Idag talar man om “öppna” eller ”distribuerade” TP-monitorer. Nu är klienten inte en dum terminal utan en PC eller Mac som begär anknytning till värddatorn som idag lika ofta är en Unix-server, inte en stordator. Deras funktion har också ändrats till att omfatta mer än enbart transaktionshanterings-tjänster. Idag inkluderar de ofta även messaging, publish and subscribe, RPC och även interface mot object request brokers (ORBs).

TP-monitorer erbjuder en plats för applikationslogiken och är en mekanism som möjliggör kommunikation mellan två eller flera applikationer. Exempel på TP-monitorer är Tuxedo från BEA och MTS från Microsoft.

Fördelen med dessa TP-monitorer är att de delar upp en applikations metoder i små portioner, transaktioner. Sedan anropas dessa transaktioner för att utföra instruktioner från en användare eller ett annat fjärrsystem. Dessa små enheter, som utgör en transaktion, gör att det blir lättare att hantera och köra dem i TP-monitorsmiljön.

Prestandamässigt har TP-monitorer ett stort värde genom att de har en bra funktion när det gäller lastbelastning. När belastningen ökar griper Transaction Managern in och sätter igång fler serverprocesser och utnyttjar flera TP-monitorer för att ta hand om den ökade belastningen.

Att implementera TP-monitorer är tidskrävande. Många gånger är miljön sådan att det rör sig om integrering av hundratals stordatorer och system till en enhetlig miljö och att integrera dessa kan ta två eller tre år.[50]

3.7.2 Application Servers

Applikationsservers har sin styrka i metodnivåintegrering. Applikationsservern utgör en fysisk plats där man kör gemensam programkod och blir därmed också en plats där man samlar affärslogiken. Gemensamma metoder samlas i applikationsservern, där de kan anropas från olika klienter.

En applikationsserver kräver att komplexa applikationer delas in i mindre enheter, s.k. transaktioner. Utvecklaren skapar en serie med delade transaktioner som förläggs till applikationsservern där de kan anropas av olika klienter, noder eller applikationer. Transaktionerna kan ses som förprogrammerade funktioner, t ex att kommunicera med en applikationsresurs som exempelvis en databas.

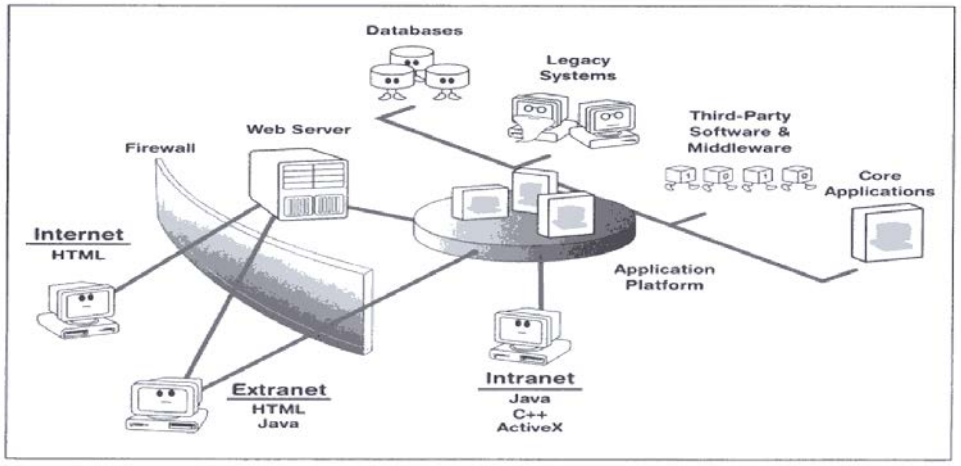
Applikationsservern kan routa transaktioner genom många olika system. Det är inte ovanligt att en applikationsserver binder samman stordatorer, NT-servrar, UNIX-servrar, filservrar och olika databaser.

Likaså arbetar man mot olika delade resurser såsom databaser, affärssystem eller köer. Integreringen sker alltså dels genom att man delar affärslogiken och dels att man integrerar bakomliggande applikationer och resurser.

Fördelen är att applikationsservern är hårt knuten till de applikationer och metoder den har kontakt med. Den affärslogik och de kontroller som är knutna till dessa delade metoder går inte att komma förbi.

[50] Gill, 2002

Svårigheten är att man blir tvungen att göra större förändringar i alla käll- och målapplikationer för att de ska kunna formas kring ett antal gemensamma tjänster. Det innebär en låg grad av flexibilitet. Om en ny applikation kommer till via ett uppköp måste även den anpassas till de gemensamma tjänsterna och kanske måste något justeras eller läggas till i dessa tjänster, vilket aldrig är en snabb och enkel process.[51]



Figur3.6 Applikationsserver är utmärkt för att binda samman system både innanför och utanför en brandvägg [52]

Applikationsservern *samlar affärslogiken* på ett ställe, dessutom kan användare av dessa servrar integrera och samordna olika resurser genom att använda *transaktionell semantik* (t ex starta, hämta information, flytta information, uppdatera information, stopp). Likaså kan de *visa information* från dessa resurser i webb browsers eller genom client/server-interface, figur 3.6.

En applikationsserver har också lastbalansering, trådpool, objektåtervinning, och förmågan att automatiskt återhämta sig från typiska systemfel.

Den stora fördelen med applikationsservrar är dess förmåga att köra multiplexering och hantera transaktioner, vilket ger en kraftig minskning av antalet uppkopplingar och den processbelastning som stora system lägger på en databas. Exempelvis kan en applikationsserver hantera över 1.000 klienter via omkring 50 uppkopplingar till en databasserver. Uppkopplingarna samutnyttjas alltså (trådar och processer) och om en uppkoppling överbelastas startas en ny uppkoppling. [53]

Applikationsservrar utgör inte bara en plats för applikationslogiken, de binder också samman många käll- och målapplikationer med hjälp av connectors/adapters vilka applikationsserver-leverantörerna tillhandahåller. En sådan backend-integrering är oftast en förutsättning för de webbaserade transaktioner som applikationsservrarna från början skapades för att kunna hantera. Applikationsservrar kan enkelt visa information från dessa

[51] Linthicum, 2002, d
[52] Linthicum, 2000, a
[53] Linthicum, 2000, b

resurser i webb browsers. Därför dominerar de idag marknaden när behoven är både att ha en webbaserad front och en back-end integrering på metod/process-nivå. Web-baserade säljsystem är givna kandidater för applikationsservrar.

Det finns idag applikationsservrar som är speciellt anpassade för webben, s.k. Web Applikation Servers. Även traditionell teknik som distribuerade objekt och TP-monitorer har utvecklats för att kunna användas mot webben, men Web Application Servers har flera fördelar. De är skapade för webutveckling. Därför förser de utvecklaren med många olika mekanismer för att bygga webbapplikationer genom att använda en applikationsserver. Dessutom är det mindre kostsamt att bygga applikationer med denna teknik.

Applikationsservrar är inte lika dyra, det finns många redskap datt använda och det går snabbare att utveckla.

Applikationsservrar är en middleware-produkt som använder andra middleware, exempelvis Message Brokers. De kommunicerar på många olika sätt, inklusive RPC[54] och MOM. Eftersom applikationsservrar är integrering inom en applikation har utvecklare möjlighet att välja och blanda olika middleware-lager och server-resurser för att möta kraven från ett integreringsbehov.

När är en applikationsserver en bra lösning? Ett integreringsbehov som behöver stödja transaktioner mellan applikationer, dela både logik och data, och där man är beredd att betala kostnaden för att ändra alla applikationer som är kopplade.

Ledande produkter inom området är BEA System´s WebLogic, IBM´s WebSphere och Netscape´s iPlanet. En ny produkt är Microsofts AppCenter Server.

Skillnader mot TP-monitorer:

TP-monitorer är dyra och kraftfulla verktyg för större företag som exempelvis banker, försäkringsbolag, flygbolag, kontokortsföretag. TP-monitorer har bättre prestanda och tillförlitlighet.

Applikationsservers erbjuder mer avancerade tjänster såsom Integrated Development Environment (IDE). De är också lättare att använda än traditionella TP-monitorer.

Många applikationsservers kan använda sig av komponentbaserad arkitektur och använder då företrädesvis EJB (Enterprise JavaBeans) som teknik vilken är en komponentmodell för JavaBeans på serversidan. EJB liknar mycket distribuerade objekt som COM och CORBA, åtminstone utifrån arkitektursynpunkt. EJB kan länkas samman för att skapa en distribuerad applikation. JavaBeans finns på applikationsservrar, vilka kan köra dem som transaktionskomponenter.

[54] en speciell variant av RPC för applikationsservrar vilken kallas transactional RPC eller TRPC (Linthicum, 2000, b)

3.8 Message brokers

Message brokers är i själva verket ”intelligenta” servrar som förmedlar meddelanden mellan två eller fler käll- eller målapplikationer. Dessa erbjuder en centraliserad infrastruktur, figur 3.7, för hantering av kommunikation och förmedling av meddelanden mellan olika system och applikationer.

Figur 3.7 Message brokers

Message brokers underlättar förflyttandet av informationen mellan två eller flera resurser, käll- eller målapplikationer och kan dessutom hantera olikheterna i applikationernas semantik och plattform.

Message brokers erbjuder även en mekanism för integrering av affärsprocesser oavsett vilken miljö de befinner sig i. Utöver detta kan Message brokers upprätta förbindelse med varje applikation och ”routa” informationen mellan dem med hjälp av flera middleware- och API-mekanismer. Message brokers är också kapabla att lagra affärsfunktioner som är byggda ovanpå redan existerande affärsfunktioner tillhörande den enhet som den är kopplad till. Message brokers erbjuder många andra tjänster också vilka kan kategoriseras som följande:

- Meddelande förvandling (Message transformation)
- Intelligent routing
- Rules processing
- Message warehousing
- Repository services
- Directory services
- Management
- Adapters

Det finns inte någon överenskommen standard för Message brokers på marknaden idag. Många utvecklare tillämpar egna lösningar med vitt skilda innehåll. Trots det kan det sägas att de flesta av dessa lösningar innehåller som åtminstone några funktioner såsom meddelande förvandling, regel motor(engine) och intelligent routing samt en del andra egenskaper.

3.8.1 Meddelande förvandling

Denna funktion känner till formatet på alla meddelanden som har förflyttats mellan applikationerna och förvandlar dessa meddelanden. Datan från meddelandet omstruktureras till ett nytt meddelande som blir förståeligt för mottagarapplikationerna. Meddelande förvandlingen består i sin tur av schemaförvandling och formatändring på data.

3.8.2 Intelligent routing

I en komplex miljö där flera applikationer är sammankopplade genom en Message broker måste det finnas mekanismer för hur meddelanden ska komma fram dit de ska. En sådan mekanism, Intelligent routing, är inbyggd i de flesta Message brokers vilken gör det möjligt för alla meddelanden att komma fram till sina målapplikationer.

När Message brokern får ett inkommande meddelande analyserar den det. Från vilket system kommer det? Finns det behov av förändring? Om så är fallet genomförs en förvandling och även andra affärsregler och tjänster tillämpas på meddelandet. Därefter routas det till rätt målapplikation. Allt detta sker omgående och så många som tusentals sådana operationer kan ske simultant.

3.8.3 Rule processing

Reglerna här är de restriktioner och bestämmelser som avgör meddelandenas förändring och härledande från en källapplikation till en eller flera målapplikationer. Utifrån dessa regler kan Message brokern avgöra vad den ska göra med ett nytt meddelande. Dessa regler bearbetas av en mekanism som kallas för ”rule engine”. Message brokern kombinerar denna rule engine och transformationsmekanismen i ett eget lager för att dynamiskt fatta beslut om förvandling och routing av meddelanden.

Ofta använder rule engine sig av traditionell boolesk logik (if, else och or) och ett högnivå-programmeringspråk för att skapa regler och anknyta händelser till varje regel som visar sig vara sant.

3.8.4 Message warehousing

Message warehousing är en databas som finns i Message brokern för att meddelandena som går genom Message brokern ska kunna lagras där. Detta bemöter kraven på sk message mining, message integrity, message archiving och auditing.

Message Mining innebär att message warehousing, figur 4.9, är en sken-data-warehouse som tillåter hämtning (extraherande) av affärs1data som kan ligga till grunden för olika beslut. Det är t ex möjligt att använda message warehouse för att bestämma antalet nya kunder och information om deras karaktäristiska särdrag som har körts genom message brokern.

Message integrity är en service som erbjuds av message warehouse för att garantera att meddelandetrafiken ska vara naturlig och persistent (ihållande). Om t ex servern går ner kan message warehouse agera som en ihållande och buffert eller kö för att lagra meddelanden som annars skulle gå förlorade.

Message archiving ger Message brokerns användare möjlighet att spara meddelande-trafiken i en arkiv för framtida bearbetning eller andra syften. Det kan t ex vara så att man i framtiden vill gå tillbaka och se på meddelandena under en viss tid eller dag. Denna funktion möjliggör kort och gott återställandet av meddelanden för analyser.

Auditing erbjuder möjligheten att kunna bedöma tillståndet för hela integreringslösningen och även erbjuda redskap att lösa problem som uppstått. Med Auditing kan man t ex bedöma storleken på meddelandetrafiken och dess belastning på systemet samt variationen på meddelandens innehåll.

3.8.5 Repository services

Denna funktion går ut på att fungera som en databas och samla information om käll- och målapplikationer. Denna information kan bestå av dataelement, inputs, outputs, inter-relationer bland applikationer. Även om dessa förvaringsplatser anses höra till applikationernas värld så har de ett självklart värde för applikationsintegreringen. Repository är en katalogliknande tjänst som inte bara har information om applikationers data och ”directory” utan även har mera avancerad information som metadata, meddelande-schema, säkerhet och ägandet tillhörande käll- och målapplikationer. Repositoryn är självklart i huvudkatalogen för hela applikationsintegreringen.

3.8.6 Directory services

Dirctory service agerar som en guide bland tusentals resurser som finns tillgängliga för applikationer och middleware. Message brokers behöver directory services för att

lokalisera, identifiera, använda och autorisera nätverksresurer applikationer och andra system. Dirctory service-funktion på message brokers ger utvecklarna möjlighet att bygga applikationer som kan på ett intelligent sät lokalisera andra resurser var som helst på nätverket. Dirctoryn vet var i nätverket sitter resurserna och kan hitta de på vägnar av applikationen. Exempelvis kan ett word program hitta en skrivare med hjälp av denna funktionalitet.

3.8.7 Management

Många Message brokers har en egen lager för management-arbetet. Denna lager tar hand om administrationen av problem domänen för applikationsintegreringen. Denna funktionalitet omfattar övervakning av meddelande-trafiken, meddelande-integriteten och samordningen av meddelande-distributionen bland målapplikationer. Det finns för få message brokers idag som har en komplet lösning för management-arbetet. Dl lösningar som finns att tillgå idag somBMC:s Patrol och Computer Associate's UniCenter är långt ifrån tillfredsställande när det gäller krav som ställs på sådana verktyg.

4. Adapters

Är kopplingen mellan applikationer/system och den EAI-lösning man bygger.
Adapters spelar en avgörande roll vid en integrering, skalbarhet och användarvänlighet är nyckelbegrepp. Helst bör leverantören erbjuda en adapter-utvecklingsprogram som eliminerar mycket av logiken - tabell-för tabell, fält-för-fält.

Det leverantörer kallar adapters är många gånger enbart sammankopplare (connectors), ett enkelt kommunikationsinterface ut eller in från ett system eller databas. Men en riktig adapter borde innehålla intelligens beträffande affärslogiken i käll- och målapplikationen.

4.1 Tunna adapters

Erbjudas av de flesta populära Message Brokers idag. I de flesta fall är den enkla API-wrappers, som levereras som en samling bibliotek, som mappar käll- eller målsystemets interface till ett gemensamt interface som stöd av message brokern. Man binder alltså en API samman med en annan API vilket påverkar prestanda men ökar inte funktionaliteten. En hel del programmering behöver fortfarande göras.

4.2 Tjocka adapters

Har mycket mjukvara och funktionalitet mellan message brokern och käll- eller målapplikationerna. Den tjocka adapterns abstraktionsnivå gör att förflyttandet eller metodanropen smärtfria. Eftersom abstraktionslagret och programvaran hanterar skillnaderna mellan applikationerna som ska integreras, är det i stort sett inget behov av programmering.

Detta abstraktionslager i adaptern döljer komplexiteten i käll- och målapplikationernas interface för användaren, som endast ser en grafisk representation av processen och metadatainformationen. I många fall kopplar användaren samman processer enbart via detta grafiska användarinterface utan att behöva ägna sig åt kodning.
Dessutom kan många abstraktionslager skapas kring de olika typerna av applikationer som ska integreras.

T ex skapas ett lager för ”vanliga” middlewaretjänster (som distribuerade objekt, budskapsorienterade middleware och transaktionsorienterade middleware). Ett annat lager skapas för standardsystem som SAP:s R/3, Baan och Peoplesoft. Ytterligare ett lager hanterar integreringen av relationsdatabas och icke-relationsdatabaser osv. Men komplexiteten bakom varje lager döljs för användaren.

4.3 Statiska adapters

Statiska adapters är vanligast idag. De måste kodas manuellt med vad som finns i käll- och målsystemet. Till exempel har en statisk adapter ingen mekanism för att förstå en databas´ uppbyggnad (schema) och måste därför konfigureras för hand för att kunna ta emot information från datakällan. Om sedan uppbyggnaden ändras kan denna typ av adapter inte uppdatera ändringarna.

4.4 Dynamiska adapters

Dynamiska adapters har förmågan att ”lära” sig om de system som den är kopplad till
Genom en upptäcksprocess som dessa adapters går igenom när de kopplas upp mot applikationerna eller datakällorna. Detta innebär exempelvis att läsa av databasens metadata (schema), information från (the repository), eller kanske källkoden, för att upptäcka struktur, innehåll och applikationssemantik i de uppkopplade systemen. Förutom att lära sig kan de även lära om om något har ändrats i systemen. Tex om attributnamnet på kundnummer har ändrats kommer det att upptäckas automatiskt av en dynamisk adapter.

Naturligtvis är det bästa valet en Message Broker med både tjocka och dynamiska adapters. Tyvärr har utvecklarna en bit kvar att gå innan de är framme vid vad som utlovats. Förutom dessa ”färdiga” adapters behöver en Message Broker-leverantör även kunna förse dig med möjligheten att själv bygga adapters för icke-standard program. Det innebär utvecklingsredskap, vanliga bibliotek, dokumentation och träning. [55]

Intelligenta adapters har stor betydelse för ekonomin kring ett integreringsprojekt. Aberdeen Group har beräknat att intelligenta adapters kan reducera implementationstiden för ett integreringsprojekt med så mycket som 70 % genom att minska behovet av applikationsexperter och kodning av anpassningar. [56]

4.5 Centraliserade och distribuerade adapters

Det finns två arkitekturer kring adapters: distribuerade och centraliserade. Centraliserade adapters körs på Message Brokern. Vanligtvis är dessa tunna adapters som bara binder Message Brokerns API till käll- eller målapplikationens API.
Distribuerade adapters däremot är tjocka adapters som finns både i Message Brokern och i käll- eller målapplikationen. Eftersom adaptern är uppdelad i två delar kan den bättre koordinera överföringar av information mellan Message Brokern och mål- eller källapplikationen. [57]

[55] Linthicum, 2002. c
[56] Traverse, 2001
[57] Linthicum, 2000, a

5. Diskussion

Låt oss nu gå tillbaka till vårt företag Iridium och deras ambition att utveckla sin verksamhet genom att integrera de system de har. De system som finns är av olika typ. Här finns ett stordatorsystem, ett affärssystem (erp-system) på IBMs minidatorplattform, client server-system och ett antal fristående applikationer på olika avdelningar. Alla dessa applikationer finns på olika plattformar och är skapade med olika programmeringsspråk (Cobol, C osv).

Integreringen på den lägre nivån, plattformsnivån, sker med hjälp av de ”basfunktioner” som vi gått igenom, såsom RPC, messaging och ORB. De mer sammansatta integreringslösningarna använder många eller alla dessa basfunktioner.

Vilken teknisk lösning ska företaget välja? All investering i och implementering av integreringslösningar bör utgå från följande frågor:

- Företagets integreringsbehov
- Hur mycket information som behöver flyttas
- Typen av information som behöver flyttas mellan systemen (ren data eller händelser och processtillstånd)
- Antalet system som behöver integreras [58]

Vi vill nu diskutera olika integreringslösningar och pröva deras lämplighet i vårt företag Iridium.

5.1 Datanivå-integrering

Stockholmskontoret har behov att ta del av den information som finns i Göteborg hos Olssons HB. Produktdatabasen, lagerinformation, information om produkter i arbete, kunddatabasen m.m. är av intresse. Bl.a. för att webbsidans information ska vara heltäckande behöver all information om alla produkter samlas i produktdatabasen i Stockholm.

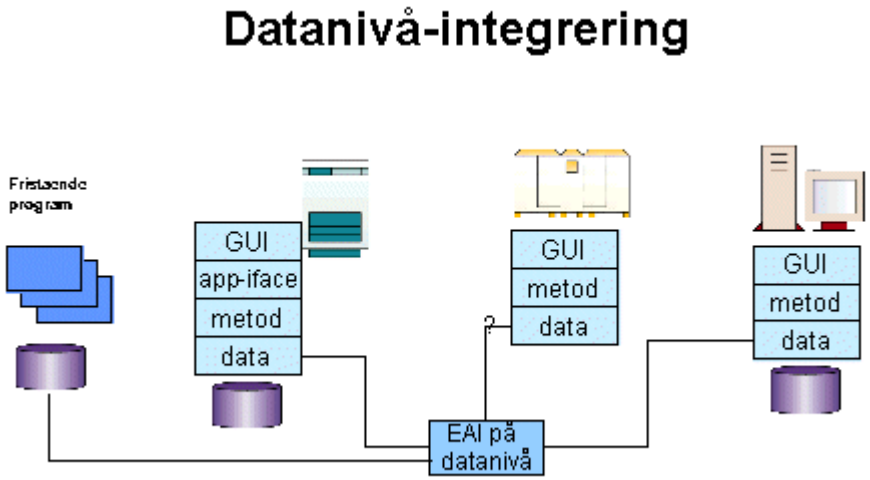
Eftersom Olssons i Göteborg blir säljbas för hela Iridiums sortiment behöver de naturligtvis en nära koppling till Stockholmskontorets datainformation. Styrelsen gör en bedömning att en integrering av databaserna är ett bra första steg och en förutsättning för det fortsatta arbetet. Man tänker sig göra en batchkörning varje natt under veckan för uppdateringar.

Lösningen är en databasintegrering. Databas-till-databas fungerar bäst när det räcker med att dela information. Till exempel när systemen bara behöver dela kundinformation och

[58] Linthicum, 2000, d

inte behöver dela metoder eller exempelvis bokföringskonton. Detta är billigt, det är en låg risk, det är utprovat och ger tillräckligt bra EAI när integreringsbehoven är lätta.

Ett problem som man inte lyckas lösa är att integrera den datainformation som finns i Göteborgskontorets stordator. Detta skulle eventuellt vara möjligt med någon av de mer avancerade dataintegreringslösningarna, men då kan man lika gärna satsa på en annan typ av lösning som en applikationsserver el. dyl.



Figur 5.1 Vid datanivå-integrering anknyter vi direkt till datanivån.

5.2 Metodnivå-integrering

Det är klart att Iridium skulle vinna på att göra en mer omfattande integrering än datanivå-integreringen. Även händelser och processtillstånd behöver integreras för att företaget ska kunna fullt ut dra nytta av en integrering. Ex när en order tas emot i Stockholm vilken berör produkter och produktionen i Göteborg bör en beställning eller produktionsorder gå direkt till Göteborgskontorets system. Förändringar i kunddatabaser eller lagersystem berör både Göteborgs och Stockholms verksamhet. Och så vidare, det finns mängder av kopplingar mellan systemen som ligger på metodnivån, i form av händelser och transaktioner.

Nu har företagsledningen även målet att integrera leverantörsledet och kundledet, dvs att skapa en e-businesslösning. Utifrån de ambitionerna behöver vi se på andra integrerings-lösningar, vi talar då om metodnivå-integrering.

Skulle en **TP-monitor**-lösning vara möjlig i vårt exempelföretag? Det beror på. Transaktionshantering kommer vi att behöva i den lösning vi väljer för att skapa automatiseringar av olika affärsflöden. Men en traditionell TP-monitor är en stor, ”tung” och dyrbar lösning som lämpar sig bra för deras ”traditionella” branchområden såsom

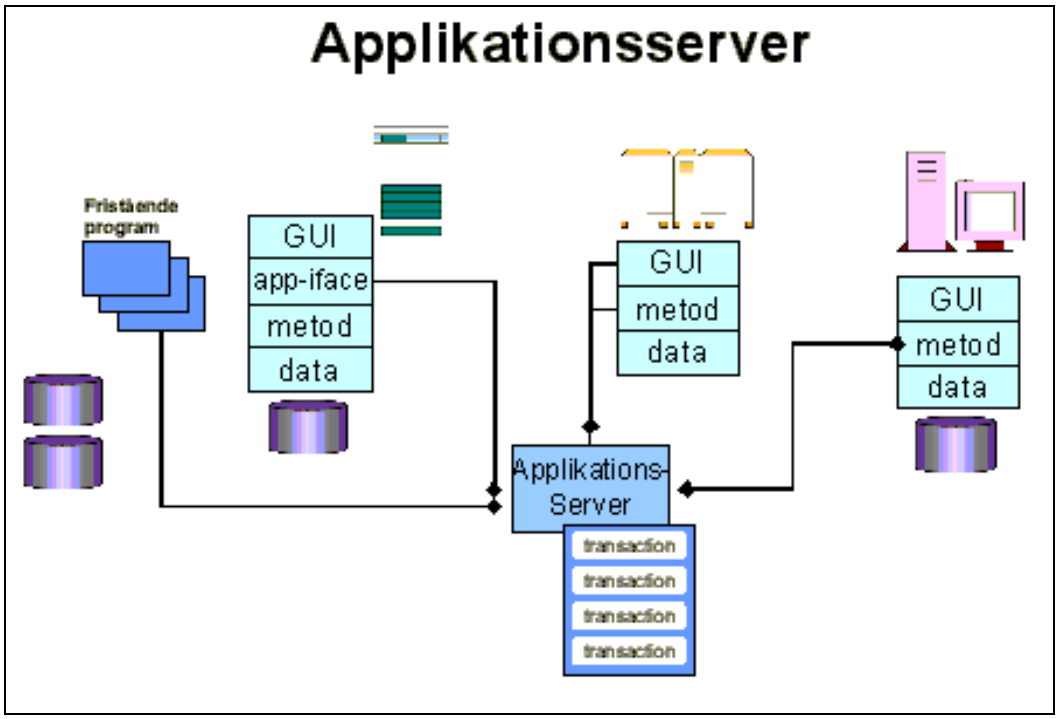
banker, försäkringsbolag, flygbolag, kontokortsföretag osv. men midre lämplig i vårt sammanhang. Däremot är applikationsservers som också bygger på transaktionshantering intressant, likaså andra lösningar som inkluderar transaktionshantering.

Är **distribuerade objekt** en bra lösning för oss? En CORBA-lösning är ett kraftfullt redskap för ett större företag med en komplicerad integrering. Även här kan vi ta en bank som exempel, där det gjorts stora investeringar i kraftfulla system samtidigt som nya tekniker och behov dyker upp som exempelvis kunders behov av självbetjäning via internet. Stordatorsystem, TP-monitorer, webservrar, MessageBrokers osv binds samman i en CORBA-lösning. Distribuerade objekt är alltså en effektiv lösning, men för vårt exempel anser vi att den är för dyr och krävande.

Om nu distribuerade objekt inte är aktuell för vår del så är distribuerade komponenter definitivt intressant i vårt sammanhang. Komponentbaserade lösningar, framför allt baserade på Enterprise JavaBeans, EJB, har kommit de senaste åren. Vi återkommer till detta när vi diskuterar applikationsservers.

Applikationsserver-lösning
Att välja en applikationsserver-lösning innebär att man gör en tight koppling mellan de olika applikationerna.
När företaget nu vill bygga en e-handelslösning behövs en integreringsplattform som kan kommunicera med alla de applikationer som finns inom och utanför företaget. En applikationsserver är lämplig när behoven är både att ha en web-baserad front och en back-end integrering på metod/process-nivå. När en beställning kommer in via webb-gränssnittet genererar den ett antal transaktioner. Exempelvis kan ordern lagras i en databas, en koll göras mot lagerdatabasen, beställningar gå ut, ev till leverantörer, en orderbekräftelse går till beställaren via email, inklusive preliminärt leveransdatum osv. Man strävar efter att ta bort manuella arbetsled så att man får en automatiserad orderhantering. Detsamma gäller produkter-i-arbete-förfrågningar, lagerförändringar, faktureringar m.m.

Det som känns mest attraktivt för Iridiums del som vi ser det är att bygga på en J2EE-plattform och att därmed utnyttja EJB (Enterprise JavaBeans) och JCA (J2EE Connector Architecture), dvs distribuerade komponenter och JCA-standarden som förser applikationsservern med ett enda interface mot de applikationer som finns i företaget. XML används som standard för dataformat och XSL blir tekniken som sköter om-mappning och förändring av datan.
En J2EE applikationsserver erbjuder skalbarhet, transaktionshantering och en miljö för administration av systemet.
Att utnyttja standardlösningar har stora fördelar. Marknaden för adapters som har JCA-standarden växer vilket gör att priserna sjunker och kunskapen på området ökar. Det blir också möjligt att fritt välja moduler/byggblock på den integreringsplattformen osv.



Figur 5.2 Applikationsserver ger transaktionshantering och samlar processer och affärslogik på ett centralt ställe.

5.3 Integreringslösning med message brokers

En strategisk övervägande vid ett integreringsprojekt är om man vill skapa en total integreringslösning eller om man vill integrera stegvis. Det skulle innebära att man först gör en delintegrering och sedan utvärderar det man gjort, drar lärdomar av projektet och sedan utvidgar integreringen, allt eftersom man bedömer behoven. Det minskar både risktagande och komplexiteten.

En datanivåintegrering skulle kunna vara ett första steg för Iridium, att samordna databaserna. Men vi påpekade tidigare (under punkt 5.1 om datanivåintegrering) att företaget skulle ha högre ambitioner med integrering än att bara uppdatera databaserna i batch-form med hjälp av t ex database-federation så bör man välja en annan teknik med andra finesser och mer omfattande funktionalitet. Iridium har högre ambitioner med sin integreringslösning och vi har nu sett att en applikationsserver är en intressant lösning. En annan lösning som är minst lika intressant är en Message broker.

Företaget vill naturligtvis integrera även sin stordator, mainframes, som finns i Göteborg, eftersom den innehåller många viktiga funktioner och data. Där finns förutom produktionsstyrning även uppgifter om lagret, prisuppgifter m.m. och dessa behöver uppdateras i realtid eller nära realtid. Det skulle annars medföra att man får tillgång till

gammal och irrelevant data vilket skulle innebära till exempel att man sålde saker som inte fanns i lager. En e-handelslösning kräver den typen av realtidsintegrering.

Det faktum att Iridium AB med sin dotterbolag i Göteborg Olsson HB har många olika applikationer och datakällor gör att det är av stort värde att försöka skapa en lösning som samtidigt som den omfattar alla komponenter, också gör det möjligt att reducera komplexiteten och underlätta konfigurationen och administrationen av integrerings-lösningen.

Låt oss då gå in på detaljerna och se vad den tekniska lösningen, dvs den middleware vi väljer, måste kunna erbjuda. Den ska kunna länka samman applikationer så de kan utbyta data med varandra. Den data som utbyts kan vara av olika format så den måste kunna känna igen formatet och ändra till ett format som accepteras av målapplikationer. Förbindelsen kan delvis gå över internet och då blir det extra viktigt med säkerhets-aspekterna. Integreringen måste också möjliggöra tillhandahållande av information via webben. Uppdateringen av informationen på webben ska ske i realtid eller nära realtid för att leverantörer ska kunna lita på att de har tillgång till den senaste informationen.

Given dessa villkor måste vi välja en middlewarelösning. Men eftersom vi har mainframes applikationer och andra datakällor inblandade blir problematiken lite krångligare. Det finns få möjligheter att integrera mainframes med andra system. Då mainframes saknar en egen API-gränssnitt, vilket de flesta erp-system har idag, blir det omöjligt att göra en direkt koppling via applikationsgränssnittet. Det måste göras på annat sätt. Det finns då två möjligheter. Det ena sättet är att gå in på låg nivå i mainframes källkod och antingen bygga en direkt koppling, skapa ett API, eller också bygga om själva mainframe-systemet så att det går som med andra applikationer att välja nivå för integreringen. Detta är alltså ett s k invasive sätt.[59]

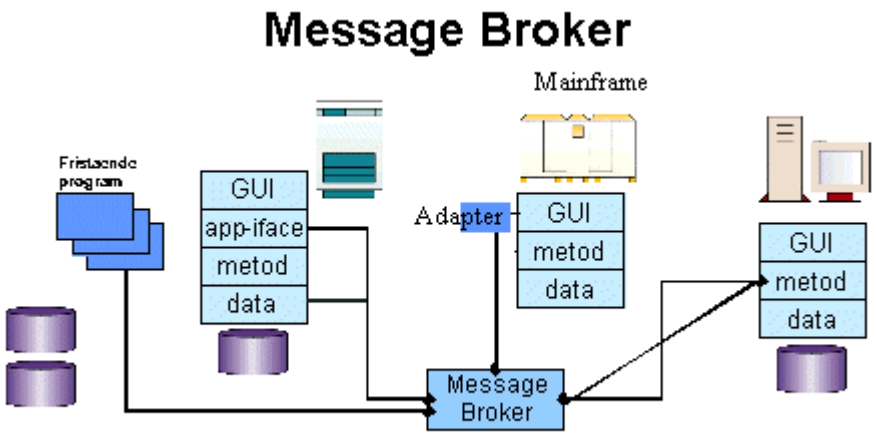
Det andra sättet går ut på att välja teknik utanför själva mainframes utan att gå in i källkoden. Då kan man genom s k "Screen scraping" fånga data från användar-GUI till mainframes. Detta sätt har utvecklats ytterligare så att man kan lägga till en länk mellan GUI och mainframe-hosten genom vilken man fångar data i en server, bearbetar det som sedan ska skickas till antingen applikationer eller en annan middleware för vidare bearbetning.

Genom att välja en anpassad adapter (server) för mainframes som kan koppla den till en middleware kan Iridium AB integrera sin mainframe-baserade databas i resten av infrastrukturen och dra nytta av dess kraft och styrka i sin verksamhet.

Nästa steg i vårt förslag är att lösa problemet med att Iridium har olika typer av applikationer dvs. formatet på data från applikationer skiljer sig från varandra så att målapplikationer inte förstår det och därmed inte kan använda den. Andra problem är t ex hur data ska transporteras, hur data ska hitta fram till den ena eller den andra målapplikationen dvs. routingen och dessutom hur vi ska gardera oss mot olika fel och

[59] Jacada white paper 2002

driftstörningar i systemen och nätverket, alltså säkerheten. Dessutom måste vi ha en länk mot webben.



Figur 5.3 Datanivå integrering med message broker

En message broker blir här ett bra val för att klara av dessa uppgifter. Message brokers erbjuder en central plats som alla företagets applikationer kan antingen kopplas till direkt eller genom andra anpassade middleware-produkter som i det här fallet en adapter för mainframe databasen, figur 5.3. Message brokers använder sig av meddelanden för att flytta information och dessutom har de en mekanism som heter "meddelande kö" som underlättar utväxlingen av data utan att applikationer behöver vänta för svar från målapplikationer. Message brokers kan erbjuda datautväxling i form av både en till en, en till flera och flera till flera.

Fördelen med att använda meddelandetekniken är att det lämpar sig speciellt för trafik över nätet då meddelanden är små och inte orsakar overhead och för mycket belastning på nätet. "En annan egenskap av message brokers är att de är av s k "non-invasive" typ, dvs. de applikationer som ska kopplas till message brokern behöver i många fall inte ändras i källkoden. I de fall det behövs ändringar så är mycket lite som ska ändras".[60] Detta medför att införandet av integreringen inte kräver extra resurser utifrån för programmering och anpassning av företagets applikationer.

Message brokers har ett lager som hanterar transformeringen av inkommande data. Där finns redan tidigare de format som inkopplade applikationer har på sin data. Data på väg in studeras genom att öppna dess schema. Då tar message brokern reda på dess format och dessutom tar de reda på vart den är på väg. När den förvandlat datas format för målapplikationen om så behövs, avgör den även den väg data ska ta.

Iridiums system finns både innanför och utanför brandväggen. Därför är det viktigt att den tekniska lösningen erbjuder även routing. Message brokers' routing funktion gör det

[60] D. Linthicum, EAI s.297

möjligt att data tar rätt väg och på så sätt minskar man flaskhalsar och eventuella stockningar på nätet samt att data faktiskt kan hitta fram till målapplikationen. "Detta är en unik funktion som message brokers har jämfört med mera traditionella meddelande orienterade middleware".[61]

Skalbarhet är en annan fördel som finns med när man tillgår message brokers. Det går att knyta många applikationer till en och samma middleware. Men det som är ännu bättre med message brokers är det går att koppla fler message brokers till varandra. Denna möjlighet öppnar vägen för framtida expansioner och utökning av systemet. Samtidigt som message brokers får en central plats i integreringen bidrar de till att minska komplexiteten i företagets datorsystem och underlättar hantering och administrationen av det.

Svagheten i message brokers består av att de saknar förmågan att ta sig an applikationers logik, dvs. de inbyggda metoderna i varje applikation, för att samla dem på en central plats och där kunna anropas av andra applikationer.[62] Detta gör att message brokers inte klarar av integrering på metodnivå på samma sätt som t ex applikation servers eller distribuerade objekt, vilka lämpar sig bättre för denna typ av integrering.

En annan sak som kan uppfattas som nackdel med message brokers är att denna typ av middleware är relativt dyr. Det beror förstås på att de för det första är ganska nya och för det andra att de har många funktioner att erbjuda. Vi tror ändå (utan att ha gjort någon närmare ekonomisk kalkyl) att Iridium AB skulle kunna klara av den kostnaden och därmed få en integreringslösning med många funktioner som passar för integreringen av företagets innehav av applikationer och datakällor. Lösningen gör att företaget kan förändra sina applikationer och system utan att behöva göra om integreringsarbetet. Man får en löst kopplad integreringslösning. Företaget kan också växa utan att för den sakens skull få problem med integreringen.

[61] Jane Griffin, 2002
[62] D. Linthicum, EAI, s. 76

6. Slutsatser

Vad kan vi dra för slutsatser? Att "utse en vinnare" för Iridiums del när det gäller integreringslösningar är inte lätt. Det var inte heller syftet med uppsatsen. Vi har använt Iridium som ett bollplank för att testa några olika lösningsförslag för att på så sätt bättre kunna förstå problemen kring integrering.

Om vi sammanfattar situationen för Iridium konstaterar vi bl.a. följande:

1. Det är relativt få system som ska integreras.
2. Man ser inga stora förändringsbehov inom den närmaste tiden av systemen (men här finns ändå en viss osäkerhet).
3. Ett tydligt behov är en kraftfull och hållbar webb-baserad e-businesslösning.
4. Man önskar helst bygga med standardtekniker, vilket har framtiden för sig.
5. Man vill om det är möjligt undvika alltför mycket ändring av kod.
6. Investeringskostnaden måste vara försvarbar.

Utifrån punkt 1, 2, 3, och eventuellt 6 skulle en applikationsserver vara lämplig. Punkt 5 förespråkar en Message Broker. Punkt 4 talar för J2EE, XML, JCA m.m.
Utifrån den kunskap vi besitter för dagen skulle vi föreslå att de tittar närmare på en produkt som BEAs WebLogic Applikation Server som är byggd på J2EE, använder sig av EJB, XML, JCA m.m., dvs standards som utvecklas starkt idag. Förslaget ger vi med det stora förbehållet att det inte finns en enda "rätt" produkt för en situation, speciellt när "kravspecifikationen" är så odetaljerad som i vårt fall.

Generellt gäller, som för all typ av systemutveckling, att det gäller att förstå problem-domänen och de affärsmål som man vill uppnå, utan att i förhand peka ut någon enskild teknik som den bästa för integrering. Många författare pekar på att en enskild produkt inte löser alla problem kring ett integreringsbehov, vilket är helt riktigt. Även leverantörerna har insett detta och bakar därför in mer och mer funktioner i sina produkter, vilket gör att kombinationsmöjligheterna mångfaldigas. Det verkar som om de renodlade middleware-produkternas tid snart är förbi.

Vi har sett att alla typer av middleware-produkter vi sett närmare på "sväller" på detta sätt. De flesta produkterna sägs klara allt eller i vart fall det mesta inom integreringsområdet. Det kan alltså behövas en uppmaning till försiktighet, se upp för överreklamering av enskilda produkter!

Att lyssna in oberoende analytiker blir därför viktigt. Intressant i sammanhanget är att oberoende testcentra dyker upp. I Computer Sweden kunde vi läsa om att en "oberoende konsult testar mellanvaror".[63] Ett svenskt företag, (Amodo), gör tester av olika mellanvaror, bl.a. den lösning vi föreslog, WebLogic-servern. Sådana oberoende testcentra kan verkligen behövas.

[63] Oberoende konsult.., 2001-11-09

Tekniken har idag blivit så pass mogen att det går att genomföra en total integrering av företagens it-infrastruktur. Processer, metoder från olika applikationers logik och inte minst de ålderdomliga mainframes kan knytas samman genom en central middleware.

Integrering av applikationer kan uppfattas som en ren teknisk uppgift. Till en viss grad är det också så, men för att utvecklare ska kunna göra en optimal integrering som motsvarar affärsfolkets förväntningar är det viktigt att ha dessa med vid planeringen av integreringen. Då kan man förstå affärsprocesser och transaktioner på samma sätt som företaget och ledningen gör. Det skulle hjälpa utvecklare att utforma en lösning som möter företagets verkliga behov på bästa sätt.

Slutligen vill vi framhålla att integrering inte är ett självändamål men att den faktiskt kan leda till klara ekonomiska fördelar för företagen både i form av sparade mantimmar, samordning av informationstillgångar och inte minst ökad konkurrensförmåga.

7. Källor

7.1 Böcker

Linthicum, David S. 2000, *Enterprise Application Integrering*, Adison-Westley, a
Zahavi, Ron. 2000, *Enterprise Application Integrering with CORBA*,OMG press.
Andersen, Erling. 1994, *Systemutveckling*, Studentlitteratur.
Stallings, William. 1998, *Operating systems*, Prentice-Hall.
Wilkes, Lawrence. 1999, *Application Integrering - Management Guide, strategies and technologies*, Butler Group

7.2 Tidskriftsartiklar

Allt fler leverantörer satsar på dataintegrering, *Computer Sweden* nr 203/02
Byttner, Karl-Johan, Svalt intresser för stora affärssystem. Computer Sweden, 2001-02-12
Nordner, Anders, Web services kräver bra affärsprocesser, Computer Sweden, 2002-02-08
Oberoende konsult testar mellanvaror, Computer Sweden 2001-11-09
Wallström, Martin, Ökning av integrering mellan företag, Computer Sweden, 2001-04-17
Wallström, Martin, Internet ger uppsving för affärssystemsbolag, Computer Sweden, 2001-04-18
UtbildningsGuide nr 1/2002, ComputerSweden, 2002

7.3 Artiklar/material från nätet

Alam, Hisham, *Understanding Integrering Strategies*, Nov 2001, www.eaijournal.com, 2002-04-14

Apte, Atul, *Designing an Integrering-Ready Application*, eai.ebizq.net, 2002-04-14

BEA *WebLogic Integrering. Introduction Application Integrering*, Oct 2001, 2002-04-14

Boucher, Karen, *Application servers on the front line*, December 1999, Middleware/Connectivity, www.standishgroup.com, 2002-04-14

Brown, Tony, *The Application Archipelago*, Jan 2000, www.eaijournal.com, 2002-04-14

Buyens, Marc, *Enterprise Application Integrering (EAI)*. Sept 99, www.eai.ittoolbox.com, 2002-04-14

Cadarette, Paul, *Achieving a complete Enterprise Integrering Strategy*, eai.ebizq.net, 2002-04-14

Dolgier Max, *Building a middleware platform*, eai.ittoolbox.cm , 220-05-18

EAI - ett integreringskoncept, Rapport 2 januari 2002, IVF Industriforskning och utveckling AB, www.ivf.se, 2002-04-14

EAI Overview, eai.ittoolbox.com, 2002-04-14

Forino, Ronald, *Data e.Quality: What is it youre integrating*. www.dmreview.com, 2002-04-14

Gill, Philip, *Open TP Monitors Extend Their Range*, www.uniform.org, 2002-05-15

Gold-Bernstein, Beth, *EAI Market Segmentation*, www.eaijournal.com, 2002-04-14

Griffin Jane, *Information Strategy: Message Brokers - The Cream of the Crop in EAI Tools* www.eaijournal.com 2002-05-08

Holland Earl, *Bring it all together with EAI*, www.ciscoworldmagazine.com, 2002-04-14

Hurwitz, Judith, *Application Integrering. Merging integrering and business value* *www.dbmsmag.com* sept 1998, 2002-04-14

Jacada white paper, *Legacy system integrering*, www.jacada.com , 2002-05-18

IDC, *The Enterprise Application Integrering Market Simmers with Robust Growth Expectations*, February 28, 2001, www.prolifics.org

Inmon William, *A brief history of integrering*. eaijournal.com, 2002-04-14

Johnson, Boucher, Hicks, *Project Manager's Guide to Middleware*, August/September 2001, www.standishgroup.com, 2002-04-14

Linthicum, David, *Application Servers and EAI*, July/Aug 2000, eaijournal.com , 2002-05-15, b

Linthicum, David, *How to Free your Information*, eai.ebizq.net, 2002-04-14 , c

Linthicum, David S. *EAI without the hype*, www.enterprisedev.com, 2002-04-14, d

Linthicum, David S. *EAI Application Integrering Exposed*, www.softwaremag.com, 2002-04-14, e

Linthicum, David S. *Microsoft may have App Server Winner*, www.devx.com, 2002-04-14, f

Linthicum, David S. *Deep Dive into Metadata*, www.softwaremag.com, 2002-04-14, g

Lutz, Jeffrey, *EAI Architecture Pattern*, March 2000, eaijournal.com, 2002-04-14

McHugh Neil, *A quick guide to integrering mainframes*, 2001-12-01 www.eaijournal.com , 2002-05-10

Pollock, Jeffret, *The big Issue: Interoperability vs. Integrering*, eai.ebizq.net, 2002-04-14

Propelis white paper, *Legacy system integrering*, www.propelis.com , 2002-05-18

Ren Frances, *The Marketplace of Enterprise Application Integrering*, www.public.asu.edu 2002-04-14

Stephansen, Steve. *The Benefits of a Peer-to-Peer Architecture*, e-serv.ebizq.net, 2002-04-14

TechMetrix Research, *Which Technology for Tomorrow's EAI?*, e-serv.ebizq.net, 2002-04-14

Traverse, Cheryl, *Adapting to Total Integrering*, September 2001, eaijournal.com, 2002-04-14

Tucker Michael Jay, *Getting a lock an linking legacy data*, www.ebizq.com , 002-05-05