Handelshögskolan
VID GÖTEBORGS UNIVERSITET
Institutionen för informatik
2004-03-23

Microsoft.NET och J2EE – Web Services

En vägledning vid val av mjukvaruplattform

Abstrakt

Den här studien riktar sig främst mot målgruppen företagare och behandlar deras problematik att inte kunna erhålla sammanställd information som inte är vinklad av exempelvis lojalitet mot återförsäljare och dylikt vid inköp av mjukvaruplattformar. De mjukvaruplattformar som uppsatsen behandlar är Sun:s J2EE samt Microsoft:s .NET. Mjukvaruplattformarna presenteras i helhet men störst tonvikt läggs vid Web Services som är en relativt ny form av tjänst inom ramen för mjukvaruplattformar. Web Services kan liknas vid webb-baserade COM-komponenter. Syftet med den här uppsatsen är att fylla detta tomrum och därmed med hjälp av studiens normativa metodik kunna erbjuda målgruppen rekommendationer och vägledning inför deras val av vilken av de två Web Serviceutrustade mjukvaruplattformarna som är lämpligast för just dem. I uppsatsen ingår även en empirisk del som skall tjäna som ett komplement till teoridelen och erbjuda en även praktisk insikt som i sin tur skall ge ett mer komplett beslutsunderlag. Empirin består konkret av ett mindre prestandatest. Även i uppsatsens teoridel ingår återgivning av två större prestandatester. Det resultat som jag har kommit fram till med den här uppsatsen är att ingen av de två plattformarna J2EE eller .NET kan kåras som någon enhällig vinnare. Vilken man skall välja beror på exempelvis sådana saker som de behov man har inom sitt egna företag, vilka behov kunderna har, vilka de är, hur stor den egna budgeten är, vilka plattformar man använder och liknande frågor. Dessutom har jag kommit fram till att man gällande prestandatester inte skall stirra sig blind på de kvantitativa resultaten utan mer se till sådana ’mjukare’ värden som de jag just har nämnt.

Nyckelord: Microsoft.NET, J2EE, Mjukvaruplattform, Web Services

Författare: Tommy Friberg
Handledare: Alan B Carlson
Examensarbete II, 10 poäng

Förord

Jag skulle under detta förord vilja tacka min handledare Alan B Carlson för hans goda stöd och de råd som han har tillfört mig och därmed uppsatsen under arbetets gång. Jag skulle även vilja rikta ett tack till min gode vän och kollega inom detta ämnesområde Jan Valtanen som har varit till stor hjälp och nytta med sin omfattande kunskap.

Trollhättan den 8 mars 2004

Tommy Friberg

Innehållsförteckning

1. Inledning

1.1 Bakgrund

Företagare och potentiella sådana verkar ofta inom en stram budget eller är intresserade av att sänka sina kostnader. I många fall besitter de inte heller själva kunskaperna som behövs för att investera i mjukvaruplattformar eller för delen den hårdvara som man måste ha till plattformarna. Om deras budget är begränsad har de i de flesta fall inte heller råd att anlita en konsult eller kan kanske känna att det skulle vara bra med en guidning innan de anlitar en dyr sådan. Problemet blir sedan att är de inte insatta i datavärlden så vet de kanske inte vart de skall leta efter information, sammanställa den och så vidare. För ofta får man använda sig av information från många olika källor innan man kan erhålla en tillfredställande informationsmängd samt att det även krävs att man har kunskaperna att sammanställa de olika informationsdelarna till helheter på ett bra sätt. I samband med insamlandet av teorin till den här uppsatsen kunde jag konstatera att tillgängligheten efter dokument som har sådana här sammanställningar och som dessutom är oberoende av exempelvis inflytande från vanligtvis Java eller Microsoftlägren eller för den delen någon annan part är inte direkt stor. Själva den totala dokumentmängden är dock omfattande men inte den med nämnda kriterier. Även kvalitén på själva framläggandet av innehållet i de dokument som finns kan vara högst varierande. Som novis kan allt det här kännas oöverblickbart och det kan väl också sammanfatta bakgrunden till denna uppsats.

1.2 Problemområde

Problemområdet kan man säga vilar i faktumet att kunna hitta bra information och samtidigt vara ytterst källkritisk mot det material som man finner. Som jag sagt ovan så är det inte mängden material som främst utgör problematiken utan att bland allt detta material kunna sålla ut vad som är relevant och även bland samma material kunna tillämpa källkritiken på ett sätt som gynnar det syfte man har, i det här fallet att ge så neutral och oberoende information till främst målgruppen företagare som möjligt. För en företagare som dessutom inte är verksam inom databranschen eller ens har speciellt stor generell kunskap om datorer så kan man därmed säga att denna person står inför ett problem. Att kunna få tag på rätt information och dessutom filtrerad enligt de behov som man efterfrågar, till exempel man tagit bort den vinkling som ofta förekommer som jag har diskuterat ovan, så måste de om de inte har någon annanstans att vända sig anlita en dyr konsult eller liknande. Även presentationen av materialet måste ske på ett sätt som är lätt för målgruppen att ta till sig, vilket annars kan utgöra ett delproblem i sig. Med tanke på att inte alla inom min målgrupp, som exempelvis småföretagare, har de pengar som krävs för professionell hjälp, så står man därmed inför ytterligare en dimension av det här problemet.

Man kan därför fråga sig vad den delmängd av min målgrupp med sämre ekonomi och kanske inte så stora krav, som exempelvis en småföretagare, skall med ett till synes dyrt system som är designat för avancerat affärsbruk till? Det kan verka att ta i lite i överkant. Svaret ligger i att dels som det kommer nämnas längre fram så finns systemen i olika prislägen och utföranden. Men även dels om en småföretagare med stram budget väljer ett system i standardutförande så kommer det enligt min mening att göra honom konkurrenskraftig redan från början. I synnerhet med tanke på att Web Services är gjorda för att verka över Internet. Jag är vidare helt övertygad om att mjukvaruplattformar med Web Services kommer att bli lika vanligt förekommande hos alla typer av företag och hos andra med för den delen som till exempel COM-komponenter har blivit idag (läs mer om dessa företeelser längre fram). Om

man läser mitt teoriavsnitt och studerar den debatt som finns runt ämnet idag så kan åtminstone inte jag finna några tecken som tyder på något annat.

Rent konkret så mynnar ovanstående problem ut i ett annat problem vilket är vilken plattform för Web Services som målgruppen skall välja som underlag för sin verksamhet. Allt detta utgör alltså sammantaget mitt problemområde.

1.3 Syfte

Syftet är att med hjälp av denna uppsats skall befintliga samt potentiella företagare kunna finna den hjälp som de behöver för att kunna välja rätt mjukvaruplattform, inkluderandes funktionalitet för Web Services, utifrån sina individuella förutsättningar och ekonomi. Tanken är även att erbjuda en insikt i vad webbservices och mjukvaruplattformar innebär och hur de fungerar. Denna del är ganska ingående för att kunna tillfredställa även dem som vill ha mer teknisk information utan att för den skull bli obegriplig för dem som vill läsa uppsatsen och kunna ta till sig dess informationsutbud på ett mer övergripande sätt (se även under avsnittet målgrupp nedan).

Dessutom förutom all teori så ingår det ett mindre experiment där jag själv har prövat på att göra ett prestandatest i liten skala för att kunna handgripligen få pröva på J2EE- och .NET-plattformarna och dess funktionalitet. Jämförelser av dessa tester med två befintliga som jag har redogjort för under teoriavsnittet finns också att ta del av. Detta för att på så sätt öka insikten av vad dessa plattformar innebär, också på ett praktiskt och inte bara teoretiskt plan.

Därmed skall den här uppsatsen tjänstgöra som en guide där företagare i olika former men även vem som helst kan få all information de behöver främst beträffande Webbservices under de två mjukvaruplattformar som jag har valt, men även för open source-alternativ, hårdvara med mera. Detta utan några snedvridande faktorer som lojalitet mot leverantörer, märkestrogenhet, ekonomiska bindningar eller andra faktorer som kan göra det svårare för den som inte är så insatt i ämnet att kunna skapa ett korrekt beslutsunderlag. Det är det underlaget som sedan skall ligga som grund för kommande investeringar av främst hela plattformar och i synnerhet Web Services men även som sagt var för hårdvara med mera.

För att slutligen tydliggöra mina motiv så kan en kort redogörelse av min erfarenhet gällande Windows- respektive UNIX/Linux-hantering vara på sin plats. Detta för att .NET hör hemma i Windows-världen och J2EE i UNIX/Linux-världen. Dessutom, och allra viktigast, så höjer det uppsatsens trovärdighet gällande en kritisk syn på systemen samt reducerar risken att mina omdömen skulle kunna tolkas för att bygga på till exempel förutfattade meningar om systemen. Något som är av stor vikt för den här uppsatsen.

Jag har större vana av Windows-användning än motsvarande för UNIX/Linux men min det är något som gradvis alltmer håller på att jämnas ut. Jag skulle beskriva mig själv som en avancerad Windodws-användare och en mycket van UNIX/Linux-användare om än kanske inte riktigt en avancerad sådan. Det här är mycket luddiga begrepp men något enklare uttryckt kan man säga att jag är van vid båda systemen om än något mer vid Windows. Jag är enligt eget tycke tillräckligt van för att kunna urskilja och kritiskt bedöma differenser mellan dessa. Dessutom med tanke på att jag nyttjar båda systemen så tycker jag att det gör mig lämplig att kunna jämföra dem. Allra sist så har jag läst ett antal högskolekurser som antingen varit rena operativsystem-kurser med inriktning mot Windows eller UNIX/Linux eller åtminstone haft starka inslag av operativystem.

1.4 Frågeställning
Utifrån allt som nämnts tidigare så har jag kommit fram till nedanstående frågeställning.

Vilken mjukvaruplattform som inkluderar Web Services är det bästa valet för en företagare?

Bör även förtydliga att jag i frågeställningen med ’bästa valet’ menar hänsyn till de kriterium som jag tidigare har nämnt som företagare enligt min uppfattning bör ta innan de genomför en investering av en mjukvaruplattform. Främsta kriteriet är att skaffa en god informationsgrund innan någon investering blir aktuell.

1.5 Avgränsning
Den här uppsatsen innehåller tre delområden som jag har valt skall utgöra dess grundstomme. I det första av dessa tre delområden har jag utsett att uppsatsen skall handla om de två idag dominerande mjukvaruplattformarna J2EE från Sun och .NET från Microsoft, även om det förekommer inslag som berättar om open-source alternativ. Detta på grund av att eftersom det är lättare att ta till sig något som är etablerat eller kommer att lanseras på bred front än något som är mer svårtillgängligt. I synnerhet för uppsatsens målgrupp som man inte med säkerhet kan förvänta sig att de har den kunskap som annars i så fall krävs.

Det andra delområdet som behandlas har jag kommit fram till genom att jag från allt som kan ingå under en mjukvaruplattform valt att koncentrera mig specifikt på Web Services. Detta med anledning av att Web Services är en modern utveckling av de mycket använda och gångbara komponenterna, COM- (Component Object Model) komponenterna, inom världen av system för exempelvis affärsbruk. Med tanke på att system med Web Services blir distribuerade över Internet och hårdvaran kommer därmed spela en annan roll än innan gör det alltihop till ett än mer tilltalande val.

Det sista delområdet som skall behandlas i den här uppsatsen är min målgrupp som alltså är företagare eller potentiella sådana som vill införskaffa Web Service-försedda mjukvaruplattformar till sin verksamhet. Men inte har kunskap och/eller resurser att kunna skaffa den information som krävs före man gör en sådan investering. Observera att uppsatsen kan självklart nyttjas av vilka intressenter som helst som finner ämnesområdet intressant. Men den grundläggande avsikten är att den i första hand skall vara en guide för etablerade eller blivande professionella företagare.

1.6 Målgrupp
Som har nämnts under syfte ovan så är målgruppen befintliga samt potentiella företagare som har en begränsad ekonomi att röra sig med eller alternativt är ute efter minska kostnader vid inköp. Det kan inkludera alla branscher. Just att uppsatsen riktar sig till i princip alla branscher innebär att de som läser den här uppsatsen kanske inte har så stor erfarenhet av datavärlden. Därför har jag försökt att göra en mycket omfattande begreppsapparat och i möjligaste mån även förklarat krångliga och tekniskt täta passager i uppsatsen. Dock är det alltid en balansgång mellan förklarande tydlighet och den stora mängd tekniska begrepp och dylikt som behövs för att förklara olika saker men som också därmed kan försvåra. Det är ändå min förhoppning att med hjälp av begreppsapparaten så skall alla ändå kunna ta till sig innehållet i denna uppsats. Därför kan man säga att det underlättar att ha datorvana när man läser den här uppsatsen men det är ingen nödvändighet.

1.7 Disposition

Under detta avsnitt skall jag i korta steg på ett övergripande sätt visa min tanke med upplägget av denna uppsats genom att berätta vad varje kapitel innebär.

Kapitel 1-Inledning
Behandlar de inledande delarna som bildar grogrunden för uppsatsen. I kapitlet kan man följa mina tankar och idéer och hur jag sammanförde dem i den utformning som bildade kärnan för det fortsatta innehållet i uppsatsen.

Kapitel 2-Metodik
Om teoriavsnittet nedan utgör den massa som med vars hjälp jag bildade mig den uppfattning som behövdes för slutsatser, diskussion med mera, så är metodiken sättet som jag gjorde det på. Enklare uttryckt hur jag gick till väga i mitt arbete med denna uppsats. Där ingår redogörelser för vilka källor jag har tittat på, hur jag har granskat dem och vilken vetenskaplig syn jag har använt mig av. Sist i kapitlet nämns några ord om källkritik.

Kapitel 3- Teori
Teoriavsnittet är uppsatsens mest omfattande avsnitt där jag tog hjälp av det som tidigare hade skrivits inom ämnet och som även passade de riktlinjer som jag drog upp under inledningskapitlet. Teorin skall verka som en förankring för de delar av ämnesområdet som jag hade valt. Den skall därmed utgöra basen för de slutsatser, funderingar med mera som jag med bland annat dess hjälp hade kommit fram till och även senare redovisas i uppsatsen.

Kapitel 4-Emperi
Empiri är här samtydigt med min egen undersökning och dokumentationen av densamma. Min egen undersökning är ett litet experiment i form av ett prestandatest som utfördes med de båda valda mjukvaruplattformarna J2EE och .NET. De användes då med Web Services och mot ett så kallad referensimplementations-program som är en testmjukvara i form av en djuraffär för Internetbruk. På klientdatorn så mättes med hjälp av programmet JMeter upp hur många förfrågningar per sekund som respektive Web Services-system klarade av. Det programmet skötte även de simulerade förfrågningar som utgjorde belastningen på servern.

I övrigt i detta kapitel redogör jag för inte bara siffror och dylikt från testet utan delger även bland annat intryck och upplevelser. Det förekommer också jämförelser med de två prestandatest som ingår i teoriavsnittet för att på så sätt kunna se vilka skillnader och likheter som fanns mellan dessa.

Kapitel 5-Slutsats
I detta det avslutande kapitel vad det gäller uppsatsens egentliga innehåll så fungerar det som sammanfattning samt redogörelse av vad jag kom fram till. Här finns de slutsatser som jag har dragit, en diskussion där teorin kommenteras med mina egna åsikter och även empiri avsnittet behandlas på samma sätt, utvärdering av metod förekommer där metodikens effektivitet granskades samt förslag på fortsatt forskning ges.

Kapitel 6-Begreppsapparat
Här finns en stor mängd begrepp som förekommer i uppsatsen. Syftet är att underlätta för läsare som inte är bekanta med den begreppsflora som förekommer i uppsatsen. Det bör väl

ändå påpekas att den trots sin omfattning ändå är ett urval, men ett urval som jag hoppas skall vara tillfredsställande för de flesta läsare.

Kapitel 7-Referenser
Här listas referenserna upp över de källor som jag har använt i den här uppsatsen. De flesta källorna är hämtade från Internet med några undantag som utgörs av bland annat böcker.

2. Metodik

2.1 Presentation av metoder

För att kunna besvara denna uppsats frågeställning så har jag genomfört en normativ studie. Detta för att kunna tillhandahålla rekommendationer vid inköp för det som är uppsatsens målgrupp, det vill säga befintliga eller potentiella företagare. För att kunna besvara uppsatsens frågeställning och för att tjäna det syfte som jag har med uppsatsen så har jag försökt ta fram ett så brett underlagsmaterial som möjligt gällande Web Services under J2EE- och .NET-plattformarna. Materialet täcker även plattformarna i övrigt för att man skall kunna få sig en helhetssyn när man köper en plattform, och tar även upp en del om open source-alternativ. På grund av uppsatsens inriktning på Web Services kommer dock som sagt var tyngdpunkten att ändå vila på just dessa men ändå inte förlora helhetsperspektivet över plattformarna. Även en empirisk del ingår i min normativa studie, närmare bestämt ett litet prestandatest, och det behandlas som en integrerad del i detta kapitel.

Enligt Backman (Backman, 1998) så skall all forskning initieras med en litteraturstudie för att ge en grund åt det ämne som skall granskas. Eftersom min normativa studie även kan karaktäriseras som en litteraturstudie så skall jag här börja med att framhäva hur jag har nyttjat min normativa metodik med källorna som grund samt hur jag har använt dem respektive var jag har hittat mina källor.

Jag har i mitt sökande efter deta material som skall agera beslutsunderlag för min målgrupps investeringar av mjukvaruplattformar främst använt mig av källor hämtade på Internet med undantaget av de böcker som jag har använt mig av. De har varit nästan uteslutande vanliga ’pappersböcker’samt en elektronisk sådan. Källorna på nätet har varit av följande sorter: Uppsatser av främst akademisk karaktär, artiklar från nätversioner av tidningar som exempelvis Computer Sweden, andra typer av artiklar inkluderandes vetenskapliga diton, nedladdningsbara dokument främst i formaten doc och pdf som till exempel har innehållit produktbeskrivningar samt så kallade whitepapers. Även exempelvis Microsoft:s och Sun:s hemsidor har besökts för inhämtande av material.

Från ovan uppräknade källor kan man grovt säga att jag har hämtat två områden av information för att på så sätt uppfylla den breda beslutsbas som är avsedd för uppsatsens målgrupp samt därmed också gå i linje med dess frågeställning. Dessa områden kan man kategorisera som förståelse respektive jämförelse/hantering.

Vad det gäller förståelse-kategorin så handlar den om att lära plattformarnas historik, uppbyggnad, funktionalitet och dylikt. Kort sagt de mer tekniskt inriktade bitarna. Dessa är de som kan verka mest svåra för den oinvigde att ta till sig. Därför har jag i denna uppsats jobbat mycket med att överbrygga denna komplexitet till något enklare. Detta har jag gjort genom att försöka så långt det går vara mycket förklarande i uppsatsen gällande de mest svåra passagerna. Men det finns en gräns hur långt man kan förklara och förenkla utan att man förlorar substansen och därför finns det en hel del svåra begrepp och även förklaringar kvar. De har jag i sin tur försökt att hjälpa läsaren med genom att i denna uppsats inkludera en mycket omfattande ordlista, som skall täcka om inte allt så upp till en nivå där en förståelse och insikt kan nås. Det är en svår balansgång denna mellan substans och lättillgänglighet, men jag tycker ändå att jag har uppnått mitt mål med att denna uppsats skall kunna fungera som en bas för information inför en investering av mjukvaruplattform med Web Services.

Denna kategoris syfte är att kunna fatta beslut angående en investering. För att kunna göra det så måste man ju förstå hur den fungerar och det för att dels kunna hantera den men ändå mer i ett tidigt skede för att exempelvis veta hur väl den kan interagera med det redan befintliga systemet. Dessutom vidare om det genom sin teknologi kan effektivisera verksamheten, om det behövs något extra för att få det att fungera och i så fall innebär det ju ökade kostnader med mera. Som man märker här så går dessa två kategorier i varandra så det är inte lätt att separera dem men jag gjorde ändå ett försök här för att man skall kunna följa uppsatsens struktur och hur jag försöker att följa dess syfte beträffande att vara informativ på en bred front.

Konkret gällande denna som jag kallar det för förståelse-kategori så jag tagit med följande tekniska bitar från de källor som passade in under den kategorin:

Arkitekturbeskrivningar med tonvikt på den treskiktade arkitekturen men även en historisk del där jag tar upp den tvåskiktade och för att riktigt öka förståelsen gällande arkitekturer så behandlas även vad en mjukvaruarkitektur är för något. Naturligtvis ingår det även två omfattande avsnitt som tar upp plattformarna som jag har valt för denna uppsats, J2EE och .NET. Vidare gör jag en dylik genomgång av Web Services som inleds med historik som tar upp hur det var innan det fanns Web Services för att på så sätt tydliggöra nyttoaspekten av att implementera sådana i verksamheten, vilket därmed framkommer när man belyser denna differens. Inom dessa områden handlar mycket om att klargöra för plattformarnas och dess ingående teknologiers och teknikers funktion samt uppbyggnad och samverkan med andra delar i ett system. Produktbeskrivningar från Microsoft och Sun är också något jag har använt mig av. Dessa har inte alltid berört just direkt plattformarna utan kanske databaser eller operativsystem som skall användas tillsammans med dem. Men att tänka långsiktigt och i helheter av ett system är viktigt så att man inte drar på sig onödiga extrakostnader gällande kompletteringar eller omstruktureringar av det egna systemet enkom på grund av att man tänkte fel från början. Kan kort sagt bli mycket dyrbart för ekonomin och effektiviteten hos företaget. Det som jag har behandlat i detta stycke anser jag är viktiga delar som man för att kunna fatta beslut bör ha en bra insikt av, för annars vet man ju inte vad man köper eller hur det kan påverka den verksamhet som man har eller skall bygga upp. Det är syftet med denna del att kunna erbjuda just den sortens beslutsgrundande information.

Om vi går över till den andra kategorin ’ jämförelse/hantering’ så handlar den om de lite mer mjukare delarna. Det vill säga sådant från det material jag har använt mig av som tar upp det som frågeställningen direkt uttalar sig om, det vill säga jämförelsen för att komma fram till vilken plattform som är bäst för just den aktuelle investeraren samt att utröna skillnader och dylikt. Även här så är det främst plattformarna J2EE och .NET som behandlas även om det förekommer en del mindre inslag av open source alternativ. Det som allmänt bör observeras gällande det jämförande materialet är att det här är andras jämförelser jag granskar och inte mina egna. Det jämförande materialet kan vidare vara av mer tekniskt karaktär vilket då glider över på den andra kategorin mera men är ändå en jämförelse. En del material har andra jämförelser som är av mer diskuterande eller till och med argumenterande karaktär eller rent utav ifrågasättande. Allt material av den här sorten är mycket bra som beslutsunderlag men det är viktigt att man tar del av de båda karaktärsorternas argumentation. En sak som jag har jobbat en hel del med är att försöka ta bort de reklamvinklingar och hävdelser av de egna produkterna eller de som man har sin lojalitet mot. Dessa stavas nästan alltid genom uppsatsen Microsoft och Sun. Jag har inte censurerat något men tonat ner dem för att de tar bort det relevanta och verkligen förvirrar någon som inte är så insatt i datavärlden och/eller att granska olika material kritiskt. Det skulle dessutom gå mot den här uppsatsen grundläggande

syften om jag tillät något sådant förutom att det som är bra i materialet inte skulle framträda på samma tydliga sätt. Så enligt min mening är det mycket viktigt att just materialet blir så neutralt som möjligt för annars blir det omöjligt att besvara uppsatsens frågeställning vilken plattform som är bäst, eller att i alla fall kunna göra så på ett korrekt sätt. Det är ändå just det här som är uppsatsen huvudtema, valet av plattform, och resterande material som jag bedömer det skall leda fram till att man kan fatta ett beslut i just denna fråga. Därför min noggrannhet att försöka undvika nämnda påverkan och vinklingar.

Två stycken prestandatester finns med i teoriavsnittet för att man skall kunna konkret få ta del av hur plattformarna beter sig under ytterst pressade förhållanden. Det är värdefull information som är nyttig inte minst med tanke på att man får reda på hur mycket effektivitet ett system kan erbjuda. Dessutom tar dessa tester upp skalbarhet vilket kan vara intressant för framtida expansion av företaget om än kanske inte direkt aktuellt just efter inköp av mjukvaruplattform. I samband med dessa tester så gör jag även jämförelser med min egen test och främsta syftet för detta är väl så att man skall kunna få en mer familjär synvinkel av det hela och i en mindre miljö och med mindre resurser än vad som förekommer i de två nyss nämnda större testerna. Mina förhållanden liknar mer hur det kan vara hos en exempelvis nyss öppnad firma, resurserna är inte överväldigande även om de kanske har något mer sådana än vad jag har. Det genererar kort sagt en igenkänningsfaktor som jag hoppas att jag kan överbrygga via dokumentationen av mina erfarenheter från mitt test till läsare av denna uppsats.

Just gällande användande så finns det i uppsatsen med ett avsnitt med ett konkret exempel av användande där man får en inblick hur det system som man funderar på att använda fungerar i den vardagliga verkligheten. Prestandatesterna är ju om än bra ändå simulerade miljöer som ägt rum i olika labb. Jag tror att förutom delgivningen som jag nämnde av mina egna erfarenheter genom mitt test så kan ett sådant här konkret användningsexempel vara bra för ge en djupare insikt i hur plattformarna och framför allt Web Services fungerar. Det är ju ändå i liknande miljöer som plattformarna skall agera. Det är därmed av högsta vikt för företagare som är intresserade av att investera i mjukvaruplattformar att se mer handgripligt hur deras eventuellt blivande investeringsobjekt kan vara av nytta för dem.

Slutligen så har jag från mina källor av den här karaktären skapat ett avsnitt i uppsatsen som behandlar de kriterier/krav som jag anser är viktiga att tänka på inför implementeringen av ett system i en verksamhet. Detta för att man skall kunna få en inblick i vilka krav som är viktiga att överväga inför sitt val av plattform och anpassning av den till företaget i fråga samt vad dessa krav innebär. De krav som jag med hjälp av mina källor har tagit med i det avsnittet i uppsatsen är:

Kostnader för de olika plattformarna inklusive hårdvara med mera vilket alltid är en kritisk fråga i alla företag.

Inlärningskurva är viktigt att överväga på grund av att ju svårare ett system är desto större anledning att betänka om det behövs anställas ytterligare en tekniker eller om man kanske kan lära sig det själv.

Skalbarhet är kanske inte av så stor relevans just nu men finns med på grund av att man bör som företagare blicka framåt och inte tänka kortsiktigt. Därför är det enligt min uppfattning bra att ta med redan vid anskaffandet av en mjukvaruplattform den goda förutsättningen att företaget kan expandera. Vid en sådan expansion så skall systemet som man skaffade lätt

kunna följa med i de förändringarna utan att det skall behöva medföra större strukturförändringar som därmed innebär onödiga kostnader.

Support är viktigt för förr eller senare behöver man hjälp med något angående systemet och har man då ett bra och väl fungerande supportavtal så kan man förhoppningsvis snabbt få ingång systemet igen när det har krånglat och slipper därmed dyrbara driftstopp eller andra störningar av verksamheten.

Mognad kan vara något som är värt att tänka på för kring ett moget system har det hunnit att utvecklas exempelvis mycket tredjehandsmjukvara och även vanligtvis en avsevärd kunskapsbas har genererats som man kan dra nytta av. Nyare system är i princip tvärtom och detta är något som jag med det här avsnittet vill uppmärksamma för det är kanske inte något som man vanligtvis tänker på vid anförskaffande av ett system, i synnerhet inte om man inte har någon tidigare erfarenhet av ett sådant.

Portabilitet, med det menas när det gäller mjukvara att den har förmågan att kunna exekveras på ett antal olika hårdvaruplattformar. Detta är i den Internetvärld vi lever i idag inte bara en förutsättning för nya systems existens utan skall man vara konkurrenskraftig som företagare och kanske även ny i den bransch man verkar inom så är det snarare en nödvändighet. Internet är idag en mycket viktig kanal för bland annat marknadsföring och enligt min mening inte något som man bör missa. Dessutom ligger det ju i själva naturen hos Web Services att verka över Internet så själva arbetet som utförs av företagen integreras ju än mer med Internet idag. Men alla företag har ju inte samma plattformar vilket är ett klassiskt Internet-problem och därför är det viktigt att man ser över vilka möjligheter det system som man är intresserad av har när det gäller portabilitet. Därför har jag ansamlat en mängd material och skapat denna punkt i uppsatsens avsnitt om implementeringskrav.

Prestanda är också någon som man verkligen inte bör ignorera. Med faktumet att trafiken blir allt tätare och intensiv på Internet vilket förhoppningsvis innebär för uppsatsens målgrupp företagarna att de drabbas av denna strida ström av kunder till sitt företag. Problematiken som då kan uppstå är att det system man har inte orkar med denna anstormning, därför bör man se över sitt system så att det innehar funktionalitet samt är designat för den belastningsmängd man har idag men även kommer att få i framtiden. Som man kan se så integreras den här punkten ganska kraftigt med skalbarhet, prestanda och skalbarhet har många gemensamma berörningspunkter som man kan märka efter den här beskrivningen och därav är de mycket viktiga att ta under beaktande vid val av plattform.

Säkerhet är en i mitt tycke självklar fråga som förvånansvärt många både små och stora företag slarvar med. Men tänker man redan på den från början och ser till att det system som man väljer har ett fullgott skydd mot exempelvis dataintrång och virusangrepp så kan man spara mycket pengar som annars hade behövts läggas på att sanera de skador som denna typ av säkerhetsrisker kan medföra.

2.2 Vetenskapligt synsätt

Det synsätt som jag har anlagt för den här uppsatsen är den av en förmedlare av upplysning till min målgrupp enligt den normativa metodik som jag har använt mig av. I förmedlarrollen ingår även rollerna av att vara samlare samt sammanställare av informationen så att den passar in på så många i min målgrupp som möjligt vilket är ett av den här uppsatsens huvudsyften. Genom att anpassa den på ett bra sätt så kan presentatören det vill säga förmedlaren på bästa sett ge målgruppen den information de behöver och då också på ett

korrekt sätt. Det i sin tur gör att målgruppen, det vill säga företagarna, på ett så bra sätt som möjligt kan fatta de beslut som krävs vid anförskaffandet av den mjukvaruplattform som de senare väljer. Sålunda om uppsatsen kan fungera på det sättet så är dess absoluta huvudsyfte till fullo uppfyllt. Så synsättet kan man kort säga är det av en förmedlare eller en spridare av väl för målgruppen bearbetad information om man så vill.

2.3 Källkritik

Som avslutning på det här kapitlet skall jag nämna något om granskningen av de källor jag har använt mig av. Det är ju av hög relevans för en uppsats som bygger på att materialet som nyttjas är så fritt från vinklingar och dylikt som möjligt. Därför kommer jag här att koncentrera mig på just dessa vinklingar.

Det svåra med identifieringen av de vinklingar som jag har funnit i undersökningsmaterialet för den här uppsatsen, är att de ofta är relativt subtila. Det sägs inte rakt ut vad man egentligen är partisk mot utan man får så att säga läsa mellan raderna. Därför kan det vara svårt för mig att här rent konkret demonstera detta med tanke på att uppsatsen riktar sig även till dem bland målgruppen som kanske inte är så insatta i ämnet och då blir subtila uttryck något som försvårar det hela alltför mycket. Jag har ändå lyckats att hitta en källa där det riktas mer rak kritik mot .NET och man inte döljer var lojaliteten ligger. Jag tänker från denna källa nedan återge en kort men mycket talande återgivelse:

> Någon sa att .NET är en lösning i sökandet efter ett problem. Java har redan löst problemet, och användare vill ha generella lösningar. Ekvivalent med detta så kan man använda sig av Karl Krauses berömda kommenter om psykiatri och säga att, trots alla löften, så kan .NET mycket väl vara sjukdomen vilket det hävdar vara botemedlet mot (Hillesley, 2001).

Tyvärr är källor med så här tydliga exempel ganska ovanliga och därför kan jag inte återge någon ytterligare sådan.

Allra sist kan jag väl säga att det bör observeras i sammanahanget att alla källor inte är vinklade eller präglade av exempelvis lojalitet i stil med det här redovisade exemplet, men att många är det om än inte så väldigt tydligt som ovan.

3. Teori
Teorikapitlet skall utgöra en grundstomme för den här uppsatsen som på grund av sin normativa metodik hämtar mycket av det som formas till rekommendationer för målgruppen just från detta kapitels innehåll. Det är under detta kapitel man som läsare kan ta del av vad jag har funnit av det som tidigare har skrivits om ämnet. Kapitlet inleds med en exemplifiering av hur man kan använda Web Services, vilket även utgör en mycket bra introduktion till ämnet. Därefter följs en förklaring av begreppet Web Services, vilket kan vara en lämplig fortsättning efter att ha sett hur dessa kan användas. Som en inledning till de mer tekniska detaljerna så efterföljs det av ett avsnitt som ger en historisk tillbakablick gällande de tekniska aspekterna av Web Services. Efter det så följer två omfattande avsnitt som ger läsaren mer detaljerad teknisk info om .NET och J2EE samt ett mindre avsnitt om open source-varianten Mono. Efter det är det dags för nästa omfattande avsnitt som försöker att klargöra skillnaderna mellan dessa plattformar samt ge råd inför valet av en av dem. Näst på tur är sedan fyra stycken avsnitt gällande prestandatester. Detta för att ge läsaren en än mer praktisk insyn av hur Web Services men även de två plattformarna i övrigt som behandlas i uppsatsen, det vill säga .NET och J2EE, fungerar under samt klarar av extrem belastning. Det första av dem är en introduktion där läsaren bland annat får en definition av vad begreppet prestandatest innebär. Detta för att ge en god start innan ämnet behandlas vidare. Därefter kommer ett avsnitt där det visas hur en specifikation över prestandatester kan se ut. Det första av de sedan två efterföljande avsnitten testar mer ordinära Webb tjänster och hur ASP.NET fungerar med dem medan det andra av dem inkluderar bland annat test av Web Services. Kapitlet avslutas sedan med ett avsnitt där det redogörs för vilka kriterier som det är viktigt att tänka på inför implementering av en mjukvaruplattform med Web Services i den egna verksamheten. Det kan vara en bra hjälp för uppsatsens målgrupp ifall de själva inte vet vilka kriterier som kan vara viktiga att tänka på och således inte heller vet vad inom dessa kriterier som är viktigt. Kriterierna jämförs därför mot mjukvaruplattformarna för att på så sätt lättare påvisa vad man bör tänka på under respektive kriterium.

Innan vi börjar med den fortsätta läsningen vill jag påpeka att det under kapitel 6: Ordlista, finns förklarat en stor mängd väsentliga begrepp, ord samt uttryck. Detta för att underlätta förståelsen av detta teoriavnsitt.

Men allra först inleds alltså kapitlet med en exemplifiering av användande gällande Web Services.

3.1 Exemplifiering av användande gällande Web Services
Nedan skall det i detta mycket korta avsnitt förevisas ett konkret exempel på hur Web Services kan vara användbara för en organisation. Detta avsnitt kan ses som en mjukstart av ämnet på grund av att en konkret exemplifiering kan ge en första insikt i vad en mjukvaruplattform kan innebära för en organisation

3.1.1 Hur Web Services ansluter applikationer i en organisation
Säg att du har ett fristående lagersystem. Om du inte ansluter det till något annat, så är det inte så värdefullt som det skulle kunna vara. Systemet kan registrera en produkt som finns i lager, men inte så mycket mer. Information om produkterna kanske måste matas in separat i bokförings- respektive kunddatasystemen. Lagersystemet kan även kanske vara oförmöget att automatiskt placera orders hos återköpare. Fördelarna med ett sådant lagersystem överskuggas av höga allmänna omkostnader. Hur som helst, om du ansluter ditt lagersystem till ditt bokföringssystem med XML, så blir det mer intressant. För då, när du köper eller säljer någonting, så kan allt som berör lager eller ditt kontantflöde registreras i ett enda steg.

Om du går vidare, och även ansluter det system som har hand om ditt säg till exempel varuhus med ordersystemen för kunder samt återförsäljare och dessutom kopplar ihop allt detta med systemet för dina transporter, och gör det med XML. Då har du plötsligt en situation där ditt ursprungliga lagerhanteringssystem har expanderats och effektiviserats så att det har blivit värt en hel del.

Du har nu full kontroll över alla aspekter av din affärsverksamhet, och varje transaktion behandlas bara en gång istället för som förut en gång för varje system som den berörde. Resultat av dessa förändringar är att mycket mindre arbete krävs och dessutom är det lägre chans för att fel skall uppstå (Microsofts svenska hemsida 2b).

3.2 Vad är en webservice?

Efter att ovan givits en första introduktion av ämnet Web Services skall här följas en närmare förklaring av vad begreppet innebär. Under detta avsnitt kommer man att få reda på vad en Web Service är för något, vad den kan användas till, presentation av det nya koncept som Web Services innebär, alternativ till Web Services, hur de är uppbyggda samt avslutningsvis en genomgång av dess mest vanliga tekniker. Men allra först alltså en definition.

3.2.1 Definition

Inledningsvis så kan man se på vad Web Services är på en mängd olika sätt beroende på preferenser, erfarenhet, vad man tänker använda dem till och en mängd andra faktorer hos dem som talar om eller bedömer dem. Det här avsnittet är tänkt som en introduktion till vad Web Services kan vara och kan tolkas som att vara. Som ni säkert förstår finns det många fler tolkningar men här återges ett fåtal som därmed kan tjäna som en bra introduktion.

Web Services utvecklades som lösningen för att på ett standardiserat sätt kunna erhålla data utan patentskyddad mjukvara samt hårdvara (Lurie & Belanger, 2002). Webb Services kan (Microsofts svenska hemsida 2a) omtalas som små program vilka är högst återvinningsbara. Andra sätt att beskriva dem är som diskreta kodenheter där varje enhet hanterar en begränsad mängd sysslor (Microsofts svenska hemsida 2b) eller som XML-baserade interface för affärs-, applikations-, och systemtjänster och egentligen gamla teknologier bärandes en ny skrud (Vawter & Roman, 2001).

Ett annat sätt att närma sig vad Web Services kan innebära är att göra som Kirtland (Kirtland 2001) och jämföra dem med en komponent. Enligt Kirtland består ett program av olika samarbetande Web Services. I likhet med vanliga komponenter så skickar Web Services med ett interface med metoder vid anrop från exempelvis användare och dessutom så abstraheras detta helt, det vill säga implementationen, från användaren. Nämnda metoder skickar och tar emot anrop och dylikt enbart under förutsättning att de är XML meddelanden. En markant skillnad i sammanhanget är att Web Services inte nyttjar till skillnad från komponenter protokoll som till exempel DCOM (Distributed Component Object Model), RMI (Remote Method Invocation) eller IIOP. Istället sker all kommunikation som sagt med hjälp av XML men med den mycket viktiga skillnaden att den sker över HTTP-protokollet som utför själva transporten på Internet.

När det kommer till mer handfasta definitioner så återger jag här ett urval av dessa:

Sun defenierar web services som:

> Ett program som accepterar förfrågningar från andra system som befinner sig på Internet eller på ett intranet, förmedlat av enkla, märkes-neutrala kommunikationsteknologier. (Lurie & Belanger, 2002)

Suns störste konkurrent Microsoft väljer däremot att definiera Webservices som:

> XML Web Services låter program dela data, och mer vitalt ändå, integrerar möjligheter från andra program utan att behöva ta hänsyn till de är konstruerade, vilket operativsystem eller plattform de används på, och vilka tekniker eller dylikt som används för att nå access till dem. (Lurie & Belanger, 2002)

Graham Glass, VD och chefsarkitekt hos Mind Electric (mjukvaruföretag för Web Services) definierar Web Services som:

> En samling av funktioner som är paketerade som en enskild enhet och publicerat via nätverket för att där användas av andra program. Web Services är byggstenar för skapandet av öppna distribuerade system, och tillåter företag och individer att snabbt och billigt göra deras tillgångar globalt tillgänglig. (Vawter & Roman, 2001)

Det verkar gällande både Sun och Microsoft som de är relativt överens beträffande definitionen av Web Services. Sett från en rent intuitiv synvinkel så kan man säga att en Web Service är en tjänst som brukas via Internet. Eller ännu närmare uttryckt, en Web Service är anropad via HTTP och returnerar konsumentens svar, eller data om man så vill, i form av XML. Att formatera datan med XML underlättar också betydligt processen av att konvertera rådata till ett för ändamålet representativt format (Lurie & Belanger, 2002).

3.2.2 I användning

Vid användande av Web Services så existerar det enligt Johansson och Ledin (Johansson & Ledin, 2001) fem huvudbegrepp som man bör ta i aktning:

- Data representeras på ett standardiserat sätt
- Meddelandeformatet är standardiserat och utbyggbart
- Tjänster beskrivs på ett på ett standardiserat och utbyggbart sätt
- En metod för att hitta Web Services som finns på en specifik Webbadress
- En metod för att hitta Web Services servrar

I Web Services så motsvaras data och datatyper av XML och XML Scheman. Skickande av standardiserade meddelanden sker med hjälp av protokollet SOAP. SOAP i sin tur består av konventioner för RPC (Remote Procedure Call) samt bindningar till HTTP protokollet. Det innebär i klartext att SOAP meddelanden skickas över Internet med HTTP protokollet (Kirtland 2001).

Vid användning av en Web Service kan det vara praktiskt att ha en förteckning över vilka meddelanden som Web Service komponenten i fråga kan skicka eller ta emot. En dylik förteckning kallas för ett kontrakt och beskrivs med ett XML-baserat språk som kallas för WDSL (Bravetti et al, 2004; Kirtland 2001).

Därefter behöver man någonstans att skicka sina Web Servicemeddelanden för integration med andra Web Services. Vet man inte vart man då skall vända sig så kan man ta hjälp av protokollet UDDI. Med hjälp av detta protokoll kan man se vad olika webbservers har för Web Services tillgängliga. Är fallet istället så att man vet vart man skall vända sig men på den URL:en (Uniform Resource Locator) inte vet vilka Web Services som erbjuds så kan man använda sig av det XML-baserade protokollet DISCO (Discovery of Web Services) (Bravetti et al, 2004; Kirtland 2001).

Vissa av de nämnda teknikerna kommer längre fram i detta kapitel att beskrivas mer utförligt.

Detta avsnitt skall avslutas genom en kort redovisning av anslutningsmöjligheter, fördelar av-, samt en exemplifiering av användande gällande Web Services.

Följande kombinationer av annars icke anslutningsbara källor, gällande webbtjänster, är möjliga med hjälp av Web Services:

Klient till klient
”Smarta” klienter eller tillbehör kan agera som värdar och använda XML Web Services som tillåter data att bli distribuerad utan någon hänsyn alls till varken rums- eller tidsaspekter.

Klient till server
XML Web Services kan distribuera data från en servers programvara till en desktop eller mobil enhet via Internet

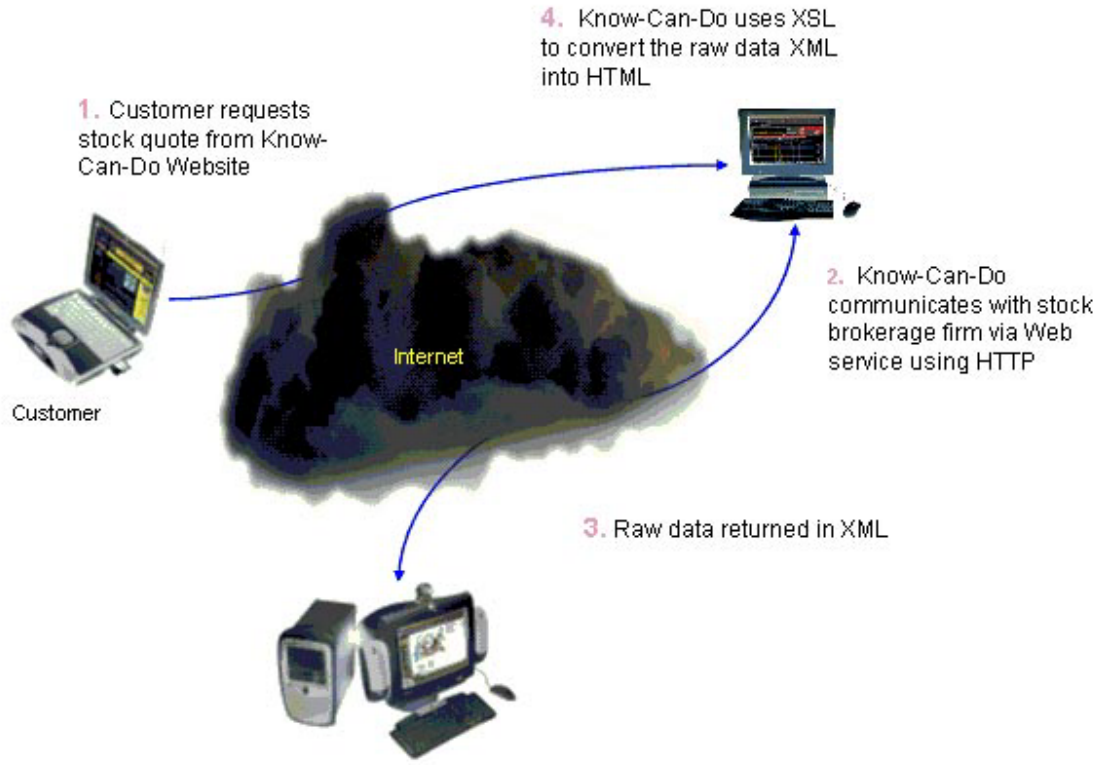
Server till server
XML Web Services tillhandahåller ett reguljärt interface mellan befintliga program inom miljön för fristående servers.

Tjänst till tjänst
XML Web Services samarbetar sekventiellt för skapandet av en mer avancerad data process. (Microsofts svenska hemsida 2a).

Fördelar med att använda web services (Microsofts svenska hemsida 2b):

- Genom förenklad kontakt med affärspartners kan nya affärsmöjligheter realiseras
- Minskar utvecklingskostnader vilket innebär reducerat nyttjande av kapital och tid.
- Egen och enkel distribution av Web Services för andra bidrar till ökade intäkter

Avslutningsvis skall figur 3:3 nedan framlägga hur det fiktiva företaget ’Know-Can-Do’ kan tänkas använda sig av Web Services (Lurie & Belanger, 2002):



Figur 3:3. Schematisk exemplifiering av hur Web Services kan användas.

(Källa: Lurie & Belanger, 2002)

3.2.3 Ett nytt koncept

Web Services signalerar om helt nya paradigmer gällande mjukvaruutveckling. Web Services förordar att stora applikationer segmenteras så att enskilda komponenter kan förekomma som Web Services. Det här sättet att utforma segmentering kan helt klart ses som en klar motsats till nuvarande praxis: Att segmentera till DLL:s (Dynamic Link Library) och COM-komponenter (Lurie & Belanger, 2002).

Faktum är att man kan dra en stark analogi mellan Web Services och DLL:s. Båda koncentrerar sig på liknande funktionalitet: exempelvis, affärspolicies och databas-access. Men de skiljer sig också åt markant. För det första, man kan nå åtkomst till en Web Service via HTTP protokollet med hjälp av virtuellt sett vilken webbläsare som helst. Det är inte alls möjligt med DLL:er som brukligen anropas av klienter hemmahörandes i samma intranet. Detta innebär att Web Services kommer att förekomma i en ny distribuerad data-era. För det andra, XML förmedlar returdatan i form av dataformatet XML. DLL:ers datareturer är ofta mycket mer låsta vid vilket dataspråk som används och blir som resultat av det mer begränsade (Lurie & Belanger, 2002).

Följande differenser mellan Web Services och deras föregångare DLL uppkom som en följd av kraftiga inom branschen förekommande trender (Lurie & Belanger, 2002):

1. HTTP som ett protokoll för Internet-access
2. XML som de facto-standard för dataöverföring

Dessa två punkter kan sägas stå för de grundläggande byggstenarna runt vilken Web Servicen är skapad

Just Web Services rörlighet som nämns ovan gör det möjligt att med hjälp av XML över Internet med HTTP-protokollet skapa en mkt enkel kommunikationsmekanism där vilket utvecklingsspråk, middleware eller plattform som helst kan delta. Detta ökar de gränsöverskridande kommunikationsmöjligheterna avsevärt. Att de här teknologierna dessutom utgör en industristandard med avsevärd acceptans inom branschen gör att det innebär en mycket låg risk att implementera dem i sin organisation. Web Services genererar förmågan att snabbt och kostnadseffektivt kunna integrera två affärsrörelser, avdelningar eller applikationer (Vawter & Roman, 2001).

En vision som existerar när det gäller Web Services är att tjänster kommer att bli självregistrerande i publika eller privata affärsregister. Dessa Web-tjänster kommer helt och fullt att kunna beskriva sig själva, inkluderandes strukturen av dess interface, nödvändigheter vid affärsuppgörelser, affärs processer samt villkor och tillstånd för dess användande. Information skall bara behöva skrivas in en gång och man skall kunna anropa andra tjänster för att fullfölja en uppgift och därmed ge en mycket för användaren personligt designad tjänst. Detta kräver dock att tjänsterna delar sådana uppgifter som användarinformation med mera (Vawter & Roman, 2001).

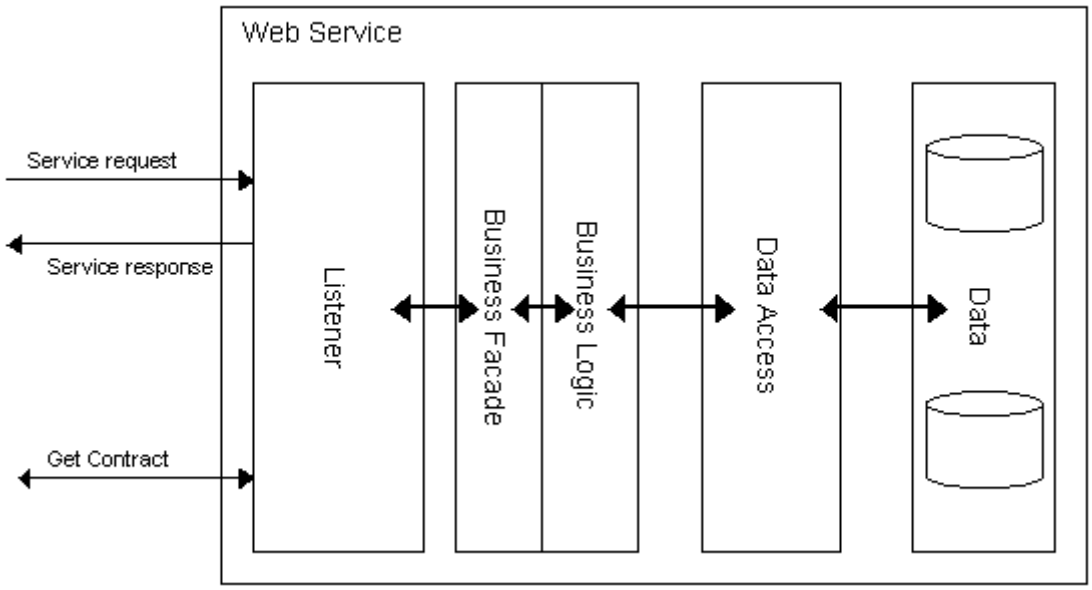
3.2.4 Alternativ gällande Web Services

Det finns en stor mängd alternativ att välja bland när det gäller Web Services. Borland erbjuder Web Services för Linux bruk med hjälp av Kylix (visuellt och komponentbaserat utvecklingsverktyg) och Java, från Sun kommer J2EE för vilket det enligt Hillesley (Hillesley, 2001) år 2001 fanns minst trettio underleverantörer till.

Mono (Se även specifikt avsnittet om Mono: ’3.7 Andra alternativ: Mono, ett open-source projekt för implementation av .NET’) och DotGNU erbjuder .NET kloner vilka bidrager till en mer blandad aspekt beträffande .NET i åtanke samtidigt som det öppnar plattformen för Linux användare. Dock kommer Mono att implementera utvecklingsverktyg för .NET men annars i övrigt vara självständigt.

3.2.5 Uppbyggnad

Arkitekturen hos en enskild Web Service är enligt följande: Den består av fem lager. Det första är datalagret som förser tjänsten med den data som den använder. Nästa skikt är data access lagret som tillhandahåller en så kallad logisk vy av datan och motverkar direkt påverkan från nästkommande lager, Business Layer. Tanken bakom detta är att det enda sättet på vilken manipulation av den underliggande datan skall vara möjlig är genom att använda data Access lagrets metoder. Det bidrar i sin tur till att datans integritet kan garanteras. Business Layer innefattar två underliggande lager: Business Logic som är själva tjänstens logik och det så kallade Business Facade som tillhandahåller den så kallade lyssnaren med de tillgängliga metoderna. Sista lagret är nämnda lyssnare (Listener) vilken har som funktionalitet att mottaga meddelanden, tolka dem samt anropa korrekt metod i Business Facade lagret. Vid sändande av eventuellt svar till klienten så ompaketerar först lyssnaren svaret till meddelande form. Kontraktsdelen omhändertages även av lyssnaren (Kirtland 2001). Nedan visas en schematisk figur över en enskild Web Service.



Figur 3:4. Övergripande framställning av ingående delar i en Web Service.

(Källa: Kirtland 2001)

3.2.6 Viktiga tekniker

Ovan har det mer översiktligt nämnts några tekniker som är vitala för att Web Services skall fungera. Här nedan tänker jag mer utförligt återge de viktigaste av dessa. Eller som Ronald Schmelzer, senior analytiker vid ZapThink ett företag som specialiserat sig på XML och Web Services uttrycker det (Hoffman): ”These are the key standards” (uttalandet inkluderar dock inte Axis som jag tog med nedan).

3.2.6.1 XML

XML är enkelt uttryckt ett allmänt och standardiserat sätt att utforma information med. XML:s oberoende ställning, det vill säga att det inte är bundet till exempelvis någon viss typ av program, gör att det kan nyttjas inom en mängd olika områden och situationer. XML är ett simpelt och anpassningsbart textformat baserat på en begränsad del av ISO – (International Organization for Standardization) standarden SGML (Standard Generalised Markup Language) (ISO 8879) som härrör från 1986 och som det än mer kända protokollet HTTP har sin grund i (Celander, 2002).

XML är helt enkelt kan man säga en rationalisering av SGML, som dessutom innehåller en syntax för skildra innehållet i ett dokument. Själva rationaliseringen innebar konkret att man tog det bästa från SGML, och kasserade resten som folk inte brukade använda samt adderade vissa justeringar för att göra XML mer smidigt för webben (Celander, 2002).

Strukturmässigt så består XML av ett eller flertalet element som i sin tur kan innehålla element eller text. Faktumet att XML endast är en syntax gör att det behövs en uppsättning regler för beskrivning av diverse datatyper, villkor gällande innehåll i ett XML–dokument som till exempel vilka element som får finnas i ett specifikt XML-dokument samt till sist vilka attribut som får associeras med de elementen. Uppsättningen av regler kallas för XML-scheman (Hammarberg, 2002). Exempel på XML-syntax:

<NAMN>
Detta element innehåller text och två element.
<FÖRNAMN>Kalle</FÖRNAMN>
<EFTERNAMN>Anka</EFTERNAMN>
</NAMN>

3.2.6.2 SOAP

SOAP är baserat på XML och har som användningsområde att erhålla informationsutbyte i utlokaliserade och distribuerade miljöer. SOAP anger inte själv något beskrivningssätt för program, som till exempel en programmeringsmodell och inte heller något motsvarande för implementationer. Det fastställer istället en simpel mekanism för att beskriva program och gör så genom att erbjuda en enhetsbaserad paketeringsmodell och kodningsmekanismer med syftet att koda data inom moduler. Kontentan av det här bidrager till att SOAP är brukbart inom en mängd skilda system (Box et al, 2000; Gordon & Pucella, 2002).

SOAP består av följande tre delar:

- SOAP-kuvert (envelope) är ett ramverk för generell beskrivelse av ett meddelandes innehåll, vem som skall hantera det, om det är frivilligt eller obligatoriskt.
- SOAP-kodningsregler som definierar en mekanism för förfarandet att omsätta ett objekts tillstånd till en informationsström, det som kallas för serialisering. Kodningsreglerna kan användas för utväxling av instanser tillhörande datatyper som är bestämda av program.
- En SOAP-variant av RPC som visar hur en konvention, med funktionaliteten att representera fjärranrop och svar, skall se ut.

Trots att SOAP är tillämpningsbar inom många sorters av meddelandesystem, och även kan använda sig av en mängd transportprotokoll så finner man ändå dess största frekvens gällande använde inom RPC (Remote Procedure Calls) skickade via protokollet HTTP. Att SOAP är plattformsoberoende innebär därför att en stor mängd av diverse program kan utbyta information, data etcetera (Cerami, 2002).

3.2.6.3 Axis

Axis är en open-source implementation av SOAP. Det är egentligen ett ramverk för att generera SOAP-processer som klienter, servrar, gateways med mera. I Axis ingår även en trivial fristående server, som i sin tur kan ansluta sig till servlet-motorer för exempelvis Tomcat, bra stöd för WDSL och hjälpande resurser för övervakning av TCP/IP-paket (Axis).

Följande förbättringar ingår numera i Axis jämfört med tidigare:

- Snabbhet
- Flexibilitet
- Stabilitet
- Komponentorienterat verkställande (deployment)
- Transportramverk
- Web Services Description Language (WDSL) – stöd (Axis)

3.2.6.4 UDDI

I Web Services protokollstack så är det UDDI som motsvarar det lager som används för att finna och publicera tjänster. Det är gjort av två delar (Cerami, 2002):

- UDDI är en teknisk specifikation för konstruktion av distribuerade bibliotek innehållandes Web Services och affärstjänster. Datan som finns i dessa bibliotek sparas som ett särskilt XML-format. API-delar som huserar inom UDDI-specifikationen frambringar möjligheten att söka efter befintlig data samt publicera ny sådan.
- UDDI Business Registry innebär en komplett implementering av alla UDDI:s specifikationer. Det innebär att registret kan tillhandahålla sökning efter befintlig UDDI-data för vem som helst och att registrering av företag och deras tjänster (services).

3.2.6.5 WDSL

I Web Services protokollstack så är det WDSL som motsvarar det lager som används för beskrivelse av tjänster. Enkelt och sammanfattande uttryckt kan man säga att WDSL är XML-grammatik som bestämmer hur ett publikt Web Services – gränssnitt skall vara (Li & Pahl, 2003). Gränssnittet kan ha följande innehåll:

- Information om alla publika funktioner.
- Datatypsinformation för alla XML-meddelanden.
- Bindningsinformation om det specifika transporteringsprotokoll som skall användas.
- Adressinformation för att kunna lokalisera en specificerad tjänst (service).

(Cerami, 2002)

Observera att ovan beskrivna tekniker endast är tillräckliga när det gäller enklare former av Web Services. Omfattande affärsutbyten kräver en mellan parterna överenskommen struktur för affärstransaktioner, transaktioner som förfrågas många gånger, scheman och dokumentflöden. En vanlig enkel SOAP implementering kan inte erbjuda detta. Då kan det vara bra att använda sig av ebXML, vilket är en samling XML-specifikationer, relaterade processer och beteenden designade för en infrastruktur för B2B-sammarbete och integration (Aoyama et al, 2002; Vawter & Roman, 2001).

Notera också att ovan beskrivna tillvägagångssätt endast är ett av många att få Web Services att fungera. Det finns även andra sätt men enligt Vawter och Roman (Vawter & Roman, 2001) så anses de beskrivna teknologierna, ej inkluderat Axis, vara de allra mest viktiga och de som kommer användas av flest intressenter.

3.3 Innan Webservices

I det här avsnittet skall jag ta upp lite hur det var innan Webb Services fanns. Detta för att ge en mer komplett bild av nyttoaspekten med denna teknologi och hur saker som den idag kan lösa tidigare bedrevs på ett mycket mer omfattande och komplicerat sätt. Nedan följer en kort redogörelse om just detta gällande de så kallade ’dot.net-bolagen’ och hur de konstruerade sina webbsajter innan de företagen i stort förpassades till att bli historia. Avsnittet avslutas sedan med en kort presentation av föregångaren till mjukvaruplattformen .NET. Men allra först några ord nedan om hur de två plattformar som den här uppsatsen handlar om direkt anknyter till denna historik.

J2EE har exempelvis historiskt sett varit en arkitektur för att bygga verkställanden på server-sidan samt att dessa har varit konstruerade i programmeringsspråket Java. Plattformen kan

användas till att bygga traditionella webbsidor, mjukvarukomponenter, eller paketerade applikationer (Vawter & Roman, 2001).

J2EE och .NET är utvecklingar av redan befintliga applikationsserver-teknologier som har använts för att bygga företagsanpassade program. De tidiga utgåvorna av dessa teknologier har historiskt sett inte nyttjats för konstruktion av Web Services. Nu när Web Services finns, så har båda plattformarna anpassat sig så att de även kan användas för att bygga just Web Services (Vawter & Roman, 2001).

3.3.1 Kort historik i exemplifierad form; Dot.Net-bolagen

När man ser tillbaka på den korta period som föregick de så kallade dot.net-bolagens undergång (cirka år1998-2000), så pekar många Internet-historiker på att dessa företag kopierade andras arbeten och verk. Mest härmades det när det gällde konstruktion av webbsajter. De här föråldrade sidorna hade som mål att ge deras besökare access till en enorm mängd av information; information, vilken när man ser tillbaka på det visade sig inte tillhöra företagens högsta grad av kompetens, utan snarare ett mål för att visa upp en häftig yta av företagen. Företagen som levererade denna brokiga mängd av information, vilken inkluderade väderleksrapporter, aktiekurser, nyheter, mail-tjänster, och mycket mera, tillverkade oftast inte densamma själva. Därför brukade de aktuella företagen köpa rådata och dess rättigheter för att återdistribueras i ett format som var tillgängligt för användarna. Efter att erhållit rådatan så kunde företagen tillverka dyra och högst tidsförbrukande program som konverterade rådatan till ett mer användarvänligt format, vanligtvis HTML. Att man dessutom använde sig av bland annat leasade anslutningar till Internet gjorde inte det hela billigare (Lurie & Belanger, 2002).

3.3.2 Windows DNA

Denna föregångare till dagens .NET började sin livsbana genom att Microsoft introducerade den år 1996. Windows DNA kan beskriva som en arkitektur vars syfte var att dela en uppgift eller ett program mellan många nätverksanslutna datorer. Ungefärligt simultant med Windows DNA så introducerades programmeringsspråket Java på marknaden av konkurrenten Sun. Det skulle enligt dem vara ett ypperligt språk för Internet med tanke på att samma mängd Javakod kunde implementeras på flertalet olika plattformar (Gustafson).

Windows DNA inkluderare vidare många väl utprovade teknologier som är i produktion idag, där ibland exempelvis Microsoft Transaction Server (MTS) och COM+, Microsoft Message Queue (MSMQ) samt Microsoft SQL Server-databasen (Vawter & Roman, 2001).

Nu till sist en förteckning av Microsoft DNA:s ingående delar. COM komponenter är vad som utgör den gemensamma faktorn för alla lager av Microsoft DNA (Britton, 2000).

Följande tjänster är delar av Microsoft DNA:s arkitektur (Britton, 2000):

- Presentationstjänster, där HTML, DHTML (Dynamic HTML), skript, ActiveX och COM komponenter ingår.
- Applikationstjänster där Internet Information Server (ISIS), COM+ och MSMQ ingår.
- Datatjänster där ADO, OLE DB (Object Linking and Embedding DataBase) ingår.
- Systemtjänster där filhantering, säkerhet, administration och nätverk ingår.

3.4 Vad är en mjukvaruplattform?

Eftersom det redan i denna uppsats har konstaterats att J2EE och .NET är mjukvaruplattformar så skall vi därför i detta avsnitt närmare ta reda på vad det innebär. Den typ av mjukvaruplattform som J2EE och .NET tillhör härstammar från en arkitektur som kallas treskiktsarkitektur och kan läsas om nedan och även dess föregångare, den tvåskiktade arkitekturen. J2EE och .NET hör hemma i denna treskiktade arkitektur och närmare bestämt i det mellanliggande skiktet. Därför benämns de ofta som så kallade 'middleware' (Peterson, 2002), vilket ungefärligen kan översättas till program i mellanskiktet. Just middleware skall den avslutande delen av dessa avsnitt handla om. Avsnittet skall dock allra först inledas med en genomgång av själva arkitekturbegreppet för att klargöra detta innan man går vidare på mer specifika tillämpningar av detsamma.

3.4.1 Arkitekturövergripande

Själva ordet arkitektur betyder enligt Nordstedts Svenska Ordbok "läran om samspelet mellan tekniska och konstnärliga faktorer vid byggande" (Nordstedts Svenska Ordbok, 1990). En Arkitektur kan beskrivas som den övergripande formulan eller strukturen av ett system om man så vill. Den visar hur ett system är uppbyggt och organiserat. Arkitekturen fungerar även som ett ramverk för systemet samt identifierar dess större komponenter samt kommunikationen mellan dessa (Sommerville, 2001). En arkitektur kan ha en mängd olika utformningar. Den kan exempelvis vara datacentrerad, dataflödesbaserad, objektorienterad eller uppdelad i lager (Sommerville, 2001). Det är alltså den sistnämnda modellen som tillämpas av J2EE och .NET.

Det finns sällan en enda modell som är tillräcklig för att lösa de problem som ett system skall göra. Istället är det vanligt att man väljer den modell som bäst passar in och sedan modifierar denna tills den passar de syften man eftersträvar. Arkitekturmodeller är till sin natur generella och därför finns det ingen som från början är helt anpassat till en aktuell situation. (Sommerville, 2001)

Olika arkitekturmodeller har olika svagheter och styrkor. Diverse känsliga punkter i arkitekturmodellen läggs det ofta tyngd vid och påverkar det aktuella systemet i form av snabbhet, säkerhet, distribution och underhåll i ett system Därför är det viktigt att vid val av modell identifiera dessa i ett tidigt skede (Sommerville, 2001).

Bör väl även för helhetsintryckets skull nämnas att större system består vanligtvis av mer än en modell, där till exempel varje delsystem kanske använder sin egen modell (Sommerville, 2001).

3.4.2 Tvåskiktad arkitektur

Den tvåskiktade arkitekturen är föregångaren till den treskiktade och är med i denna uppsats för att man skall kunna följa utvecklingen av arkitekturer. Tanken är sedan specifikt för denna uppsats att från ovanstående definition gå via denna form av arkitektur till den senare nedanstående. Den tvåskiktade arkitekturen är den enklaste formen av så kallad klient-serverarkitektur, som sålunda benämns som tvålagers klient-serverarkitektur. En applikation i denna arkitektur agerar som en server med en uppsättning av klienter. Arkitekturen kan vidare förgrenas i två varianter. Den första kallas för tunna klient modellen och innebär att all logik och datalagring med mera finns på serversidan medan klienten endast står för presentation av data. Den andra varianten kallas för fet klient modellen och här har serverns inflytande minskas och den ansvarar endast för datahantering medan programmets logik implementeras

mjukvarumässigt av klienten samt handhar kontakten med användare av systemet (Sommerville, 2001).

I den tvåskiktade arkitekturen så finns det inget enskilt lager för affärslogiken, det infördes senare av den treskiktade arkitekturen, utan den förekommer både hos klient och server. Klientens huvudsakliga uppgift i den här formen av arkitektur är att bevara ett gränssnitt mot användarna så att dessa kan kommunicera med servern (Jones, 2001). I den här typen av arkitektur så jobbar klienten direkt mot en resurs som till exempel en databas (Krakowski & Dahlgren, 1998).

Denna typ av arkitektur har sitt lämpligaste användningsområde inom enklare miljöer så som mindre nätverk. En av fördelarna är att endast klientkommunikation är möjlig vilket medför att man besparar sig från att investera i ytterligare dyra servrar. Även hantering av kostnad för protokoll faller bort. Administrationen är enkel på grund av att data för det mesta lagras centralt.

Däremot så är arkitekturen mindre väl lämpad för större nätverk. Administrationen försvåras avsevärt med den här modellen bland annat på grund av att varje klient måste konfigureras för varje server och då går det inte att samtidigt dela arbetet mellan flera servrar. Så skalbarheten är på så sätt ganska begränsad (Jones, 2001). Även faktumet att logiken till stor del finns hos klienten bidrager till den här arkitekturens nackdel, inte minst för att klienterna då brukar bli väldigt tjocka (Krakowski & Dahlgren, 1998).

3.4.3 Treskiktad arkitektur

Med den tvåskiktade modellen kom också en del problem så som att när det gäller varianten med den tunna klienten så erhöll man skalbarhets- samt prestandaproblem och anledningen till det var att klienten tog kraft från servern. Problemen men den andra varianten, den med fet klient, var av administrativ karaktär. Med den feta klientmodellen så måste man nämligen vid exempelvis en uppdatering uppdatera samt implementera den nya versionen av ett program på alla klienter, vilket kan ta tid och tid innebär som bekant pengar (Sommerville, 2001).

För att få ordning på de problem som den tvåskiktade arkitekturen innebar så skapade man en ny arkitektur med ytterligare ett skikt. I denna arkitektur så innebar varje skikt en mer bestämd typ av funktionalitet, nämligen: presentation, affärsregler (logik) samt datahantering och lagring. Den stora skillnaden är att man har centraliserat logiken till ett eget lager och därmed tagit den logik som fanns på server respektive klient och placerat den i detta lager. Fördelar som det innebär är bland annat lättare handhavande och då inte minst administrativt samt enkelheten i att byta ut enskilda komponenter (Kajbrink & Lorentsson, 1999).

3.4.4 Definition av Middleware

Middleware uppkom som en lösning när företag ville transportera information mellan stordatorer, databaser och användarterminaler på åttiotalet. Använt i sin renaste form så är middleware ett program som möjliggör informationstransport från en applikation till en eller flera andra simultant med isolering av utvecklaren från beroenden mellan kommunikationsprotokoll, operativsystem och hårdvaruplattformar (Talarian refererad i Balic, Björklund & Jägmark, 2001).

Middlewares klassiska arbetsuppgift har varit att agera mellanhand åt plattformar och inkompatibla databaser. Fördelen med det har varit att man har som utvecklare kunnat koppla nämnda plattformar och databaser till middlewarelagret och låta det sköta överföringar och

dataöversättningar. Idag har middlewarelagrets roll utökats mer än till att vara ett kitt mellan inkompatibla system. Middlewarelagret kan nu agera datatransportör både internt i en organisation samt mellan organisation och Internet (Talarian refererad i Balic, Björklund & Jägmark, 2001).

3.4.5 Middlewaretyper

Det finns olika typer av Middleware som har olika funktionalitet och användningsområden. Ett av de vanligaste är följande:

3.4.5.1 Application-Server Middleware

Application-Server Middleware har mångt mycket annat som användningsområden än att isolera datasystems affärslogik, detta med hjälp treskiktsapplikationer. En definition är:

> *”Ett program som körs på en maskin av medelstorlek och hanterar applikationsoperationer mellan browser-baserade klienter och en verksamhets ursprungliga affärsapplikationer eller databaser”.*
>
> (Webopedia 1999 citerad från Balic, Björklund & Jägmark, 2001)

3.4.6 Användning av middleware

Över åren har omfattande summor investerats i de IT-system som vi har idag. Men ett problem med dessa är att de är långt ifrån alltid, sett ur en arkitektonisk synvinkel, överensstämmande med varandra. De kan bland annat vara upp till tjugo eller trettio år gamla och bestå av en mängd olika operativsystem och tekniska plattformar. Delsystemen i en sådan kontext är ofta många liksom utvecklandet av så kallade punkt till punkt kopplingar. Detta har som konsekvens att exempelvis dubbellagring av data, så kallad redundans uppkommer och även så kallad inkonsistent data. Strukturen är sålunda komplex och svåradministrerad i sådana system (Fisher, refererad i Balic, Björklund & Jägmark, 2001).

Företag börjar därför allt mer rikta sitt intresse mot hur de skall komma till rätta med de problem som ovan beskrevs. De eftersträvar system som kan ge dem frigörelse av exempelvis var datan är lagrad och dessutom på ett uniformt sätt samlar det perspektiv som företaget vill presentera utåt En Middlewarelösning kan åstadkomma detta genom att få program att interagera på ett sätt som utgör ett stöd för verksamheten snarare än ett ok. Dock kräver varje organisation på grund av sin särart en grundlig analys för att man skall finna den lösning som passar just dem (Fisher, refererad i Balic, Björklund & Jägmark, 2001).

Slutligen så har dessutom allt eftersom datasystem blivit än större och alltmer komplicerade även problematiken gällande att designa dem ökat. Förut kunde det enbart handla om datastrukturer och funktionalitet men numera kan det även innefatta strukturen gällande hela systemet (Garlan & Shaw, 1994).

3.5 Microsoft.NET

Efter genomgången ovan av arkitekturer samt middleware-begreppet så är det nu dags att ta itu med den första av de två mjukvaruplattformar som jag har valt att återge i denna uppsats: Microsofts.NET. .NET är för övrigt ett exempel på en plattform som nyttjar den nyss beskrivna treskiktsarkitekturen samt använder sig av en den typ av tjänster som jag har valt som ett annat delområde för den här uppsatsen: Web Services. Web Services tas upp lite längre fram men nu först en genomgång av .NET.

3.5.1 Definition av .NET

Faktumet är att datavärlden idag består av en enorm mängd operativsystem, programmeringsspråk och standarder utan kompatibilitet med varandra. Det har bidragit till stora problem och en liknande mängd kostnader. .NET är tänkt att kunna bryta dessa skiljelinjer, men vad är egentligen .NET? I detta inledande avsnitt hoppas jag kunna besvara den frågan.

Det är uppenbart att .NET är något som är svårt att enkelt definiera, vilket detta förkortade pressuttalande från en pressrelease av Microsoft värnar om:

> .NET representerar en uppsättning, en miljö, en infrastruktur för programmering som stödjer nästa generation av Internet som en plattform. Det är en miljö som gör det möjligt […] .NET är också en användar-miljö, en uppsättning av grundläggande användartjänster som finns på klienten, i servern, i molnet, som är förenlig med och bygger på den programmeringsmodellen. Så det är både en användarupplevelse och en uppsättning av upplevelser för utvecklare, det är den konceptuella beskrivningen av vad .NET är (Hillesley, 2001).

.NET kan beskrivas som många olika saker beroende på vad det egna perspektivet är. En del ser .NET som Microsofts nya Visual Studio-utvecklingsmiljö. Några ser det som ytterligare ett programmeringsspråk (C#). Medan andra uppfattar det som ett nytt ramverk för data och meddelanden baserat på XML (eXtensible Markup Language) och SOAP (Simple Object Access Protocol) (Farley, 2000). Uppfattningen finns även att .NET i stort är en omskrivelse av Microsofts tidigare plattform för att utveckla så kallade enterprise program: Windows DNA (Vawter & Roman, 2001).Här är hur som helst en lista över de tekniska komponenter som enligt Farley utgör .NET –plattformen (Farley, 2000):

- **C#**, ett ’nytt’ språk eller egentligen en Java klon (Hillesley, 2001) som förut gick under namnet Cool. Med C# kan man skriva klasser och komponenter som integrerar element skrivna i programspråken C, C++ och Java. Dessa kan dessutom addera extra funktionalitet, som metadata – taggar, som relaterar till komponentutveckling.

- En ’**Common Language Runtime’** (**CLR**), vilken exekverar byte-kod i ett ’Internal Language’ (IL) format. Kod och objekt som är skrivna i ett format kan, skenbart, kompileras med en ’IL runtime’, så snart som en IL kompilator har blivit utvecklad för det aktuella programspråket.

- En uppsättning grundläggande komponenter (**base components**), som är tillgängliga från CLR:n, som tillhandahåller olika funktioner (funktioner för nätverk, containers etcetera)

- **ASP+**, en ny version av ASP (Active Server Pages) som stödjer kompilering av ASP:s i CLR och därför skriver ASP skript som använder vilket språk som helst med en IL anknytelse.

- **Win Forms och Web Forms**, nya ramverk för UI (user interface)-komponenter vilka är tillgängliga från Visual Studio.

- **ADO+**, en ny generation av ADO-dataaccess komponenter som använder XML och SOAP för datautbyte (Farley, 2000).

Tyngdpunkten gällande .NET har flyttats från faktumet att användare skall 'prata' med ett program till att program kommunicerar med andra program (Johansson & Ledin, 2001).

.NET kan betraktas som en standard för beskrivning av data. Den kod som är i överensstämmelse med denna standard kallas således för .NET kompatibel kod. Resultatet av detta blir att koden i fråga kan integreras med valfritt programspråk under förutsättning att det språket följer denna standard. Därför kan ett objekt i C++ vara skapat från en klass i Visual Basic (Johansson & Ledin, 2001).

Kommunikationen mellan komponenter i .NET sker på grund av dess distribuerade miljö enbart med XML-baserade protokollet SOAP. I klartext innebär det att man exempelvis kan skicka en förfrågan via HTTP- (HyperText Transfer Protocol) protokollet och få ett SOAP-formaterat svar tillbaka (Johansson & Ledin, 2001).

3.5.2 .NET:s innebörd

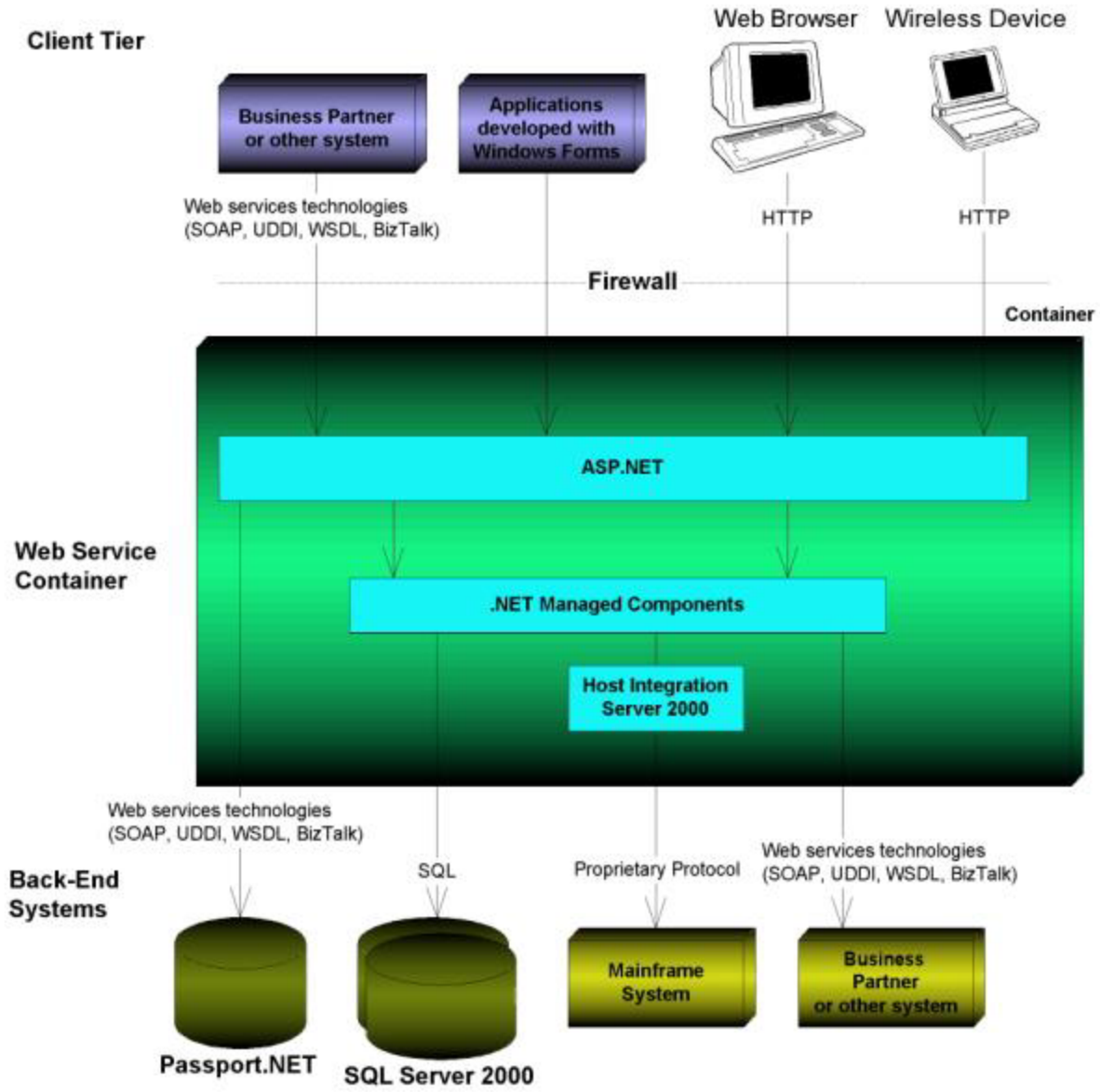
Den grundläggande innebörden av .NET är att man med hjälp av en standard kan möjliggöra integration mellan olika program och Web Services. Vidare kan det här tolkas som ett nytt slags operativsystem om man så vill där Internet är grunden för det hela och där man inte är låst på något sätt till ett enda program eller dess funktionalitet. Det innebär att man kan blanda friskt för att på så sätt kunna få fram bästa möjliga service åt användarna. Datan kan då sägas existera på Internet istället för på någon specifik hårdvara och även åtkomsten av densamma är oberoende av vilken hårdvara du nyttjar lokalt. All denna funktionalitet transporteras ut på nätet med hjälp av protokollet SOAP (Johansson & Ledin, 2001).

Ett exempel på ovanstående frihet speglas i .NET:s stöd för programmeringsspråk. År 2001 hölls det på att ta fram .NET-stöd för tjugoen programmeringsspråk, bland annat Eiffel, Python, Smalltalk, Cobol, Pascal och Oberon (Lindberg 2001, s. 137). Idén är att kommunikation mellan olika språk och plattformar med tillhörande datautbyte skall kunna äga rum smärtfritt. .NET-konceptet grundar sig på det distribuerade system paradigmet, vilket innebär att data behandlas där så anses lämpligast (Microsoft Whitepaper refererad i Johansson & Ledin, 2001; Watkins 2000).

Ytterligare en central del i .NET-konceptet gäller komponenter som är tänkta att finnas distribuerade på Internet, så kallade Web Services. Som nämnts ovan så innebär det då att data och applikationer kommer i framtiden ligga spridda på Internet. En möjlighet med detta är att man kan då hyra tjänster eller programvaror istället för att köpa dem. Gränserna mellan vad som är lokala program eller distribuerade kommer därför att försvinna. (Erlanger, 2001; Lindberg, 2001 s. 137)

3.5.3 .NET:s arkitektur

Nedan så förevisas i figur 3:1 en utvecklingsmodell över Microsoft.NET gällande Web Services och hur det fungerar. Den följs nedan även av en övergripande förklaring av figuren.



Figur 3:1. Utvecklingsmodell över Web Services med .NET.

(Källa: Vawter & Roman, 2001)

Kortfattat så kan ovanstående figur förklaras enligt följande:

.NET programmet huserar i en så kallad container, vilken förser den grad av service som är nödvändig för så kallade enterprise program som till exempel, transaktioner, säkerhet och meddelande tjänster.

Affärslagret av .NET-programmet är byggt genom användandet av .NET-hanterade komponenter. Det här lagret utför affärsprocesser samt datalogik. Det ansluter till databaser användandes Active Data Objects (ADO.NET) och existerande system som använder sig av tjänster som kommer från Microsoft Host Integration Server 2000, så som exempelvis COM Transaction Integrator (COM TI). Det kan även ansluta till affärspartners nyttjandes Web Services teknologier som SOAP, UDDI (Universal Description, Discovery and Integration protocol) eller WDSL (Web Services Description Language).

Affärs partners kan ansluta till .NET programmet via Web Services-teknologier som SOAP, UDDI, WDSL och BizTalk.

Traditionella 'tjocka' klienter, webbläsare, trådlösa apparater kopplade till Active Server Pages (ASP.NET) vilka i sin tur begagnar användarinterface i HTML (HyperText Markup Language), XHTML (Extensible Hypertext Markup Language) eller WML (Wireless Markup Language). Mer avancerade användargränssnitt byggs i Windows Forms (Vawter & Roman, 2001).

3.5.4 XML-webbtjänsters tillgänglighet på .NET plattformen

Utveckling av XML-webbtjänster har redan tagit fart och tusentals utvecklare sysselsätter sig med detta och tar fram dessa med hjälp av .NET och Visual Studio.NET. Det skapas även dagligen tjänster som tas fram med SOAP och XML. Till hands kommer även att finnas en mängd viktiga byggstenstjänster för förenklande av utvecklingen beträffande .NET-plattformen (Microsofts svenska hemsida 1).

Microsoft håller även på att ta fram en grundläggande uppsättning Web Services som kan kombineras med andra webbtjänster eller användas direkt med så kallade 'smarta' klientapplikationer. Ett exempel på dessa tjänster är The Microsoft MapPoint Web Service vilket tillåter dig att integrera högkvalitativa kartor, och annan lokationsintelligens i dina applikationer, affärsprocesser, och webbsidor (Microsofts svenska hemsida 2a).

3.5.5 Nygammalt sätt att kompilera kod i .NET

.NET är innovativt på så sätt att det inte kompilerar applikationer i något eget kod-format, det vill säga, det kompilerar inte applikationer till exempelvis Intelspecifik kod. Istället är nu kompileringen en två-stegs process. Den kod som är skriven av en utvecklare kompileras nu till formatet Microsoft Intermediate Language (MSIL). Därefter så kompilerar exekveringsmiljön Common Language Runtime (CLR) koden till applikationsspecifik kod vid exekvering. Om det här låter bekant beror det på att Microsoft har inspirerats av Java. C# som inkluderas under .NET-plattformen bär en högst tydlig likhet med Java. Microsoft har också bifogat ett program som konverterar Javakod till C#.

Rykten har även gjort gällande att utvecklare håller på och arbetar på en Linux CLR. Om dessa rykten skulle vara sanna, så skulle CLR:n kunna tjäna som ett sätt att tillåta .NET-utvecklade applikationer access till exekvering på Linux-plattformen (Lurie & Belanger, 2002).

3.5.6 Microsoft.NET:s produkterbjudande

I produkterbjudandet från Microsoft så ingår .NET-plattformen och som en del av denna även .NET-miljön. Skall inleda med att beskriva vad .NET-plattformen innebär.

Enligt vad som står på Microsofts svenska hemsida (Microsofts svenska hemsida 1) så finns det i .NET-plattformen följande nedanstående tjänster/verktyg för att utveckla (verktyg-.NET-miljön), hantera (servrar) och använda (byggstenstjänster och smarta klienter) XML-webbtjänster:

3.5.6.1 Smarta klienter: programvara

Smarta enheter är PC:s, laptops, arbetsstationer, telefoner, hand-datorer, Tablet PC:s, med mera. Vad det är som gör dessa enheter 'smarta' är deras förmåga att nå access till Web

Services, vilket gör att man kan få tillgång till sin data oavsett var man befinner sig eller vilken typ eller mängd enheter du nyttjar (Microsofts hemsida a).
Specifikt för .NET gällande dessa 'smarta' enheter så tar man hjälp av för syftet anpassad mjukvara för att enheterna skall kunna jobba tillsammans med .NET. Enligt Microsoft (Microsofts svenska hemsida 1) så innebär de 'smarta' enheterna vissa fördelar i följande miljöer:

Personlig. Påstås underlätta det egna arbetet genom att användandet av den så kallade .NET-identiteten, den egna profilen och data. Känner av närvaro vid dator och meddelanden anpassas följaktligen.

Nätverk. Känner av bandbreddsbegränsningar, stöd för on- samt offlineanvändning av program och känner igen tillgängliga tjänster.

Information. Kan nå access till och analysera data oavsett var man befinner sig och vid vilken tid som helst.

Andra enheter. Känner igen och ger information om datorer, andra 'smarta' enheter, servrar och Internet. Kan skicka vidare tjänster till andra enheter samt på ett smidigt sätt komma åt information på en dator.

Program och tjänster. Applikationer och data presenteras alltid på mest lämpliga sätt för enheten. De metoder för anslutning och indata som är bäst för aktuell kontext aktiveras. För webbtjänster används teknikerna XML, SOAP och UDDI. Programmering och utökning är möjlig för enheterna, då gjorda av en utvecklare (Microsofts svenska hemsida 1).

3.5.6.2 Byggstenar för webbtjänster: .NET My Services/Hailstorm
Teknologin som först kallades för 'Hailstorm' var baserad på standardprotokollen XML och SOAP. Det var en öppen standard. Tanken var att de skulle kunna anropas av alla program, tjänster och verktyg oavsett operativsystem (Windows, Apple, Macintosh, Pocket PC, Palm, Linux och olika former av UNIX), objektmodell eller programmeringsspråk (Microsoft svenska hemsida d, 20010312).

En grundläggande tanke med tjänsten var att den skulle kunna erbjuda privatpersoner att spara alla sina data på Microsofts Servrar (Lotsson, 20020411). En annan grundläggande tanke med tjänsten var att en enskild användare av Internet skulle bara behöva identifiera sig en gång per session oberoende av hur många eller vilka sajter eller tjänster han besöker (Kempe, 20030311). Tjänsten skulle vidare kunna ta hand om (Lotsson, 20020411) åt användare e-post, dennes adressbok, kalender och dylika dokument från vilken dator som helst bara den var ansluten till Internet. Men det var också tänkt att tjänsten skulle erbjuda så mycket mer än bara så. Den skulle fungera som en mäklare för Internetbaserade tjänster. My Services skulle tagit hand om inbetalningar och inköp. Även informationen som tjänsten skulle tillhandahålla skulle underlätta affärer, resor och nöjesliv.

Men den 10 april 2002 (Lotsson, 20020411) så fick Microsoft ge upp. Det som fick den på fall var de tänkta kundernas motvilja mot att lägga sina privata data och uppgifter för att inte tala om ekonomi hos Microsofts servrar. Trots att tjänsten är nerlagd så kommer Microsoft troligtvis att sälja tjänsten till företag som vill driva den själva.

Värt att nämna i sammanhanget är att det även finns ytterligare ett alternativ för den som är sugen på att använda sig av en tjänst som My Services. Sun erbjuder en motsvarande tjänst som kallas för Liberty Alliance. Den stora skillnaden mellan Suns tjänst och Microsofts är att i Sun:s fall så skall inget enskilt företag ha hand om all information (Kempe, 20030311).

3.5.6.3 .NET-servrar
Den distribuerade datamodell som Web Services representerar vill öka kraven på servrars infrastruktur. Det vill säga att alla de kombinationer som görs möjliga när Web Services kan kommunicera med varandra och även erbjuda tjänster på Internet kommer således innebära en helt ny form av krav på servrar (Microsofts hemsida b).

För att bemöta detta så kommer Microsoft att erbjuda servers som är anpassade för dessa villkor. Enligt dem så kommer skalbara servers att på ett mycket ingående sätt vara integrerade med XML och skydd av dessa teknologier kommer samtidigt att erbjudas genom industristandardiserade teknologier (Microsofts hemsida b).

Speciell anpassad mjukvara kommer att finnas för att tillgodose ovan beskrivna tillhandahavande (Microsofts hemsida b).

3.5.6.4 .NET-miljön
.NET-miljön som ingår under .NET-plattformen, består av verktyg för interaktion med XML-tjänster. Miljön kan definieras som en blandning av XML-webbtjänster och lokal programkod när sådan anses nödvändig. Några av de kända Microsoft-program i som kommer att flyttas över till .NET-miljön är: Microsoft Office, MSN, Microsoft bCentral Small Business Portal samt Microsoft Visual Studio Development System (Microsofts svenska hemsida 1).

3.5.6.5 Utvecklingsverktyg i .NET-miljön
Med Microsoft Visual Studio.NET och Microsoft.NET Framework så kommer utvecklare att snabbt kunna konstruera Web Services och integrera dem med andra program. De flesta utvecklare kan fortsätta att utveckla sina existerande kunskaper, detta på grund av att .NET Framworks CLR tillåter dem att utveckla Web Services användandes vilket modernt programmeringsspråk som helst (Microsofts hemsida c).

3.6 Sun Soft J2EE
Efter presentationen av .NET så skall vi nu gå över till en genomgång av den störste konkurrenten till .NET, nämligen mjukvaruplattformen J2EE från Sun. Även J2EE inkluderas numera med Web Services och passar därmed in under de områden som uppsatsen skall behandla. Genomgången börjar dock med en introduktion för att sedan följas av en övergripande beskrivning av plattformen där bland annat en rad teknologier samt dess struktur presenteras. Kapitlet avslutas med att följande tre områden behandlas: Hur man kan arbeta med och utveckla systemet det vill säga programmering, frågor kring den extra funktionalitet som många javaåterförsäljare adderar till plattformen samt till sist en kort presentation av det utvecklingsspråk som genomsyrar hela plattformen framför alla andra: Java. Men som sagt var vi inleder allra först med en introduktion.

3.6.1 Introduktion
Java 2 Plattformen, Enterprise Edition (J2EE), är designat för att förenkla komplexa problem gällande utveckling, implementering och skötsel av multi-skikts enterprise lösningar. J2EE är

en industristandard, och är resultatet av ett stort industriellt initiativ lett av Sun Microsystems (Vawter & Roman, 2001).

Det verkar som om även de som förstår Web Services ofta missförstår J2EE. Förutom utvecklare, så tror de flesta att J2EE är en produkt som man kan köpa från Sun, precis som man gör från Microsoft med .NET. Tvärtom, J2EE är en uppsättning av specifikationer, där varje sådan dikterar hur olika J2EE funktioner måste operera. Till exempel, Java Transaction Service (JTS) specifikationen identifierar hur man bygger en tjänst som tillåter distribuerade transaktioner (Lurie & Belanger, 2002).

Inte för att det var med Web Services i åtanke som J2EE:s Servlet teknologi utvecklades, men den tillåter ändå Web Services-utveckling. En så kallad Servlet utför all processering, inkluderandes anrop av Enterprise JavaBeans (EJB:s) som sedan returnerar data till servleten. Servleten formulerar sedan ett svar som paketeras i XML-format för att därefter returnera det till klienten. Suns nya Java Web Services Developer Pack (WDSP) kan erhållas genom en enstaka nerladdning och innehåller allt som behövs för utveckling av Web Services. Det innehåller Java XML Paket, vilket befriar utvecklare från mycket låg nivå-analyserande, plus Java Server Pages (JSP) Standard Tag Library, Java WDSP Registry Server och Tomcat Java Servlet och JSP container (Lurie & Belanger, 2002).

3.6.2 J2EE-en övergripande beskrivning

Java 2 Enterprise Edition (J2EE) är en plattform som man kan tillverka avancerade program med. J2EE är en Java-variant vars tillämpningsområde är servrar och EJB (Enterprise Java Beans) är förteckningen över hur tillämpningar skall genomföras (Computersweden, 2001). Komponentstandarden Java Beans ligger som grund för EJB och EJB:s syfte är att på ett tillförlitligt sätt inkapsla affärslogik på servrar (Computersweden, 1997). J2EE visar istället på en standard att konstruera affärsapplikationer av den flerskiktade sorten. Logiken bakom J2EE är att simplifiera företags programvara genom att grunda dem på standardiserade och modulära komponenter. Syftet är även att erbjuda fullständig service åt dessa komponenter. Parallellt med denna strävan efter service så vill man reducera svår programmering genom att hantering av flertalet applikationsdetaljer sker automatiskt (Java 2 Plattform, Enterprise Edition (J2EE) Overview, 2002).

En hörnsten hos J2EE är kompatibiliteten. För att åstadkomma detta får tillverkare och programmerare med flera som betalar licens för Java utföra särskilda tester av de produkter som de framställer för att på så sätt kunna konstatera kompatibilitet med J2EE (Jacobsson, 1999).

J2EE stödjer fullständigt följande tekniker: Enterprise Java Bean (EJB)-komponenter, Java Servlet API (Application Program Interface), Java Server Pages (JSP), och XML. Inom J2EE finns det inkluderat totala specifikations- och uppfyllelsekrav för säkerställelse av möjligheten att samma program skall kunna anpassas med en rad olika affärssystem som kan stödja J2EE (Java 2 Plattform, Enterprise Edition (J2EE) Overview, 2002). Nedan skall denna övergripande beskrivning av J2EE-plattformen gå genom några av de tekniker som nyss har nämnts samt några övriga som är relevanta för J2EE samt även sist i avsnittet granska dess struktur, för att på så sätt skapa en första inblick i J2EE:s uppbyggnad och funktionalitet.

3.6.2.1 Java böna

Java-bönor (Java Beans) finns både som ett paket i Java (java.beans) och som en specifikation i form av ett dokument som åskådliggör tillhandahavandet av klasser och gränssnitt i paketet

java.beans. Just paketet java.beans ger även funktionalitet vid framställning av bönor (Johnson, 1997).

En java-böna är en återvinningsbar mjukvarukomponent vilket betyder att dess struktur är ämnad för utvecklare att kunna konstruera autonoma mjukvarukomponenter som sedan kan sammansättas till större programkomponenter. Just det här med att ha fristående bönor som är lätta att byta och bygga program med dem är grundtanken bakom bönorna (Zukowski).

3.6.2.2 Enterprise Java Bean (EJB)
Enterprise Java Beans (EJB) är ett ramverk av objektorienterad natur samt avsett för flerskiktade spridda system och sammankoppling med program (Bergquist & Ericsson, 2001).

Inkapslingen av ett programs affärslogik sker dessutom med en serverbaserad komponent (The J2EE Tutorial – What Is an Enterprise Bean, 2002).

Filosofin bakom EJB är återvinningsbara komponenter som huserar i ett mellanskikt och som skall kunna föra samman klienter och bakomvarande system, så kallat ’back-end’. Exempel på detta är plattformar och databaser (Hemrajani, 1999). Strukturen hos EJB är av tre skikt (Sundberg, refererad i, Arnoldsson, Lupander & Jönsson, 2003): presentationsskiktet, affärslogikskitet och dataskiktet. Inom den strukturen kan man stöta på följande begrepp som exempelvis (Ljunggren & Viker, 2001):

- EJB-Server: Applikationsserver enligt J2EE förteckningen.
- EJB Container: Stället för körning av Enterprise Beans.
- Enterprise Java Bean: Exempel på detta: entitetsböna, sessionsböna, message-driven böna
- EJB Home: Objekt som finner och konstruerar en EJB.
- EJB Objekt: Klientkontakt med affärsmetoder

3.6.2.3 EJB Container
EJB containern utgör en mycket vital del av EJB strukturen med tanke på att en enterprise böna inte kan existera utanför denna. Funktionaliteten mellan en enterprise böna och en container är analogt med hur exempelvis en webbläsare hanterar en Java applet (jGuru, 2000).

Containerns uppgift är att konstruera nya instanser av böner och även se efter att dess lagring på servrar utförs korrekt av just servrarna (Bergquist & Ericsson, 2001). När det kommer till exekvering av bönan så är containerns ansvarsområden följande: fjärråtkomst av bönan, säkerhet, fortlevande, transaktioner, konkurrens, samt access till och samordning av resurser. Stor del av funktionaliteten abstraheras av containern, det vill säga sker automatiskt, så att programmeraren kan koncentrera sig på affärslogiken och containern på den övriga logiken (Chen et al, 2002; jGuru, 2000).

Som en säkerhetsaspekt vid anrop från ett klientprogram så kapslar containern in enterprise bönan från direkt åtkomst med applikationen. Detta för att containern först skall kunna kontrollera att anropet har gått korrekt till (jGuru, 2000).

3.6.2.4 Remote och Home interface
Livscykeln för en böna företräds av ett så kallat home interface och bönans affärsmetoder av ett remote interface. Vid klientanrop agerar ett home interface som en global, världslig, portal för denna. Genom detta home interface får klienten tillgång till bönans remote interface och därmed dess metoder (Sundberg refererad i Arnoldson, Lupander & Jönsson, 2003).

3.6.2.5 Entitetsböna
Entitetsbönan (entity bean) är en symbol för ett affärsobjekt inom ett affärssystem och det finns flertalet sätt att känna igen en sådan (The J2EE Tutorial – What Is an Enterprise Bean, 2002): En entitetsböna är icke-förgänglig och kan därmed lagras, ofta i en databas. Bönan kan ha flera klienter och dessa kan förändra en och samma mängd av data. De är också försedda med primärnyckel samt kan ha förhållande med en annan entitetsböna.

3.6.2.6 Sessionsböna
Sessionsbönor (session bean) har hand om interaktionen mellan entitetsbönor och andra sessionsbönor. Även generella uppgifter från klienter och resurstillgång handhas av sessionsbönan. Sessionsbönan är ett förgängligt affärsobjekt vilket betyder att den inte beskriver data på samma sätt i en databas som en entitetsböna (jGuru, 2000).

3.6.2.7 Message-driven bean (MDB)
Message-driven bönor (MDB) är konstruerade för att kunna hantera asynkrona program (The J2EE Tutorial - What Is a Message-Driven Bean?, 2002). Message-driven bönors stora skillnad från entitetsbönor är att de består bara av en enda bönklass. Detta omöjliggör klientkontakt via ett interface som med till exempel sessionsbönor.

3.6.2.8 Java Management Extensions (JMX)
JMX gick tidigare under namnet Java Management API (JMAPI). Det är en arkitektur med java-bas och dito service som är inriktad på program- samt nätverksbehandling (Sullins & Whipple, 2002). Genom nyttjande av JMX-specifikationen kan det konstrueras webbaserade, distribuerade, dynamiska och modulära lösningar för hanterande av java-baserade resurser. De befintliga nätverkslösningar som finns i aktuellt system kan även integreras (Java Management Extensions (JMX) Specifikation; Newport).

JMX beskriver en standard för att konstruera JMX-objekt som kallas för Management Bean, MBean. En MBean finns inuti en container som är definierad av standarden och möjliggör för en JMX-klient att anropa metoder och därmed erhålla access till attribut hos en MBean (Newport).

3.6.2.9 MBean
Management Bean, Mbean, är ett java-objekt som använder sig av särskilda gränssnitt, ’interface’, som är styrda av vissa designmönster samt man kan med hjälp av dem bearbeta hanteringen och kontrollen av resurser (AdventNet Inc refererad i Lupander Arnoldson & Jönsson, 2003; SpiritSoft). Genom MBean:s interface får man den nödvändiga information som är behövlig för att kunna administrera, hantera och styra den (SpiritSoft).

Java Management Extension (JMX) beskriver fyra olika sorter av MBeans, där varje sort beskrivs genom, exempelvis, implementering av vissa interface eller att den rättar sig efter särskilda namnkonventioner (SpiritSoft).

MBeans kan förse både Java- och icke java-program med administrering, hantering och styrning (Orbism refererad i, Lupander Arnoldson & Jönsson, 2003).

De fyra olika sorterna av MBean är: Standard, Dynamic, Open samt Model. De olika sorterna är situationsanpassade genom att varje sort motsvarar olika sorter av administrerings-, hanterings-, samt styrningsbehov (SpiritSoft).

Standard MBean är mest simpel gällande konstruktion och implementation. Passar bäst i kontexter där hanterings- och resursgränssnittet gällande en resurs är stabilt (SpiritSoft).

Dynamic MBeans kräver att Dynamic MBean interfacet implementeras samt att dess metoder och attribut visas vid exekvering. Detta skall ske genom att vid anrop förse dem som anropar med beskrivande metadata. Är mest lämpligt i motsats till Standard Mbean när hanterings- och resursgränssnittet gällande en resurs är föränderligt samt sammanslagning av resurser som inte från början är överensstämmande med varandra (SpiritSoft).

Open MBeans är dynamiska och nyttjar endast så kallade primitiva wrappertyper och några av dess attribut och metoder. Vid exekvering kan dessa hittas och brukas av vilken klient som helst. Inga extra JAR-filer (ungefär som en ZIP-fil) är nödvändigt vid användning (SpiritSoft).

Model Beans även de dynamiska med egenskaperna att de kan beskriva sig själva samt är totalt konfigureringsbara vid exekvering. De utgör ett dynamisk sätt att styra resurser och får till sin hjälp för detta en allmän MBean-klass med ett så kallat default (från tillverkaren inställt)-beteende (SpiritSoft).

Gemensamt för alla MBeans är att de visar sina attribut och metoder för att på så sätt kunna tillåta att resurser som är representerade av en MBean bli kontrollerade av hanterings- och styrningsapplikationer (administrationsapplikationer) (SpiritSoft).

3.6.2.10 Applikationsserver
Eftersom det är vanligt att J2EE används i samband med en applikationsserver så följer här nedan en allmängiltig gemomgång av en sådan. Det skall dock observeras att denna rubrik även är giltig för .NET-plattformen.

Ett system med flera nivåer eller en variant av klientserversystem är vad man kan beskriva en applikationsserver som. Servern hanterar, egentligen exekverar, den totala applikationen eller delar av den på olika servrar samt samanknyter dem med varierande komponentobjekts-interface som till exempel CORBA (Common Object Request Broker), .NET eller J2EE (Susning.nu refererad i Lupander Arnoldson & Jönsson, 2003).

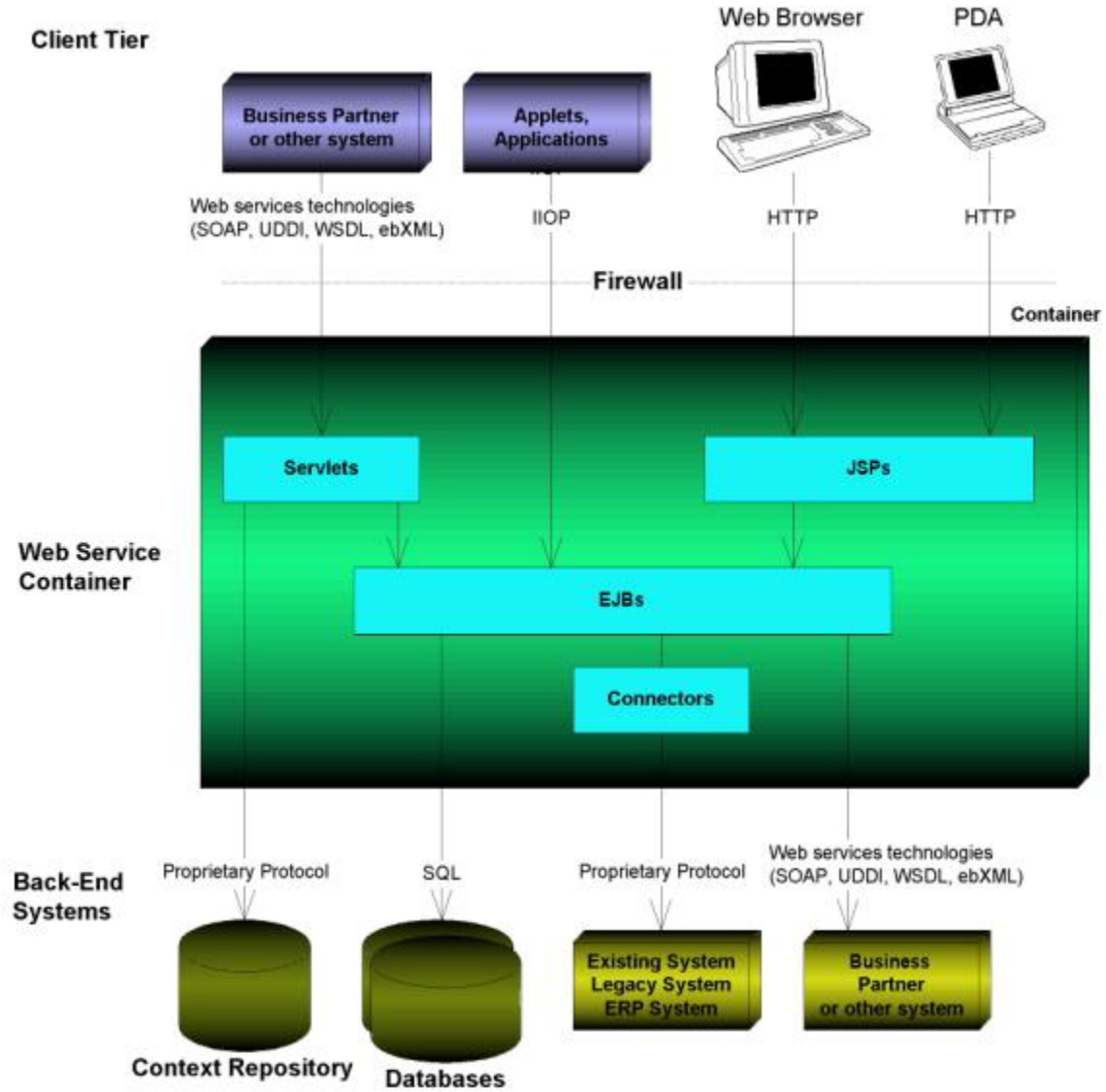
Vid implementation av en applikationsserver så medföljer samtidigt en arkitektur bestående av flera lager. Tanken är att servern placeras i ett lager mellan klienten och databasen samt flyttar logiken från klienten och databasen till komponenter. Applikationsserverns roll är därefter att köra logiken och hantera databaskommunikationen (Norlin refererad i Arnoldson, Lupander & Jönsson, 2003).

Fördelen med ovan nämnda isolering av affärslogik är att en centraliserad samling av logik även innebär att säkerhet och administration blir central. Om servern erbjuder möjlighet för användandet av digitala certifikat så kan man erhålla en säker programplattform utan att den som utvecklar detsamma behöver oroa sig för det. Centraliseringen av administrationen

innebär också att logiken kan uppdateras utan påverkan av gränssnitt. Kontentan när man använder en applikationsserver kan sägas vara att man konstruerar modulära system, erhåller bra skalbarhet på grund av att man får till exempel en effektivisering gällande databaskopplingar och resurser återanvänds samt delas mellan klienter. Ett kraftfullt och dynamiskt system erhålles därmed (Norlin refererad i Arnoldson, Lupander & Jönsson, 2003).

3.6.2.11 Struktur

J2EE är numera utrustat med stöd för att bygga XML-baserade Web Services. Dessa Web Services kan interagera med andra Web Services som har byggts enligt J2EE-standarden eller inte (Vawter & Roman, 2001). Nedan visas J2EE Web Services utvecklingsmodell.



Figur 3:2. J2EE Web Services utvecklingsmodell.

(Källa: Vawter & Roman, 2001)

Kortfattat så kan figur 3:2 ovan förklaras enligt följande:

J2EE applikationen befinner sig inuti en container, vilken förser applikationen med den kvalitet som är nödvändig för tjänster som är avsedda för enterprise program, som exempelvis transaktioner och säkerhet (Vawter & Roman, 2001).

Affärslagret utför affärs processande och datalogik. I storskaliga J2EE-program, så byggs affärslogiken med hjälp av Enterprise JavaBeans (EJB) komponenter. Det här lagret ansluter sig till databaser användandes Java Database Connectivity (JDBC), SQL/J (Structured Query Language/Java) eller existerande system då användandes Java Connector Architecture (JCA). Det kan också ansluta till affärspartners nyttjandes Web Services-teknologier som SOAP, UDDI, WDSL, ebXML (electronic business Extensible Markup Language) genom Java API:erna för XML (JAX API:s) (Vawter & Roman, 2001).

Affärspartners kan ansluta till J2EE applikationer genom Web Services-teknologier som SOAP, UDDI, WDSL, ebXML. En servlet, vilket är ett anrops/svar-orienterat Java-objekt, kan acceptera Web Services anrop från affärspartners. Servleten använder JAX API:s för att utföra de aktuella Web Service tjänsterna (Vawter & Roman, 2001).

Traditionella 'tjocka' klienter som applet program eller program ansluter direkt till EJB lagret genom Internet-ORB Protocol (IIOP) snarare än med Web Services. Sedan så är generellt sett dessutom de 'tjocka' klienterna enligt Vawter och Roman skrivna av samma organisation som utvecklade J2EE applikationen, och därför är det enligt dem ingen idé med ett samarbete med XML-baserade Web Services (Vawter & Roman, 2001).

Webbläsare och trådlösa enheter ansluter till Java Server Pages (JSP:s) vilka använder sig av användarinterface genererade i HTML, XHTML eller WML (Vawter & Roman, 2001).

Ett något mer kortfattat och kanske mer översiktligt sätt att strukturellt beskriva J2EE är:

> De gemensamma byggstenarna för alla lager i J2EE arkitekturen är Java-komponenter. Varje lager stödjer en mängd olika plattforms API:er som till exempel JMS (Java Messaging Services) som stödjer message queuing.
>
> Sun Soft J2EE arkitekturen består av följande lager:
>
> - Klient-lager med Webbläsare som kan köra applets eller Java applikationer.
> - Web-lager med en Webbserver som kör Java Server Pages (JSP) och Java Servlets.
> - Enterprise Java Beans lager med EJB container.
> - Enterprise Information System lager med databas eller stordatorapplikation.
>
> (Britton refererad i Balic, Björklund & Jägmark, 2000)

3.6.3 Programmering under J2EE

J2EE främjar Java-centrerat datoranvändande och som sådant så måste alla komponenter som är utvecklade för användning i en J2EE-miljö (som till exempel EJB komponenter och servlets) vara skrivna i utvecklingsspråket Java. För att använda J2EE så måste man anförtro sig att koda i alla fall några av sina e-Business- (Internetbaserade affärssystem) system till Java genom att använda detsamma. Andra språk kan länkas till en J2EE-lösning genom Web Services, CORBA, JNI (Java Native Interface) eller JCA. Men, de här språken kan inte blandas med Javakod. I teorin, så är JVM bytecode språkneutral, men i praktiken så kan den här byte-koden bara användas med Java (Vawter & Roman, 2001).

Skapande av nämnda byte-kod går till enligt följande:

1. Utvecklare skriver källkod i Java.
2. Javakoden är kompilerat till byte-kod, vilket är en korsbefruktad mellanliggande kod, halvvägs mellan källkod och maskinspråk.
3. När koden är klar att exekveras, så tolkar Java Runtime Environment (JRE) byte-koden och exekverar den när applikationen körs.

J2EE är en applikation som är gjord i Java. J2EE-komponenterna transformeras till byte-kod och exekveras av JRE när de skall köras. Även containrar är vanligtvis skrivna i Java (Vawter & Roman, 2001).

3.6.4 Extra funktionalitet från olika återförsäljare

Alla återförsäljare som erbjuder J2EE plattformen bidrager med extra funktionalitet som inte tillhör standarden. Några av dem påverkar anpassningsförmågan till många plattformar, så som utökad EAI - (enterprise application integration) funktionalitet, E-handels komponenter, eller avancerad B2B- (business-to-business) integration. Andra funktioner, så som load-balancing, transparent fail-over, och caching, påverkar inte detta utan är indirekta tjänster som körs i bakgrunden av containern (Vawter & Roman, 2001).

3.6.5 Java

Java är ett fullt utvecklat programspråk som är mycket mer än bara ett media för material på webbsidor. I likhet med exempelvis C++ så kan man utveckla fristående program som inte behöver en webbläsare för att kunna exekveras (Skansholm 1999)

Det som skänker en extra dimension till Java är att det är plattformsoberoende. Det innebär att program utvecklade i Java kan köras på ett flertal olika plattformar som till exempel: Linux, Windowsserien eller Macintosh. (Lidén & Svensson, 2001)

Java är ett objektorienterat språk som sålunda konstruerar objekt. Dessa objekt beskrivs med hjälp av så kallade klasser. Exempelvis så finns det grafiska klasser som Java använder vid frambringandet av fönster, menyer, knappar med mera (Skansholm, 1999).

3.7 Andra alternativ: Mono, ett open-source projekt för implementation av .NET

Det finns även, förutom de två ovan beskrivna plattformarna, fler alternativ varav ett skall presenteras här. Det kommer i form av ett open-source projektet vid namn Mono, som stöttas av företaget Ximian, vilket i sin tur blev uppköpt av Novell (Loney). Mono är en fri mjukvaruimplementation av aspekter från Microsofts så kallade .NET Framework (Hillesley, 2001).

När uppköpet offentliggjordes, så engagerade sig Novell i att fortsätta stödja Mono-projektet, vilket är konstruerat för att tillåta utvecklade applikationer att nyttja Microsoft.NET för att exekvera dem på Linux, Unix, Windows och andra plattformar, där Novell tror att deras nätverk- och infrastrukturtjänster kan komma till nytta. Enligt Ximian så har mer än 150 utvecklare runt om i världen bidragit till Mono, inkluderandes ett omfång av företag som använder Mono-teknologi för att bygga kommersiella produkter (Loney).

Ximians grundare Miguel de Icaza och Nat Friedman, välkända open-source visionärer, hjälpte till att grunda Mono-projektet liksom de har gjort med Gnome- (GNU Network Object Model Environment) projektet, så när uppköpet var ett faktum så fortsatte Icaza och Friedman sitt arbete med Mono fast då under företaget Novell. Detta på grund av att många open-source projekt är personligt ledda och så även detta (Loney).

De Icaza har även blivit attackerad som upphovsman, tillsammans med hans företag Ximian, gällande Mono. Mono är inte den enda fria mjukvaruimplementationen av .NET, men har blivit föremål för mycket bestörtning, karaktäriserat av passionerade missförståelser från en högljudd minoritet av Linux:s community. De Icaza har blivit anklagad för att vara naiv, och falla för en fälla som skulle vara gillrad av Microsoft, och att han även slösar bort sin tid och kraft som skulle bättre kunna användas till att vidareutveckla GNOME (Hillesley, 2001).

Men sanningen är både mer enkel och komplicerad än så. Att ignorera Microsofts strategiska rörelse mot Web Services skulle vara klart ansvarslöst. Mono, och den alternativa fria mjukvaruklonen, DotGNU, är avsiktliga försök att erbjuda öppna alternativ till .NET som tillåter Webben att förbli den öppna och heterogena miljö som den är tänkt att vara. Dessutom är tanken med dessa öppna alternativ att Webben skall vara oberoende av inskränkta licensierade eller kommersiella intressen (Hillesley, 2001).

3.8 Skillnad mellan .NET- och J2EE – Web Services

Från att precis ha behandlat de mer rent teknologiska bitarna är det nu dags att göra en närmare granskning av en av de springande punkterna för denna uppsats; Valet av plattform som är utrustad med Web Services. Avsnittet kan man säga kortfattat består av en rad jämförelser av olika grad samt argument som alla till sist utmynnar i en diskussion kring vilken plattform som är mest lämplig. Det bör också observeras för tydlighetens skull att de jämförelser som förekommer under det här avsnittet är andras jämförelser som jag presenterar och inte mina egna. Men avsnittet inleds först med en presentation.

3.8.1 Presentation

I det här Avsnittet skall jag försöka framlägga skillnaderna utifrån studerat material, det vill säga befintliga jämförelser, mellan de två plattformarna .NET och J2EE. Det är inte en helt lätt uppgift då åsikterna är många och teknologierna, som ni säkert själva har kunnat konstatera på andra ställen i uppsatsen, inte alltid är helt lätta att varken beskriva eller förstå. Jag har också försökt att till viss mån tona ner den emellanåt mycket högljudda rivalitet som finns mellan Sun och Microsoft som i vissa fall kan ta sig rent löjeväckande proportioner. Jag har inte tagit bort den helt för det vore att censurera men tonat ner den en aning för att undvika de mest ej relevanta ordkrigen.

3.8.2 Introduktion till jämförelse

Denna introduktion innan jämförelserna börjar på allvar har som syfte att ge en första övergripande känsla för plattformarna och är därför av en relativt allmän karaktär.

Inledningsvis så kan man säga att de båda plattformarna .NET och J2EE erbjuder ungefärligen samma uppsättning av funktioner, även om de förekommer på olika sätt. Genom att tillåta komponentinteraktion över många programmeringsspråk , så öppnar .NET upp för programmerare av bland annat språken Perl, Eiffel samt Cobol. (Farley, 2000).

Kampen mellan .NET och J2EE kommer enligt Vawter och Roman att bli en mycket komplex om än intressant historia att följa. Både en stor mängd av löften och sanningar omger dessa omtalade plattformar. Som ett exempel nämner de att J2EE är suveränt om man ser till hur många återförsäljare som är anslutna till plattformen, men skall å andra sidan inte enligt Vawter och Roman betraktas som något osjälviskt alternativ bara för att det råkar vara av open source-karaktär. Alla deltagande återförsäljare ingår enligt dem med ekonomisk profit i åtanke, så väl som att även använda det som ett vapen i kampen mot Microsoft gällande vem som skall bli herre på täppan över mjukvaruplattformar (Vawter & Roman, 2001).

Vidare enligt Vawter och Roman så låter J2EE dessa återförsäljare samarbeta och därmed stå på en gemensam fast grund. Många av återförsäljarna har själva blivit uppköpta eller genomgått sammanslagningar, så organisationer som är intresserade av plattformen måste visa på gott omdöme när de väljer en av dem (Vawter & Roman, 2001).

Beträffande .NET så är även det enligt Vawter och Roman långt ifrån något osjälviskt initiativ. Det är ett monopolistiskt initiativ förklätt med osjälviskhet. Microsoft har hävdat att .NET handlar om Web Services som är öppna och kan förenas genom samarbete. I verkligheten gör snarare Microsoft sina Web Services slutna och privatägda. Microsoft vill troligtvis öka sin lösnings kostnad om ett monopol kan åstadkommas, och därmed kommer innovationsförmågan att signifikant bromsas upp (Vawter & Roman, 2001). Detta tankesätt kring .NET får medhåll av Richard Hillesley (Hillesley, 2001) som också är kritisk mot Microsofts öppenhet och hävdar att ett system som låtsas vara öppet men ändå driver alla användare mot Microsofts servrar på grund av att autenticiteten och auktorisationen inte kan garanteras vara öppen, måste enligt Hillesley betraktas som ytterst felaktigt och kan leda till att en kil skapas mellan användare av open source – produkter och användare av Microsofts motsvarigheter.

3.8.3 Hur står sig .NET och J2EE i en jämförelse?

Under denna rubrik så börjar själva jämförelsen, som alltså är återgivningar av befintliga jämförelser, som man kan säga är uppdelat i två delar, en mer övergripande som är baserat på två tabeller för att på så sätt generera en god översikt. Den första av dessa tabeller förevisar likheter mellan plattformarna medan den senare ger en mer detaljerad översikt. Denna första del följs av en andra som är mer utförlig och ger en djupare insikt än vad den första delen gör. Efter dessa två delar följer även en del med kompletterande material. Hela jämförelsemomentet avslutas därefter med ett beslutsavsnitt där det diskuteras vad som kan vara lämpligt att tänka på vid valet mellan dessa två plattformar.

3.8.3.1 Jämförelse, del ett

Som en första hjälp för att förstå de båda modellerna så uppvisar tabell 3:1 analogierna mellan J2EE- och .NET-plattformarna enligt Vawter och Roman (Vawter & Roman, 2001):

Tabell 3:1. Analogier mellan J2EE och .NET (Källa: Vawter & Roman, 2001)

Funktion	**J2EE**	**.NET**
Teknologi-typ	Standard	Produkt
Middleware återförsäljare	30+	Microsoft
Interpreterare	JRE	CLR
Dynamic Web Pages	JSP	ASP.NET
Mellanskikts-komponenter	EJB	.NET- hanterade Komponenter
Databas-acess	JDBC, SQL/J	ADO.NET
SOAP, WDSL, UDDI	Ja	Ja
Implicit mellanskikt (s.k 'load –balancing' etcera)	Ja	Ja

För att följa upp den här lätta introduktionen av del ett så följer som sagt en mer ingående översikt. I denna översikt skall de ovan nämnda teknologierna och än fler beskrivas något mer ingående om än fortfarande översiktligt och i tabellform.

Som man kan se i tabellen nedan så har till exempel .NET plattformen en rejält vedertagen uppsättning teknologier till sitt förfogande. Men enligt Farley (Farley, 2000) så presenterar Microsoft sina teknologier med förevändningen att enbart vara ett alternativ till andra existerande plattformar, så som J2EE och CORBA, men har i själva verket enligt honom som syfte att locka kunder från dessa så att de istället övergår till Microsofts motsvarighet. Men hur är det med det egentligen? Ett sätt att komma till klarhet med det är att granska Farleys nedanstående punkt-för-punkt jämförelse i tabell 3:2 (Farley, 2000):

Tabell 3:2. Punkt-för-punkt jämförelse av .NET och J2EE (Källa: Farley, 2000)

.NET	**J2EE**	**Beskrivning av teknologi**
C# programmerings-språk	Java Programmerings-språk	Både Java och C# härstammar från C och C++. Mest signifikanta funktioner (tex automatisk minneshantering och hierarkiska namnrymder) är presenterade i båda. C# lånar några av de konceptuella komponenterna från JavaBeans (händelser, attribut etc) adderar några egna (som metadata-taggar), men implementerar dessa funktioner på ett annorlunda sätt i syntaxen. Java går att använda på vilken plattform som helst som har en Java VM (JVM, Java Virtual Machine). C# gick år 2000 bara att använda under Windows. C# är specifikt bunden till den så kallade IL Common language runtime (se nedan) och exekveras som 'just in

		time' (JIT) kompilerad byte-kod eller kompileras helt och hållet till plattformens språk. Java kod exekveras som Java Virtual Machine (VT) byte-kod som antingen är tolkat av VM, är JIT kompilerad eller i likhet med C# helt kompilerat till det inhemska språket.
.NET Common 'Components (även kallat '.NET Framework SDK')	Java Core API	Högnivå .NET komponenter kommer att inkludera stöd för distribuerad access och SOAP (se nedan ADO+)
Active Server Pages+ (ASP+)	Java Server Pages (JSP)	ASP+ kommer att använda språken Visual Basic, C# och troligen andra språk för kodsnuttar. Allt blir kompilerat till det inhemska språket genom 'the common language runtime' (till skillnad från att bli interpreterat varje gång som ASP:er). JSP använder Javakod (snuttar, eller JavaBean referenser), kompilerat till Java byte-kod (antingen 'on-demand' eller 'batch-compiled' beroende på JSP implementationen
IL Common Language Runtime	Java Virtual Machine och CORBA IDL och ORB	**.**NET:s 'Common Language Runtime' tillåter kod härrörandes från multipla programmeringsspråk att använda delade uppsättningar av komponenter på Windows plattformen. Ligger till grund för nästan allt av .NET Framework (vanliga komponenter, ASP+, etc) Javas Virtual Machine specifikation gör det möjligt för Javas byte-kod att bli exekverad på vilken plattfrom som helst som har kompatibel JVM. CORBA tillåter att kod från flera språk använder delade uppsättningar av objekt, på vilken plattform som helst där en så kallad ORB närvarande. Inte alls lika nära integrerat med J2EE Framework.
Win Forms och Web Forms	Java Swing	Liknande webbkomponenter (tex baserade på JSP) är tillgängliga Javas standardplattform, några licensierade komponenter finns dock att tillgå genom Java IDE:s etc Win Forms och Web Forms RAD utveckling stöds genom Visual Studio IDE – för närvarande så stöds ingen annan IDE. Stöd för Swing finns i många Java- IDE:s och verktyg.
ADO+ och SOAP-baserade Web Services	JDBC, EJB, JMS och Java XML bibliotek (XML4J, JAXP)	ADO+ grundar sig på förutsättningen av utbyte av XML-data (mellan fjärr-dataobjekt och lager av multi-skikts applikationer) ovanpå HTTP (även känt som SOAP). .NET:s Web Services förutsätter generellt SOAP:s meddelandemodell. EJB, JDBC etc lämnar

datautbytes protokollet åt utvecklarens godtycke, och är istället funktionella ovanpå antingen http, RMI/JRMP eller IIOP.

3.8.3.2 Jämförelse, del två
Denna andra del av jämförelsen kommer som nämnts bli mer detaljerad för att på så sätt kunna urskilja mer specifika differenser mellan plattformarna

Enligt Hanson (Hanson, 2002) så består Web Services av fyra grundläggande utmaningar runt vilka Hanson har skapat en jämförelse som jag i detta avsnitt skall återge.

De utmaningar som utgör hans antagande är följande:

1. Beskrivning av tjänst
2. Implementering av tjänst
3. Publicering, upptäckt och bindande av tjänst
4. Anrop samt exekvering av tjänst

Avsnittet kommer att i tur och ordning behandla dessa områden samt granska hur de två plattformarna tar sig an var och ett av dessa områden.

Beskrivning av tjänst
För att Web Services skall kunna få stor spridning så är det viktigt att man kan beskriva det på ett strukturerat sätt. Det så kallade ’Web Services Descripting Language’(WDSL) tar i tu med det här behovet genom att definiera vad man kan kalla grammatik för Web Services. Denna grammatik beskrivs som en samling av ändpunkter eller portar med funktionaliteten att bära med sig meddelanden (Hanson, 2002).

I WDSL så skiljer man i mellan var ändpunkter och meddelanden faktiskt verkar/finns och beskrivningen av desamma. Ett faktiskt, och ej endast beskrivet, protokoll tillsammans med beskrivningen av dataformatet för en särskild sort av ändpunkt, där en ändpunkt vanligtvis är en port, utgör en så kallad bindning. Vidare så är ändpunkt definierad genom att associera en webbadress med en bindning. Det vill säga man gör det därmed möjligt för access till ändpunkten över Internet. Slutligen så kallar man en samling av dessa ändpunkter för en tjänst (Hanson, 2002).

J2EE
J2EE-stödda Web Services utbyter information med intressenter genom att använda WDSL för att på så sätt kunna fastställa det korrekta formatet för varje överfört XML-dokument. Om en tredje part vill uppsöka information om ett företags Web Services, i syfte att utbyta information, så finns dessa i ett register (Hanson, 2002).

.NET
I likhet med J2EE:s Web Service, så stödjer en .NET Web Service WDSL specifikationen och använder ett WDSL dokument för att beskriva sig själv. Men .NET skiljer sig i det här fallet gällande definitionen av en ändpunkt då den sker genom användandet av en så kallat XML-namrymd som finns inom ett WDSL dokument för att på så sätt kunna genomföra en unik definition av en Web Service:s ändpunkter (Hanson, 2002).

.NET är försett med en komponent på klientsidan som låter en applikation anropa Web Services operationer (de saker som aktuell Web Service kan utföra), som är beskrivna av ett WDSL dokument, samt en komponent på serversidan som styr tjänstens förehavanden till ett COM-objekts metodanrop, server sidans komponent är i sin tur beskriven i en ’WDSL- och Web Services Meta Language’- (WSML) fil. Denna fil behövs för Microsofts implementation av SOAP (Hanson, 2002).

Implementering av tjänst

Implementering av Web Services innebär för närvarande att strukturera data och verksamhet inifrån ett XML-dokument som är kompatibelt med specifikationen av SOAP. När en Web Service väl en gång är implementerad, så sänder en klient ett meddelande till en komponent i form av ett XML-dokument följt av att komponenten i sin tur skickar ett XML-dokument tillbaka som svar (Hanson, 2002).

J2EE

Existerande Javaklasser och applikationer kan bli så kallat ’wrapped’ det vill säga dölja sig själva genom användandet av Java API:n för XML-baserad RPC (JAX-RPC) och därmed framställa sig själva som Web Services. JAX-RPC använder XML för att göra RPC anrop och utsätter ett API för ’marshalling’ (paketera parametrar och returnera värden som skall distribueras) och även denna process inverterat så kallad ’un-marshalling’ men då applicerat på argument samt för att sända och ta emot proceduranrop (Engelen, 2004; Hanson, 2002).

Med J2EE, så kan affärsrelaterade tjänster som är skrivna som Enterprise JavaBeans (EJB) vara förklädda och förevisade som Web Services. Resultatet av en sådan förklädelse är en Web Service med stöd för SOAP som är kompatibel med ett WDSL-gränssnitt och baseras på de ursprungliga metoderna hos EJB (Hanson, 2002).

Arkitekturen hos J2EE:s Web Services är en sammanställning av XML-baserade så kallade frameworks, ramverk. Dessa förser företag med infrastrukturer som gör det möjligt för dem att integrera logik hos affärstjänster som tidigare endast förekom som licensierade gränssnitt. För närvarande stödjer J2EE Web Services via Java API:n för XML parsing (JAXP). Denna API tillåter användare att utföra vilken verksamhet som helst med Web Services genom att manuellt använda sig av parsing på XML-dokument (Hanson, 2002).

.NET

.NET-applikationer exekveras inte längre i inhemsk maskinkod. Alla program kompileras till en mellanliggande binärkod som kallas ’Microsoft Intermediate Language (MSIL). Den här portabla och binära koden är därefter kompilerad ytterligare en gång till inhemsk kod med hjälp av en ’Just In Time ’ – kompilerare (JIT) vid exekvering medan själva exekveringen utförs av en så kallad virtuell maskin som här benämns som ’Common Language Runtime’ (CLR). Med .NET-plattformen, vill Microsoft erbjuda många språk som är baserade på ’Common Language Infrastructure’ (CLI), såsom ’Managed C++’, JScript, VB.NET och C#. ’Microsoft SOAP Toolkit’ erbjuder komponenter som konstruerar, sänder, läser och processar SOAP-meddelanden (Hanson, 2002).

Publicering, upptäckt och bindande av tjänst
När en Web Service väl har blivit implementerad, så måste den lämpligtvis även bli publicerad någonstans där intresserade parter kan hitta den. Informationen om hur en klient kan ansluta sig själv till en Web Service och interagera med den, måste även det bli exponerat på en plats som är tillgänglig för den aktuelle klienten. Den anslutnings- och interaktionsinformationen är ofta hänvisad till som så kallad bindningsinformation (Hanson, 2002).

För närvarande är register det primära medlet för att publicera, upptäcka och binda Web Services. Register innehåller datastrukturer och taxonomier, det vill säga hierarkiska ordningsstrukturer, som används för att beskriva Web Services och återförsäljare av de samma. Ett register kan antingen upprätthållas av en privat organisation eller av en neutral tredje part (Hanson, 2002).

IBM och Microsoft har framtagit en specifikation vid namn 'Web Services Inspection Language' (WSIL) som tillåter program i webbservrar att leta efter XML Web Services. WSIL utlovas att komplettera UDDI genom att göra det lättare att upptäcka tillgängliga tjänster när de finns på webbsajter och inte som nu när de finns listade i UDDI-register (Hanson, 2002).

J2EE
Sun Microsystems framhåller sin Java API för XML-register (JAXR) som ett enda allmänt API som skall kunna arbeta med många olika registertyper. JAXR gör det möjligt för sina klienter att komma åt Web Services genom en Web Service implementerare som exponerar Web Services som är byggda på en implementation av JAXR specifikationen (Hanson, 2002).

Det finns tre olika implementationer av JAXR:

- JAXR pluggable providern, vilken implementerar funktioner från JAXR specifikationen som är självständiga från någon registertyp.
- Register-specific JAXR provider, vilken implementerar JAXR specifikationen på ett sätt som är typiskt för register.
- JAXR Bridge Provider, vilken inte är specifik för ngt särskilt register. Den tjänstgör som en brygga till en klass av register så som ebXML eller UDDI.

.NET
Först, så tog Microsoft fram det så kallade DISCO-protokollet för att upptäcka Web Services. En publicerad DISCO-fil är ett XML-dokument som innehåller länkar till andra källor som kan beskriva Web Service:n ifråga. Sedan UDDI har blivit mer spritt så har även Microsoft trots allt beslutat att stödja UDDI för att på så sätt öka möjligheten att kunna samarbeta mellan olika lösningar.

Förutom att erbjuda en .NET UDDI-server, så förser UDDI SDK stöd för Visual Studio .NET och förlitar sig till .NET framework. Produkter som Microsoft Office XP är numera försedda med möjligheten att upptäcka tjänster med hjälp av UDDI.

Anrop samt exekvering av tjänst
'Simple Access Protocol' (SOAP) är ett enkelt, lättillgängligt XML-baserat protokoll som definierar ett ramverk för meddelanden som används för utbyte av strukturerad data och datatyp- information över webben.

SOAP-specifikationen består av fyra huvudsakliga delar:

- Ett obligatorisk så kallat kuvert för inkapsling av data
- Valfria regler för kodning av data för att representera applikationsdefinierade datatyper, och en modell för serialisering av modeller som inte följer syntax.
- Ett mönster angående utbyten av anrop/svars-meddelanden
- Ytterligare en bindning mellan SOAP och HTTP

SOAP kan användas i kombination med vilket annat transport- protokoll eller mekanism som kan transportera det så kallade SOAP-kuvertet.

Mottagare av Web Services agerar som så kallade 'SOAP-lyssnare' (listener) och kan underrätta intresserade parter, så som andra Web Services, applikationer etcetera, när en Web Service förfrågan är mottagen. SOAP-lyssnaren utvärderar ett SOAP-meddelande mot motsvarande XML-scheman som finns definierade i en WDSL-fil. Därefter 'un-marshals' (tages emot och packas upp) SOAP-meddelandet. Inom SOAP-lyssnaren så kan de som skickat meddelandet anropa motsvarande implementation av Web Service-kod. Till sist så anropas affärslogiken för att erhålla svaret. Resultatet från affärslogiken omvandlas till ett SOAP-svar och återsänds till den som anropade med Web Service:n (Hanson, 2002).

J2EE
J2EE använder Java API:n för XML-baserad RPC (JAX-RPC) för att sända SOAP – metodanrop till avlägsna parter och sedan även mottaga svaret. JAX-RPC möjliggör det för Javautvecklare att bygga Web Services som innefattar XML-baserad RPC-funktionalitet enligt SOAP-specifikationen (Hanson, 2002).

När en JAX-RPC är definierad och implementerad, så är tjänsten verkställd på en server med en så kallad 'JAX-RPC runtime system'. Momentet för verkställande är beroende av vilken typ av komponent som har blivit använd för att implementera JAX-RPC tjänsten. Till exempel, en EJB-baserad tjänst är verkställd i en EJB-container (Hanson, 2002).

Under verkställandet av en JAX-RPC tjänst, så konfigurerar verktyget för verkställandet en eller fler bindningar av protokoll för den aktuella JAX-RPC tjänsten. En bindning förenar en abstrakt definition av en tjänst med ett specifikt XML-baserat protokoll samt transport. Ett exempel av bindning är protokollet SOAP över, eller kanske rättare sagt användandes av, HTTP (Hanson, 2002).

.NET
I Microsofts .NET ramverk, så kan intresserade parter få access till en Web Service genom att implementera en Web Servicelyssnare. För att kunna utföra denna implementering, så måste systemet i fråga kunna förstå SOAP-meddelanden, generera SOAP-svar, förse Web Service:n med ett så kallat WDSL-kontrakt samt utlysa dess existens via UDDI (Hanson, 2002).

.NET-programmerare som utvecklar SOAP-baserade Web Service-lyssnare och konsumeter har för närvarande tre val gällande detta (Hanson, 2002):

1. Använda den inbyggda .NET SOAP meddelandeklassen
2. Skapa en Web Service-lyssnare för hand, med hjälp av MSXML (Microsoft Extended Markup Language), ASP, eller ISAPI (Internet Server API).
3. Nyttja Microsofts SOAP-toolkit version 2 för att konstruera en Web Service-lyssnare som ansluter till en affärsfasad, implementerat genom att använda COM.

Microsoft SOAP Toolkit 2.0 erbjuder en komponent för klientsidan som tillåter ett program att anropa funktionaliteten hos Web Services som i sin tur är beskrivna av ett WDSL-dokument (Hanson, 2002).

3.8.3.3 Jämförelse, en komplettering till del två
Detta avsnitt tjänar som en komplettering till den ovanstående omfattande delen gällande jämförelse av plattformarna. Här tas fram aspekter som bara har snuddats vid i ovanstående genomgång och bidrar därmed till ytterligare synvinklar.

Till delvis skillnad mot Hanson ovan så tycker Lurie och Belanger (Lurie & Belanger, 2002) i sin jämförelse att följande två områden är de mest signifikanta gällande att bedöma skillnader mellan plattformarna; Hur de hanterar multiplattforms, det vill säga förmågan att kunna agera på flera plattformar samt förmågan att stödja flera olika programmeringsspråk. Dessa skillnader enligt Lurie och Belanger skall jag nu nedan återge.

Multiplattform
Från en utvecklares perspektiv så förser både J2EE och .NET honom eller henne med de verktyg som är nödvändiga för att skapa Web Services. Medan J2EE briljerar med stöd av flera plattformar, så om man får tro Microsoft kommer denna glädje inte att vara så länge. Microsoft har nämligen profilerat sin .NET-plattform för användande av en två-stegs kompileringsprocess, vilken tillåter Microsoft att förse så kallad ’runtime’-miljö för olika plattformar, och därmed är likheten enligt Lurie och Belanger med hur Java fungerar slående (Lurie & Belanger, 2002).

Det har även från Microsofts håll förekommit rykten om en FreeBSD-port för .NET. FreeBSD är ett kompatibelt operativsystem som härstammar från BSD (Berkely Software Distribution) Unix. Fastän ett dylikt stöd skulle kunna utmana Javas exklusiva plattformsoberoende så skall man inte vara helt säker på det än. Det kan ta år innan CLR:er för de vanligaste operativsystemen är färdiga. Vidare, så gjorde Microsoft en gång liknande hävdelser att konstruera portar för andra plattformar gällande DCOM (Distributed Component Object Model). Dessa portar blev aldrig realiserade (Lurie & Belanger, 2002).

Stöd av många utvecklingsspråk
J2EE:s enskilda språkfundament, Java, kontrasteras starkt av .NET:s motsvarighet, vilken stödjer mer än två dussin språk, inkluderandes Fortran, COBOL, C++ och Visual Basic. Fast man skulle kunna hävda att en lösning med ett enda språk framhäver en mer smidig lösning, så måste man medge att .NET erbjuder en fördel för organisationer som vill utnyttja sina utvecklares fulla potential. För de utvecklare som för närvarande använder sig av andra språk än Java, så låter .NET dem utveckla Web Services i sina respektive mer så att säga hemtama språk. Detta med faktumet att de behöver minimalt med träning och kan jämföras med situationen om de skulle behövt lära sig ett nytt språk (Lurie & Belanger, 2002).

Microsofts strategi kan förskjuta Java som endast ytterligare ett programmeringsspråk. Sun:s försvar mot det är att de säger att Java är inte bara ett språk utan även en plattform (Lurie & Belanger, 2002).

3.8.3.4 Jämförelse, övrigt
Andra märkbara funktioner som inte finns hos Suns plattform, inkluderar enligt Lurie och Belanger ASP.NET:s förmåga att rendera webbsidor i olika HTML-format, vilka abstraherar utvecklare från versionsspecifik HTML. Servlets kan åstadkomma samma sak, men kräver då mer manuell kodning för att det skall kunna realiseras (Lurie & Belanger, 2002).

3.8.3.5 Jämförelse, argument inför val
Nedan anges i punktform ett antal argument för båda plattformarna som enligt Vawter och Roman påvisar för- och nackdelar på ett jämförande sätt. Detta kan tjäna som en bra tankeväckare inför det avslutande avsnittet när det är dags att diskutera det slutgiltiga valet av plattform (Vawter & Roman, 2001):

Argument som stödjer båda plattformarna

- Oavsett vilken plattform som du väljer, så behöver nya utvecklare utbildning på den kommande plattformen (Java utbildning för J2EE, och motsvarande i objektorientering för .NET).
- Du kan idag bygga Web Services användandes båda plattformarna.
- Båda plattformarna erbjuder en låg systemkostnad, så som lösningar av typen JBoss/Linux/Cobalt för J2EE, eller Windows/Win32 hårdvara för .NET.
- Båda plattformarna erbjuder en ’enskild återförsäljare’-lösning.
- Skalbarheten för båda lösningarna är teoretiskt sett obegränsad.

Argument för .NET och mot J2EE

- Microsofts främsta marknadsföringsteam jobbar med .NET
- .NET släppte sin version av Web Service:s innan J2EE, och har således ett intellektuellt försprång
- .NET har idag en bättre version för så kallad ’shared context’ än J2EE
- Med Visual Studio .NET så har .NET en enastående verktygsversion
- .NET har en enklare programmeringsmodell, som gör det möjligt för mindre avancerade utvecklare att vara produktiva utan att trassla till det för sig själva
- .NET erbjuder språkneutralitet vid utvecklandet av nya e-handelsprogram, medan J2EE behandlar andra språk som separata program
- .NET har till stor fördel att det drar nytta av det operativsystem som finns på det aktuella systemet

Argument för J2EE mot .NET

- J2EE marknadsförs av en hel industri
- J2EE är en utprövad plattform, med några nya API:er för Web Services. .NET är en omskrivelse och innebär därför en stor risk som vilken förstagenerations teknologi som helst

- J2EE har en mer avancerad programmeringsmodell, lämplig för vältränade utvecklare som vill bygga mer avancerade objektmodeller och dra fördel av prestandafunktioner.
- J2EE ger dig plattformsneutralitet, inkluderandes Windows. Du får också bra (men inte fri) plattformsfrihet gällande mjukvaran. Det här gör att man slipper heterogena verkställande-miljöer
- Med J2EE kan man använda det operativsystem som man föredrar, så som Windows, UNIX eller stordator. Utvecklare kan använda den miljö som de känner sig mest bekväma i.

3.8.3.6 Val av plattform

Efter denna granskning av hur J2EE och .NET hanterar Web Services, så återstår det svåra beslutet, Vilken av dessa man skall välja. Från en rent tekniskt synvinkel så har varje plattform sina för- respektive nackdelar (Hanson, 2002). Dessa är även användbara (Vawter & Roman, 2001) och kan leda till samma mål.

Den huvudsakliga fördelen, kanske, när det gäller användandet av .NET i samband med Web Services är att det enligt Hanson har från början konstruerats för just det ändamålet, medan J2EE har blivit justerat till denna funktionalitet genom tilläggandet av fler API:er. En fördel som J2EE har är dess stora antal återförsäljare och kan således lättare anpassa J2EE till ditt existerande system (främst kanske med applikationsservrar i åtanke), inkluderandes ett otaligt antal open-source projekt. På många sätt så står open source J2EE-applikationsservrar närmare den standard som har grundlagts av Sun, genom det faktum att de adderar inte licensierade tillägg för att lösa problem (Hanson, 2002). Just denna differens mellan J2EE och .NET gällande antalet plattformar som de kan användas till där J2EE kan brukas på flera stora plattformar och .NET bara kan användas i en Windows-baserad servermiljö. Med detta i tanke så säger Gustafson (Gustafson) att har man redan en Microsoft miljö så finns det ingen anledning att byta samt att denna miljö är både billigare och har bättre prestanda i Windows än vad Java har. Däremot om du använder en annan miljö så kan Java vara det bättre valet av de båda. Eller för i enlighet med .NET:s fördelar citera en nöjd anförskaffare av systemet ifråga:

> *”Man har smidigheten från de gamla komponenterna men på ett mycket enklare och renare sätt. När vi väl insåg det så var det ganska lätt. .NET är ju inte bara de här webbgrejerna utan det är en helt ny bas för alla deras miljöer och man måste gå över till .NET antingen man vill eller inte i allfall om man jobbar i en Windows mijö”*. (Dahn Nilsson citerad från Johansson & Ledin, s. 54, 2001)

En fördel som bör nämnas gällande J2EE med tanke på deras stora antal återanvändare är att det har gett upphov till en stor mängd verktyg, produkter och applikationer på marknaden. Dessa innehar totalt sett mer funktionalitet än vad någon enskild återförsäljare skulle kunna anskaffa. Men denna styrka kan också verka åt motsatt håll, det vill säga som en nackdel. J2EE verktyg kan ofta inte dela mjukvara mellan olika plattformar som en följd av felaktigheter berörande möjligheten att inte vara bunden till en viss hårdvara. Det begränsar förmågan att blanda verktyg utan att behöva ta till omfattande lågnivå-programmering. Det här blir särskilt märkbart om man använder sig av en enklare implementation av J2EE. Där så måste man blanda för att få en fullständig lösning, och det är priset man får betala när man väljer ett mindre komplett paket. Större återförsäljare erbjuder kompletta lösningar för Web Services. .NET erbjuder en ganska komplett lösning från en återförsäljare: Microsoft. Den lösningen kan sakna några av de mest avancerade funktionerna som finns att tillgå under

J2EE. Generellt sett så skall dock ändå den kompletta Web Services-visionen som Microsoft erbjuder motsvara i omfattning jämfört med det som en större återförsäljare av J2EE erbjuder (Vawter & Roman, 2001).

Ytterligare en aspekt som man måste överväga vid valet av plattform och som inte är så teknikrelaterat är själva organisationens agerande. En organisation som existerar i en så snabbt omväxlande miljö som Internetvärlden eller bara interagerar med denna genom nyttjandet av datasystem, måste också anpassa sig till detta och aptera sig i enlighet med sin kontext. Organisationen måste kunna fortsätta vara operativ på ett tillfredställande sätt även under exempelvis ett systembyte eller dylikt och i ett sådant läge inte behöva stänga ner affärsverksamheten. Detta vore förödande med tanke på intäkterna från verksamheten. Därför är det viktigt för en modern organisation att deras kritiska system fungerar på detta sätt. Just för Internetvärlden som är väldigt dynamisk och föränderlig så är det mycket viktigt att bland annat kunna alternera mellan olika plattformar, språk, system etcetera. Det gör att kraven på organisationens systemarkitektur blir höga. Detta är aspekter som är mint lika viktiga att tänka generellt på och kanske än viktigare vid val av mjukvaruplattform. (Arnoldsson, Lupander & Jönsson, 2003). Även att sådant som inköpscykler skulle spela en större roll än det teknologiska vid den här typen av val är något som Loney (Loney) hävdar.

Detta sätt att när det gäller valet av plattform se till ett mycket vidare perspektiv än enbart de tekniska aspekterna präglar även Vawter och Romans åsikter (Vawter & Roman, 2001). De säger att innan man beslutar sig, så rekommenderas det att man koncentrerar sig på större affärsaspekter. Tänka på vilken kompetens som finns gällande utveckling, existerande system, vilka relationer man har just nu till återförsäljare samt tänka på kunderna. Dessa skäl är nästan alltid de som är förankringen vid beslut, inte några mindre funktioner hit eller dit (Vawter & Roman, 2001). Även David Chapell**,** föreläsare och författare på ämnet mjukvaruutveckling (Chapell refererad från Gustafson), menar att man inte skall lägga för stor vikt vid den tekniska biten när man väljer. Han fortsätter med att säga att själva idén med dessa båda plattformar är att skapa öppna lösningar som har etablerade standarder i botten.

Sammanfattningsvis så kan man säga enligt J,Jeffrey Hanson chefsarkitekt på Zaraus (Lurie & Belanger, 2002), att såvida man inte startar ett system helt från början, så kommer valet av vilken implementation man skall välja för sina blivande Web Services till stor del bero på det befintliga systemet. Om man har ett team av duktiga programmerare, med ett befintligt affärssystem, så kommer man troligtvis fortsätta använda det systemet varken man väljer att använda J2EE eller .NET (Hanson, 2002). På tal om programmerare så säger Lurie och Belanger (Lurie & Belanger, 2002) en intressant sak genom att hävda att även om ett av dessa system skulle visa sig överlägset så det inte just det förrän det används av programmerare och dylika människor som innehar denna höga kompetens. Verktygen är endast så bra som de utvecklare som handhar dem (Lurie & Belanger, 2002). Allra sist så bör man nog också tänka på att trots all utveckling och framsteg med mjukvaruplattformar så är de ännu fortfarande mycket i sin linda enligt Colin Adam (Hoffman), redaktör på tidningen Webservices.org, en online-community som är tillägnad Web Services.. Han fortsätter med att säga att mycket arbete kvarstår inte minst gällande standarder. Web Service-teknologier har dessutom, som han fortsätter att säga, sådana saker framför sig att ta itu med som bland annat säkerhet, kvaliteten på tjänsterna samt transaktioner.

3.9 Prestandatester

Under detta avsnitt kommer jag att presentera två stycken prestandatester där det första testar mer ordinära Webb tjänster och hur ASP.NET fungerar med dem medan det andra inkluderar bland annat test av Web Services. Dessa två test skall sedan jämföras med vad jag själv har kommit fram till under empirin i den här uppsatsen. Testerna i sig som presenteras här har jag tänkt skall ge en god bild av hur Web Services men även de två plattformarna i övrigt som behandlas i uppsatsen, det vill säga .NET och J2EE, fungerar under samt klarar av extrem belastning.

Avsnittet inleds dock inte med en prestandatest utan en introduktion till ämnet som följs av specifikation för prestandatest. Introduktionens syfte är att ge målgruppen som kanske inte är bekant med vad prestandatest innebär, en första inblick av detsamma. Specifikationen som sedan följer denna introduktion ligger till grund för Middleware Companys prestandatest som jag lite längre fram i avsnittet granskar. Men nu allra först en introduktion.

3.9.1 Prestandatester – introduktion

3.9.1.1 Källa

Källa till avsnitt 3.9.1 är om inget annat anges: (Butler & Farell).

3.9.1.2 Definition

Man kan enligt Butler och Farell tänka sig prestandatester som exekvering av en uppsättning av test på ett system för att jämföra dess prestanda och kapacitet med andra systems. Prestandatest mäter enligt dem:

- Prestanda hos ett helt system
- Prestanda hos ett specifikt subsystem, så som diskar, video, CPU (datorns processor) eller datorns minne
- Prestanda hos en särskild applikation, så som en databas eller CAD/CAM-program
- Prestanda hos server-baserade applikationer, så som databaser, fil, meddelande eller Webb -applikationer
- Kapacitet hos ett helt affärssystem

Alla bra prestandatester använder sig enligt dem av en väldefinierad testmetodik baserad på hur systemet som skall testas används i verkliga livet. Prestandatester mäter systemets prestanda på ett fastställande och reproducerbart sätt, tillåtandes en användare att ordentligt bedöma prestandan och kapaciteten hos sitt system. När prestandatest enligt Butler och Farell används ordentligt så kan de redivisa följande kriterier hos ett system:

- Pålitlighet
- Så kallade ’flaskhalsar’ (faktor i ett system som bromsar upp exempelvis prestandan)
- ’Enterprise’ kapacitet (duglighet i en affärsmiljö)
- Opartisk köp-information

3.9.1.3 Nyttoaspekten hos prestandatest

Varför använder man sig då av prestandatest kan man ju fråga sig? Enligt Butler och Farell så ligger nyttoaspekten i följande tankegångar; Pålitliga servrar i affärsbruk är lika med bra kundservice. Bra kundservice är lika med maximerad förtjänst. Deras tankegångar fortsätter med att de säger att i en högst konkurrenskraftig och icke-förlåtande e-handelsmiljö, så är

systemets prestanda den faktor som håller kunderna nöjda. Vidare så bidrager nöjda kunder till högre inkomster samt förtjänster hos företaget i fråga.

Som motpol till Butler och Farells uttalande ovan så kan man i stycket ’ 3.9.4.4 Slutsatser som kan dras från denna, likväl som från prestandastudier i allmänhet’ (Middleware Company Case Study, 2003) läsa om att det även finns åsikter om att andra kriterier, som hur lätt det är att använda, vilket stöd som finns för olika verktyg eller vad den totala ägandekostnaden är, än prestandatest kan vara nog så viktiga vid val av server-applikatonsplattfomar.

3.9.1.4 Olika sorter av prestandatest
På SPEC (The Standard Performance Evaluation Corporation), ett oberoende företag som skapats för att etablera, underhålla samt värna om standardisede prestandatest (SPEC Webbsida), webbsida så presenteras en rad prestandatest som kan användas inom många olika områden. Nedan följer tre stycken exempel på dessa områden samt medföljande citat som beskriver ett prestandatest inom varje av dessa områden:

CPU

> ***“CPU2000***
> *Designed to provide performance measurements that can be used to compare compute-intensive workloads on different computer systems, SPEC CPU2000 contains two benchmark suites: CINT2000 for measuring and comparing compute-intensive integer performance, and CFP2000 for measuring and comparing compute-intensive floating point performance.”*

Enterprise Services

> ***“appPlatform***
> *Currently under development, SPECappPlatform is designed to measure the scalability and performance of enterprise platforms (such as J2EE and .NET) running web services. The benchmark will include features that are typical in customers' enterprise applications, such as transactions, persistence services, web services, and messaging.”*

Web Servers

> ***“WEB99***
> *A client/server benchmark for measuring the maximum number of simultaneous connections that a web server is able to support. The benchmark load is presented by client software on client machines networked to server machines running an HTTP server.”*

3.9.1.5 Krav som kan ställas på ett bra prestandatest
Väl definierade design mål är viktigt för ett bra prestandatest. Ett bra prestandatest måste enligt Butler och Farell inneha följande karaktäristika:

- **Skalbarhet** - Prestandatest, liksom affärssystem, måste kunna klara av stora kvantiteter av användare i en testprocess utan att mer omfattande förändringar skall behövas.

- **Omfattande arkitektoniskt omfång** - Företag har ofta en brett omfång av operativsystem och mjukvaruapplikationer som skall testas. Prestandatesten måste vara flexibla och kunna exekveras simultant på ett affärssystem.
- **Lätta att använda och förstå** – System administratörer har inte tid att analysera komplicerade prestandatester. Ett prestandatest måste vara lätt att installera, exekvera samt förstå.
- **Vara representativt för det system som skall testas** – Prestandatester måste på ett korrekt vis representera det system som skall testas. Till exempel, om systemet som skall testas är en databas, så måste prestandatestet reflektera datamätningar med hög validitet, förbättrade accesstider hos databasens kunder, samt andra faktorer som är viktiga för det generella handhavandet av databasen.
- **Precision** – Tester som utförs med prestandatester måste vara precisa, och reflektera de krav som ställs på affärssystem. Om ett prestandatest inte är precist, så kan ett företag snabbt drabbas av för hög belastning vilket kan orsaka missnöje hos kunder samt förlust av förjänster.

Några andra faktorer som också är viktiga enligt Butler och Farell vid prestandatest är följande:

Datapunkter (Data points)
Datapunkter kallas resultatet från ett prestandatest. De representerar sammanfattningen av de specificerade resultaten från en användarbelastning. Datapunkter genererar den detaljerade informationen som behövs för att kunna fatta korrekta antaganden gällande kapaciteten hos det system som testas. Ett exempel på en datapunkt är ’transaktioner per sekund’. Datapunkter måste definieras före prestandatestet implementeras.

Repeterbarhet (Repeatability)
Repeterbarhet representerar resultat från multipla exekveringar av samma prestandatest med konsistent konfiguration av testmiljö och parametrar. Om ett prestandatest inte är tillförlitligt gällande repeterbarhet, så har resultaten enligt Butler och Farell ingen mening.

Korrekt statistik
Korrekt statistik är viktigt under ett prestandatest. Resultatdata från testen, förser användaren med det underlag som krävs för att avgöra om så kallade ’flaskhalsar’ eller potentiella problem beträffande affärssystemet kan förekomma.

Efter denna introduktion som jag hoppas har givit en givande första inblick av prestandatester så skall vi nu nedan ta itu med själva prestandatesterna. Först i form av en specifikation.

3.9.2 Middleware Companys specifikation för bland annat prestandatester

3.9.2.1 Källa
Allt material under avsnittet 3.9.1 härrör från följande källa om inget annat anges: (The Middleware Company Application Server Plattform Baseline Specification).

3.9.2.2 Specifikationen
Denna specifikation som omnämns samt även beskrivs till viss del i avsnittet ’ 3.9.3 J2EE and .NET (RELOADED) – Yet Another Performance Study (J2EE)’ nedan kommer här att presenteras övergripande för att ge en inblick i hur en referensram som är avsedd för bland annat prestandatester kan vara utformad.

Specifikationen är till sin karaktär en både funktionell och beteendemässig beskrivning över en applikation som kan användas i fallstudier av olika slag. Detta för att i sin tur studera andra applikationer i olika applikationsserver-miljöer, inkluderandes J2EE och .NET. Applikationen i fråga kan vara av typen referensimplementationer som Petstore och Petshop som Middleware själva i sin fallstudie använde sig av. Vidare enligt Middleware så existerar idag inte sådan specifikation, vilket också är ett av deras syften att fylla den saknaden.

Specifikationen är enligt Middleware specifikations-baserad i motsats till implementations-baserad. Med andra ord så förväntas det i en fallstudie som följer den här specifikationen att kodbaserna är mycket lika varandra genom att i mesta möjliga mån specificera dess funktioner och beteenden. Dock inkluderar inte specifikationen faktisk programkod.

Bidrag är enligt Middleware välkomna men endast så länge som de rättar sig efter denna specifikation.

Den övergripande filosofin är att alla testade implementationer måste vara hundraprocentigt funktionellt likväl beteendemässigt överrensstämmande med varandra. Detta åstadkoms enligt Middleware genom att följa denna specifikation. Men däremot så kan olika optimeringar och implementations-arkitekturer testas och jämföres med varandra på en och samma referens-hårdvaruplattform.

Det tillåter vidare enligt Middleware att varje bidragande team som använder denna specifikation som grund kan verkställa vad de bedömer som det bästa angreppssättet för att bygga programmet som är avsett för deras plattform. Men de understryker samtidigt med eftertryck vikten i villkoret som de ställer att all källkod och testskript för varje implementation som testas samt skripten för databelastning skall publiceras. Så länge som det villkoret uppfylls så kan utvecklare enligt dem granska och diskutera bidragen för att därmed bättre nå förståelse för de avvägningar som varje Anggreppsätt innebär. Detta även med tanke på programmet i frågas komplexitet, underhåll, hur välskriven koden är samt många andra faktorer.

Enligt Middleware så brukar faktumet att all kod skall visas upp avskräcka i de fall där det förekommer specialskriven kod som därmed är optimerad för tillfället och således skall kunna åstadkomma bättre prestandaresultat. Koden vid dessa tillfällen fortsätter de, brukar inte heller följa generella designprinciper för program eller rekommendationer om bästa handhavande från den aktuella plattformens återförsäljare.

Till sist så vill Middleware genom ytterligare förtydligande till det som sagts ovan ge en djupare insyn i att det här är inte en rent funktionell specifikation, utan också en beteende- och strukturell sådan. Som än gravare förtydligande vill de framlägga följande: att den specificerar hur databaser och tabeller är strukturerade, hur de används, vilka operationer som måste genomföra en läsning eller skrivning till databasen, eller operationer som måste det eller inte (beroende på så kallad 'caching'), var data kan bli 'cachad', hur mycket data som kan bli 'cachad' och hur länge etcetera.

3.9.3 The Nile Ecommerce Application Server Benchmark - Microsoft prestandatest

3.9.3.1 Källa
Allt material under avsnittet 3.9.2 härrör från följande källa om inget annat anges: (Nile Ecommerce Benchmark, 2001)

3.9.3.2 Introduktion
Nile Bench Bencmark är ett komplett testprogram i form av en enkel webb-baserad applikation, ett så kallat ’benchmark’. Det täcker i sina tester hela systemet, ’end-to-end’, samt är figurerat som en e-handelsapplikationsserver. Det har blivit vida använt av oberoende testlaboratorier inkluderandes Doculabs, eWeek och PC Magazine för att utföra prestandatester på applikationsserver-produkter. Nile ville därför använda samma program för att mäta prestandan och skalbarheten hos ASP.NET för en typisk dataintensiv Webbapplikaton.

Följande implementationer av Nile Benchmark programmet användes:

- ASP.NET skrivet i C# (Windows 2000)
- ASP och VB6 COM+ (Windows 2000)
- ISAPI/ATL Server C++ (Windows 2000)
- JSP, optimerad J2EE-baserad version som kördes på en vid tiden för de här testen framstående Java applikationsserver (Linux 7.1)
- EJB, optimerad J2EE-baserad version som kördes på en vid tiden för de här testen framstående Java applikationsserver (Linux 7.1)

3.9.3.3 Nile programmet
Nile programmet är konkret utformat som en enkel bokhandel på nätet vars syfte är att framhäva användandet av en applikationsserver för produktkataloger, specifik sökning efter produkter, bläddrande bland produkter, handhavande av kundkonton, så kallade ’kundvagnar’ samt ordertransaktioner. Slutligen så är inte säkerheten något som prioriteras i detta program utan alla användare loggar in anonymt. I ett ’riktigt’ program så skulle säkerligen sidan med kundkonton och motsvarande sida med transaktioner använda sig av exempelvis protokollet HTTPS (HyperText Transfer Protocol Secure).

3.9.3.4 Testplattformen
Testplattformen bestod av följande hårdvara:

- En Compaq ProLiant 8500 databas
- En Compaq ProLiant 8500 Webb/applikations-server
- Servern konfigurerades med två separata gigabit nätverkskort för att kunna hantera klientbelastningen
- Så kallat ’gigabit network backbone’ från Cisco. Nätverket uppdelades i två subnät om femtio klienter vardera
- Etthundra Dell datorer i form av arbetsstationer som skulle användas för att generera klientbelastningen
- Mjukvarorna Windows 2000 Server samt programmet ’Benchmark Factory 2.5’ från Quest hade som uppgift att generera klientbelastning

- En så kallad ’Benchmark Factory Controller’ för att administrera testerna, samla in och analysera resultaten

3.9.3.5 .NET implementationen

.NET implementationen använde sig av ASP.NET med så kallad ’inline’ C#-kod i Webbformulär med mellanskiktets komponentlogik isolerad i en separat .NET-sammansättning. Implementationen brukade ADO.NET dataläsare (i motsats till ett så kallat ’dataset’) för optimal prestanda. Den här implementationen använde sig även av .NET-versionen av SQL server för dataaccess.

3.9.3.6 ASP/COM+ implementationen

Den här implementationen använde ASP (VB script) sidor för att aktivera COM+ komponenter skrivna i Visual Basic 6.0.

3.9.3.7 ISAPI/ATL server implementationen

Den här implementationen använde sig av C++ för att implementera Nile som en högpresterande ISAPI applikation.

3.9.3.8 J2EE Java Server Pages versionen

Den här implementationen använde sig av JSP sidor i form av förkompilerade så kallade ’servlets ’ som kördes på den vid testet senaste (anno 2001-10-21) versionen av en då framstående J2EE-baserad applikationsserver. Implementationen var skapad och optimerad av en oberoende återförsäljarexpert (ISV) av J2EE-produkten, och var dessutom utvecklad enligt de klart definierade riktlinjerna som fanns för konstruktion av högpresterande program. Det implementerades mot en av de rekommenderade, och vid den här tiden mycket frekvent använda RDBMS- (relational database management system) databaspaket som stöddes av applikationsservern, användandes den rekommenderade tunna JDBC drivrutinen för den här databasen.

3.9.3.9 J2EE Enterprise Java Beans versionen

Denna implementation använde sig av JSP sidor för att aktivera EJB-komponenter som inkapslade all affärs- samt dataaccess-logik hos mellanskiktet. EJB:erna var implementerade som statuslösa sessionsbönor enligt de rekommenderade riktlinjer som fanns för den testade applikationsservern. Det implementerades mot en av de rekommenderade, och vid den här tiden mycket frekvent använda RDBMS-databaspaket som stöddes av applikationsservern, användandes den rekommenderade tunna JDBC drivrutinen för den här databasen.

3.9.3.10 Testen

Nile använde sig av programmet Benchmark Factory 2.5 från Quest Software för att generera belastningen samt mäta prestandaresultaten. Programmet levererades med förkompilerade skripts vars funktion var att testa Nile programmet och validera alla returnerade sidor för eventuella fel. Skripten simulerade en blandning av sex olika typer av användare som utförde en rad transaktioner vilka i sin tur var modellerade efter verkliga användarmönster av programmet. Bland skripten kunde man finna användarlogin, skapande av användarkonton, sökning gjord för specifikt ändamål, lägga till produkter till den så kallade ’kundvagnen’ samt placera ordrar. Nile ville för testen använda sig av ett tredjepartsverktyg som var konstruerat för att användas med både Microsofts och J2EE:s applikationsservrar. Med den mjukvara som de valde så kunde relativt få klienter med lätthet pressa systemet ordentligt.

För att maximera denna omtalade stress så kördes alla tester utan någon så kallad ’think time’, det vill säga den mer mänskliga egenskapen att ha viss tid till att tänka efter på olika sidor etcetera.

Nile använde sig vidare av olika användarbelastning i testerna och även övervakade i samband med det inte bara svarshastigheten utan även felnivån. Detta för att kontrollera så att ingen server returnerade felaktigheter under testerna. Nile körde upp mot 750 virtuella användare utan ’think time’. Det var nog för att passera punkten för maximal prestanda gällande antalet betjänade sidor per sekund för samtliga teknologier som testades, och även fullständigt översålla CPU-användandet hos mellanskiktet i samtliga fall.

3.9.3.11 Resultat med så kallad ’ output cache’ funktionalitet

Nedanstående tabell 3:3 visar hur många sidor per sekund som de olika teknologierna klarade av inkluderandes ’output cache’:

Tabell 3:3. Tabellen visar hur många sidor per sekund som de olika teknologierna klarade av inkluderandes ’output cache’-funktionaliteten (Källa: Nile Ecommerce Benchmark, 2001)

Clients	**ASP.NET C# Windows 2000 SQL Server 2000**			**JSP Major J2EE App Server Linux 7.1 Major RDBMS**			**EJB Major J2EE App Server Linux 7.1 Major RDBMS**			**ASP/COM+ Visual Basic 6.0 Windows 2000 SQL Server 2000**			**ISAPI C++/ATL Server Windows 2000 SQL Server 2000**		
	2CPU	4CPU	8CPU	2CPU	4CPU	8CPU	2CPU	4CPU	8CPU	2CPU	4CPU	8CPU	2CPU	4CPU	8CPU
1	25	27	27	71	79	89	64	67	53	42	45	44	250	259	258
25	663	661	661	834	1280	1395	619	890	890	469	874	993	1568	2043	2089
50	1301	1328	1328	828	1246	1393	647	859	900	462	857	1260	1633	2825	3190
100	1399	2661	2657	834	1230	1361	667	846	892	463	815	1243	1628	2847	3968
250	1387	2713	4004	803	1190	1330	638	834	860	464	844	1135	1629	2858	3989
750	1393	2643	3884	760	1178	1290	580	790	820	478	821	1129	1619	2867	3999

3.9.3.12 Resultat utan ’output cache’ funktionalitet

De huvudsakliga testerna utfördes av Nile och som tidigare har nämnts med ’output cache’ funktionalitet när teknologin i fråga tillät det. ASP självt hade inte stöd för det. Nyttjandet av den här teknologin på en dynamisk webbsida kunde avsevärt minska belastningen på mellan- samt dataskikten, medan behandlandet av antal sidor per sekund förbättrades avsevärt. Men enligt Nile så borde man observera att alla sidor inte kunde använda sig av denna teknik.Nile tyckte hur som helst att det skulle vara användbart att se hur varje teknologi klarade sig prestandamässigt utan denna funktionalitet, det vill säga när varje enskilt anrop behandlas. Detta skall nedanstående tabell 3:4 visa:

Tabell 3:4. Tabellen visar hur många sidor per sekund som de olika teknologierna klarade exklusive ’output cache’-funktionaliteten (Källa: Nile Ecommerce Benchmark, 2001)

Clients	**ASP.NET C# Windows 2000 SQL Server 2000**	**JSP Major J2EE App Server Linux 7.1 Major RDBMS**	**EJB Major J2EE App Server Linux 7.1 Major RDBMS**	**ASP/COM+ Visual Basic 6.0 Windows 2000 SQL Server 2000**	**ISAPI C++/ATL Server Windows 2000 SQL Server 2000**
	8CPU	8CPU	8CPU	8CPU	8CPU
1	27	59	50	44	236
25	664	685	576	993	3061
50	1318	663	551	1260	3319
100	2527	662	593	1243	3341
250	3093	641	572	1135	3386
750	3037	610	548	1129	3398

3.9.3.13 Sammanfattning
I denna sammanfattning som Nile gjorde av dessa prestandatester så konstaterade de att både skalbarhet och prestanda var oerhört viktiga aspekter vid val av en applikationsserver-teknologi. De fortsatte med att säga att deras kunder behövde prestandadata som hade hög validitet för att kunna fatta beslut grundade på korrekt information. Vidare hävdade de att deras kunder även behövde källkod och testskripten för att själva kunna återskapa resultaten.

Det trycktes även på att implementationerna som testas skall vara optimerade för den plattform som de fungerar bäst på för att testerna skall vara rättvisa.

Till sist visade prestandatesterna enligt Nile att den implementation av Nile:s prestandaprogram som använde sig av Microsofts ASP.NET var överlägset bättre prestandamässigt. Detta med hela 345 procent, än motsvarande applikation implementerad på en för den här tiden för testen ledande version av J2EE:s applikationsserver som nyttjade EJB:s. Vid det beskrivna tillfället användes åtta processors-system och ’output caching’ brukades av båda produkterna. De visade också att Micrsofts.NET:s version av Nile var enligt Nile vida överlägset, med 421 procent, motsvarande EJB-version användandes åttaprocessors-system när ’output caching’ inte användes.

3.9.4 J2EE and .NET (RELOADED) – Yet Another Performance Study

3.9.4.1 Källa
Allt material i det här avsnittet härrör från följande källa om inget annat anges: (Middleware Company Case Study, 2003).

3.9.4.2 Bakgrund
I Oktober 2002, publicerade The Middleware Company resultaten från en fallstudie som jämförde prestanda och skalbarhet mellan J2EE och .NET. Studien var i sin tur var baserat på Pet Store-applikationen. Resultatet från den här fallstudien skapade mycket oro inom J2EE:s utvecklar-community när det i studien visade sig att konkurrenten .NET hade varit överlägset J2EE. Detta kan i och för sig vara en rätt naturlig reaktion med tanke på att deras community bestod och fortfarande består mestadels av Java/J2EE-anhängare och utövare.

Middleware bemötte ovanstående kritik genom säga att de gjorde sitt bästa för att öppet diskutera meningen eller för den delen bristen därav beträffande deras resultat från studier och dylikt. De påstod också att de var mycket noga med att förtydliga i de många fall där resultaten inte enligt dem kunde eller skulle generaliseras.

De ville vidare i framtiden kunna hjälpa deras kunder, och för att kunna göra så anser de sig behöva vara både erfarna samt skickliga gällande både .NET och J2EE.

Till sist så var allt detta startskottet för vad som efter många förbättringar skulle bli både ett nytt prestandatest år 2003, men även att från allt det arbetet så skapade Middleware en egen specifikation för hur de anser att fallstudier skall genomföras.

3.9.4.3 Middleware:s mjuka värderingar i enlighet med deras specifikation
Processen som Middleware anammar, och som följer deras specifikation, vid tester är mindre formell, mer av sammarbetskaraktär samt öppen till sin karaktär. Företaget vill därmed

poängtera den grava felaktigheten i att det enda som skulle vara viktigt i sammanget skulle vara de slutgiltiga resultaten. Det är enligt företaget så långt ifrån sanningen man kan komma. De känner att de slutgiltiga resultaten från en fallstudie gällande prestanda är en viktig 'datapunkt', men inte den mest signifikanta aspekten av en sådan studie. Debatterna och diskussionerna mellan de olika deltagarna, medlemmarna av expertgruppen och community:n, noggranna granskningar av de beslut som gjordes beträffande specifikationerna och kodbaserna, dokumentation av erfarenheter, svårigheter och roliga anekdoter från fallstudien, öppna diskussioner med deltagarna och mycket mer är allt mycket informativt, mer lärande och viktigt än att bara se till resultaten.

Bör även nämnas att specifikationen enligt The Middleware Company är tillämpbar på fler kategorier av fallstudier än prestandavarianten.

3.9.4.4 Slutsatser som kan dras från denna, likväl som från prestandastudier i allmänhet
För applikationer som är specifikt avsedda för affärsbruk, så är prestanda viktigt. Men inte viktigare än mycket annat. The Middleware Company har upptäckt bland sina kunder att när de väljer server-applikationsplattformar för sina företag, så är prestanda inte ens bland de fem saker som de prioriterar som de viktigaste.

Andra överväganden som bland annat anpassning till standarder, till vilken grad produkten är en så kallad de-facto standard i sin klass, produktens eller teknologins stabilitet, hur lätt den är att använda, stöd för olika verktyg, den totala ägandekostnaden, skalbarhet och många andra liknande faktorer övervägde enligt The Middleware Company prestandan som kriterium vid val av plattform bland deras kunder.

Slutligen sa Middleware att gällande själva resultaten från prestandatest så var det ändå bäst att använda sitt eget omdöme, studera detaljerna i testerna samt använda dem i dess kontext.

3.9.4.5 Rapport
Den här rapporten innehöll resultaten från fallstudien (gällande prestanda). Till sin hjälp hade de en expertgrupp samt representanter för återförsäljare av J2EE- och –NET-plattformarna.

Fallstudien är baserad på Middleware Companys specifikation, som skapades av en inbjuden panel bestående av J2EE- och .NET industri-experter och professionella användare.

Fallstudien delades in i tre olika testområden:

- **Web-applikationstest.** Här testades prestanda vid användande av en typisk webb applikation som matades med en stadigt ökande användarbelastning. Testet utfördes med två olika databaser, Oracle 9i och Microsoft Server 2000.

  Testresultatet visade att både .NET och den snabbaste J2EE-plattformen var ungefärligen jämställda gällande detta deltest. J2EE-lösningen var aningen bättre än .NET-lösningen, omkring två procent bättre, när Oracle 9i användes. Men när Microsoft SQL Server användes så var istället .NET-lösningen ungefär elva procent bättre. Generellt så presterade båda J2EE-implementationerna lika bra mot båda databaserna. .NET-implementationen presterade så gott som likartat när Oracle 9i användes och något bättre vid användandet av Microsoft SQL Server.

- **Tjugofyra-Timmars Pålitlighets-Test.** Det här testet mätte den varaktiga uthålligheten samt pålitligheten för plattformarna över en tjugofyratimmarsperiod. Under denna period så kördes ett transaktionsintensivt testskript mot webbapplikationen som gjorde att en konstant maximal användarbelastning erhölls under testperiodens alla tjugofyra timmar. I testet så användes den databas för varje kodbas som den presterade mest tillförlitligt och övertygande med. För båda J2EE-implementationerna så användes databasen Oracle 9i. För .NET-kodbasen så nyttjades databasen Microsoft SQL Server.

  Resultatet från detta test var att den snabbaste J2EE- och .NET- plattformen åstadkom en nästan identisk prestation, endast mindre än två procents skillnad.

- **Web Services-test.** Det här testet prövade prestandan hos applikationsservern när den var värd för en grundläggande Web Service via protokollet SOAP 1.1.

  Testresultatet visade att .NET-plattformen utklassade den snabbaste J2EE-plattfomren med över tvåhundra procent**.**

3.9.4.6 Kategorier som användes i studien
Applikationen som beskrevs under avsnittet ’3.9.2 Middleware Companys specifikation för bland annat prestandatester’(The Middleware Company Application Server Plattform Baseline Specification) kan implementeras i vilket programmeringsspråk, på vilken applikationsserver-platttform, användandes vilken databas som helst.

Men specifikationen definierade även (gjorde så år 2003) tre kategorier som beskrev olika sorter av applikationen mer ingående.

Dessa tre kategorier var:
- J2EE-EJB-CMP
  Ett J2EE-program som har konstruerats med hjälp av J2EE, Enterprise JavaBeans, i synnerhet användandes CMP2 (Container Managed Persistance, v2).
- J2EE-SERVLET-JSP
  Ett J2EE-program som har konstruerats med hjälp av JSP och Servletfunktioner ihop med J2EE, men explicit ej användandes EJB:s.
- .NET-C#
  Ett .NET-program som har tillverkats enligt Microsofts guider för bästa praktiska handhavande för affärsrelaterad mjukvara, dock signifikant reviderad sedan Middlewares första test i Oktober år 2002.

3.9.4.7 Kodbaser som används i studien
Den här studien använde följande kodbaser för ovan beskrivna kategorier:

Kodbas för .NET-C# kategorin
För att kunna åstadkomma en kodbas för .NET-C# kategorin så gjorde Middleware en förfrågan till skaparen på Microsoft om denne kunde göra en ny version anpassad för 2003 års test av den .NET-kodbas som även figurerade i Oktober 2002-testet. Den versionen resulterande i kodbasen som kallades för msPetShop.

Liksom kodbaserna för de andra kategorierna, så sändes varje version av denna kodbas till expertgruppens webbsida för dess samarbete. Expertgruppen liksom även senare andra, inbjöds för att granska och koda den här kodbasen i enlighet med specifikationen och för särskilda prestandatest.

Kodbas för J2EE-EJB-CMP2 kategorin
Den här kodbasen grundar sig i den respons som erhölls från det första fallstudien gällande prestanda. Även responsen från den egna personalen införlivades. De förändringar som sedan genomfördes listades, samt den påverkan på prestandan som de åstadkom mättes informellt i små prestandatest. Denna påverkan tillsammans med en beskrivelse över varje förändring listades också. Kodbasen kallades för mPetStore.

Kodbas för J2EE-SERVLET-JSP kategorin
Kodbasen som användes i den här kategorin var open source varianten av JPetStore, en J2EE-implementation av Petstore som inte använder EJB:s. Kodbasen kallades för JPetStore.

3.9.4.8 Olika tester som genomfördes
Som man kan läsa ovan så genomfördes tre stycken tester. Dessa tester beskrivs även i denna fallstudie ytterligare en gång men då mer utförligt. Men på grund av att den här studien ändå i originalutförande är relativt omfattande och med tanke på min uppsats inriktning så väljer jag här att endast mer utförligt återge Web Services-testet.

Web Services Test
Det här testet mätte karaktäristiken beträffande prestandan hos Web Services för kodbaser i alla kategorier. Testet konfigurerades enligt följande: etthundra fysiska datorer som var och en simulerade ett antal användare som gjorde direkta SOAP-anrop mot Web Servicen. Testet mätte förmågan hos applikationsservern beträffande behandlandet av inkommande SOAP-anrop samt agerandet som en förmedlare av Web Servicen. Middleware kallade den här mätmetoden för ’Direct Activation of the Web Service’.

3.9.4.9 Applikationsservrar som användes i fallstudien

För .Net-C# kategorin:
För denna kategori så användes Microsofts .NET Framework 1.1, vilken även enligt deras påpekande också är en del av Windows Server 2003.

För J2EE-SERVLET-JSP och J2EE-EJB-CMP2 kategorierna:
För de här kategorierna så använde Middleware sig av två J2EE applikationsservrar, vilka kom att omnämnas som J2EE Applications Server X och J2EE Applications Server Y, detta för att undvika problem med de så kallade ’End-User License Agreements (EULA:s)’. Dessa licenser gjorde det nämligen svårt för Middleware att avslöja namnen på servrarna i samband med att resultat från prestandatesterna publicerades.

3.9.4.10 Testlabb och test-mjukvara
Middleware använde sig av testprogrammet Mercury LoadRunner 7.5 för att genomföra prestandatesterna.

Testerna genomfördes i ett storskaligt testlabb som bestod som sagt av etthundra fysiska klientmaskiner som var kapabla att generera en mycket hög samstämmig användarbelastning.

Ett så kallat ’gigabit backbone’ från Cisco användes, där varje server konfigurerades med två stycken gigabit nätverkskort varav var och ett av dessa hade kontakt med en underliggande uppsättning bestående av femtio klienter.

Två separata databasservers konfigurerades även på nätverket.

Databaser, klienter och nätverksanvändande övervakades under testerna för att vara helt säkra på att dessa inte blev flaskhalsar. Gällande alla kodbaser så rapporterade testerna förmågan hos själva applikationsserver-mjukvaran för det program som testades.

Det operativsystem som användes i alla testar var Windows Server 2003.

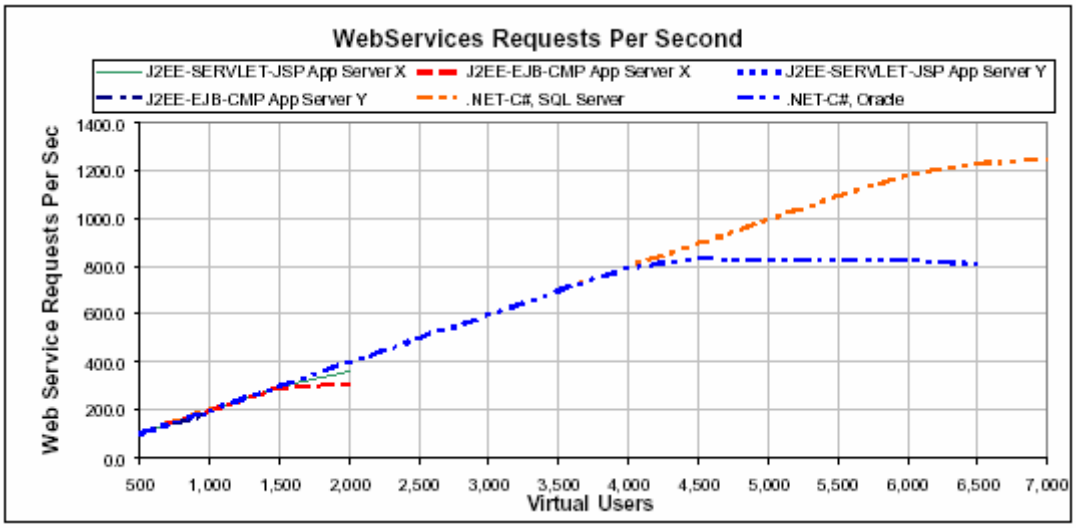
3.9.4.11 Web Services test resultat
I likhet med den mer ingående beskrivelsen av testerna ovan så återger jag här bara testresultaten från Web Services testet eftersom det är av mest signifikans för den här uppsatsen.

Beskrivning av testet
Den Web Service som ingick var specificerad så att den simulerade ett realistiskt testfall för att därigenom visa prestandan hos en applikationsserver inkluderandes aktivering av objekt över SOAP, så väl som serialisering av både enkla och komplicerade datatyper/objekt till XML.

I det här testet så sände klienterna ett SOAP-anrop gällande en slumpvis utvald order av tiotusen möjliga. Varje order bestod av fem upprepade varor i det returnerade orderobjektet. Detta satte stor stress på den enskilda applikationsserver som hanterade Web Service:n, och testet var gjort sådant för att det skulle visa hur väl applikationsservern presterade som en Web Service värd, hanterandes SOAP-anrop från tusentals samtidiga användare. All denna simulering åstadkoms med hjälp av testskript.

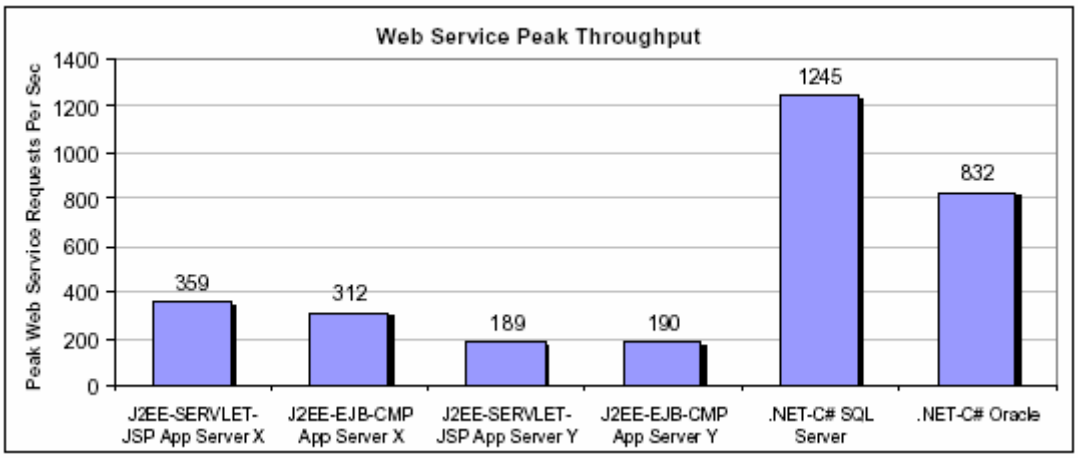
I detta test så användes den databas för varje kodbas som databasen i fråga presterade mest tillförlitligt och övertygande med i webbapplikationstestet, och även andra informella test. För båda J2EE-implementationerna så användes databasen Oracle 9i. För .NET-kodbasen så nyttjades databasen Microsoft SQL Server 2000.

Resultat av Web Services Test i siffror
Figur 3:5 och 3:6 nedan visar resultaten från testerna med webbskripten som var avsedda för Web Services. Där skripten genererade simulerad och ökad användarbelastning för de fem kodbaserna och deras valda databaser, och den maximala så kallade ’throughput’ (ungefär antal webbsidor per sekund) som de åstadkom.



Figur 3:5. 'Throughput' i form av antal anrop av Web Servicen per sekund vilken ökar i takt med att användarbelastningen också ökar för de olika konfigurationerna.

(Källa: Middleware Company Case Study, 2003)



Figur 3:6. Den maximala 'throughput' som har åstadkommits av varje konfiguration under Web Service Testet.

(Middleware Company Case Study, 2003)

3.10 Krav för att implementera system i verksamheten

Under detta avsnitt skall jag närmare granska vilka aspekter som kan vara viktiga att tänka på inför implementeringen av mjukvarusystemen J2EE eller .NET i den egna verksamheten. Det här kan vara en bra hjälp för uppsatsen målgrupp ifall de själva inte vet vilka aspekter som kan vara viktiga att tänka på och således inte heller vet vad inom dessa aspekter som är viktigt.

3.10.1 Kostnader för de olika plattformarna inklusive hårdvara med mera

Det finns en stor mängd J2EE-baserade implementationer på marknaden att köpa, med priser som varierar högst dramatiskt. Detta gör det möjligt för företag att välja den plattform som bemöter deras budget samt önskad servicegrad. Det förekommer gällande Implementationerna både mer avancerade lösningar liksom enklare motsvarigheter (Vawter & Roman, 2001).

Vad det gäller hårdvara så stödjer J2EE UNIX - och Mainframe system, medan både J2EE och .NET stödjer Win32-plattformen, vilket generellt sett är det mindre kostsamma alternativet av de båda. Poängen gällande hårdvara är annars just att det går att få en budgetlösning både med Microsoft- och J2EE-arkitekturerna. Microsofts lösningar har ett aggressivt pris, medan J2EE-arkitekturen tillåter dig att välja din servicenivå. Exempelvis, med J2EE så kan du ha antingen ha en avancerad och exklusiv lösning (iPlanet som körs på Sun Solaris i en E-10000 server) eller en mindre avancerad och därmed mer budgetorienterad lösning (jBoss som körs på Linux med en Cobalt RAQ server). Det finns också ett sortiment av fri och/eller open source-verktyg och tjänster som stödjer Java och XML. Det skall noteras att du betalar för vad du får, och de flesta organisationer vill inte ha lågbudgetlösningar utan tar hellre till sig ett system som är en gyllene medelväg mellan de två exemplen ovan (Vawter & Roman, 2001).

Det är rekommenderat att inte endast utgå från priset vid val av plattform. Det bör betänkas att priset är endast en droppe i havet i jämförelse med själva projektets totala kostnad. Det definieras i sin tur som priset för serverplattformen, kostnaden för att utbilda utvecklare, kostnaden för att utveckla och konstruera en lösning på den valda plattformen, kostnad för underhåll av den lösningen, och troligtvis även kostnader för förlorade affärsmöjligheter relaterade till felaktigt val av plattform. Så kort sagt se till andra faktorer än endast priset (Vawter & Roman, 2001).

Som ett komplement till ovanstående diskussion gällande prisbild vid val av plattform så skall jag här presentera några relativt färska prisexempel gällande .NET-plattformen. Daterat 2003-05-16 så var priset för en serverkonfiguration som inkluderade Windows 2000 Server och MS SQL Server från cirka 15 000 SEK till ungefär 100 000 SEK för de mer avancerade lösningarna (Dustin, 2003). Ett billigare alternativ kan då vara en typisk Linux-distribution. De enklaste av dessa finns exempelvis tillgängliga för nerladdning från Internet, alternativt köpas på CD-ROM skivor för några hundralappar (Arnoldsson, Lupander & Jönsson, 2003).

Dessutom får man även beräkna att Microsofts produkter ofta karaktäriseras av ett enklare handhavande och känns många gånger lättare att initiellt förstå sig på än motsvarande Linux produkter. För trots att man eventuellt får betala ett högre pris för en Microsoft-Server än för en motsvarande för Linux så kan Linux-lösningen på grund av sin komplexitet medföra att man måste exempelvis anlita en dyr konsult för att få igång servern (Arnoldsson, Lupander & Jönsson, 2003). Dock kan man säga till Linux försvar att handhavande, gränssnitt etcetera blir allt enklare för varje ny version som kommer även om det än så länge är en bit kvar till användarvänligheten hos Microsofts produkter.

Som ett avslutande argument så skall jag uppmärksamma Microsofts nya form av licensavtal, som kan öka kostnaderna avsevärt. De nya licensavtalen går under namnet ’Software Assurance’ och omnämns av Microsoft som ett två-årsskydd där man förbinder sig till nya uppdateringar från dem. På grund av att man låser sig till en enda leverantör under så pass lång tid så innebär det att man samtidigt undviker billigare alternativ som kan dyka upp under tiden (Hillesley, 2001).

Den här typen av licens har kritiserats av ’The Infrastructure Forum (tif), Engelsk branschorganisation för IT-branschen som hävdar att det har ökat utgifterna för deras medlemmar mellan 94 och 130 procent för varje företag. Liknande kritik har framförts av Elite, ’British Computer Society´s ’ forum för IT-direktörer samt Imis, samfundet för professionella IT-chefer. Tif uppskattar att de ändrade licenserna kommer att kosta deras

medlemmar, vilka inkluderar alla större företag som är verksamma i Storbritannien 800 miljoner pund över de fyra kommande åren (eftersom källan är skriven år 2001, så alltså till och med år 2005) (Hillesley, 2001). Eftersom denna förändring gällande Microsoft-licenser även har skett i Sverige så är det något att ta under beaktning vid val av deras produkter.

3.10.2 Inlärningskurva

Enligt Arnoldsson, Lupander och Jönsson (Arnoldsson, Lupander & Jönsson, 2003) så kräver mjukvaruplattformen J2EE en stor mängd befintlig kunskap samt en väl utvecklad förmåga att på egen hand leta efter samt finna mer kunskap för att kunna på ett tillfredställande sätt börja att använda J2EE och dess relaterade tekniker. De poängterar också att i den fallstudie de genomförde där det fanns tid så att de grundligt kunde sätta sig in i de tekniker de behövde. Som de fortsätter så är det inte alls säkert, att det i en företagsverksamhet skulle finnas de möjligheterna.

3.10.3 Skalbarhet

All material i detta avsnitt kommer från följande källa: (Vawter & Roman, 2001)

Skalbarhet är ytterst viktigt vid en expansion av Web Services-verkställanden över tid, på grund av att man kan aldrig förutsätta hur nya affärsmål kommer att påverka användartrafiken.

En plattform är skalbar om en ökning gällande hårdvaruresurser resulterar i en motsvarande linjär ökning gällande stödet för en ökad användarbelastning medan svarstiderna kvarstår som de samma som innan. Med den här definitionen, så är den underliggande hårdvaran (Win32, Unix eller Mainframe) ointressant när det gäller skalbarhet, och det med anledningen av att både J2EE och .NET tillåter addering av ytterligare maskiner som följd av ökad användarbelastning medan svarstiderna kvarstår. De större/mer kända implementationerna av J2EE-arkitetkuren, likväl som .NET, har en funktionalitet som kallas för ’load-balancing’ vilken medför att ett kluster av maskiner tillåts att samarbeta och underhålla en användarbelastning som kräver skalbarhet över tid.

Den signifikanta differensen mellan J2EE och .NET beträffande skalbarhet är att sedan .NET endast stödjer Win32-plattformen, så krävs det ett större antal maskiner än för ett motsvarande J2EE-verkställande och detta med anledning av processorbegränsningar. Den mångfald av maskiner det därför kan innebära kan vara svårt för organisationer att frambringa.

3.10.4 Support

3.10.4.1 Support gällande stöd för verktyg, protokoll och dylikt

Idag, stödjer J2EE Web Services genom Java API för XML-parsing (JAXP). Det här API:et tillåter utvecklare att utföra vilken handling som helst med Web Services med hjälp av manuell parsing av XML-dokument. Till exempel kan du använda JAXP för att utföra handlingar med SOAP, UDDI, WSDL och ebXML (Vawter & Roman, 2001).
Ytterligare API:s är också under konstruktion. De här är API:er som skall underlätta för utvecklare när de skall genomföra Web Services-handlingar som anslutning till affärsregister, konvertera XML-till-Java och vice versa, parsing gällande WDSL-dokument samt meddelandehantering med exempelvis ebXML (Vawter & Roman, 2001).

En mängd J2EE-kompatibla tredjeparts verktyg är tillgängliga idag och det medför snabb utveckling av Web Services. Det fanns år 2001 minst sexton SOAP-implementationer som stödjer Java. Nästan alla av de här Implementationerna är byggda på J2EE (Servlets eller JSP). Det fanns då bara fem UDDI API-implementationer att tillgå, och fyra av dem stödde Java (IBM UDDI4J, Bowstreet jUDDI, The Mind Electric GLUE, and Idoox WASP). Tredjeparts mjukvaruåterförsäljare som Tradia, CapeClear och The Mind Electric, erbjuder också verktyg för utveckling av Web Services (Vawter & Roman, 2001).

Även Microsoft.NET erbjuder organisationer att bygga Web Services. Verktygen som följer med Microsoft.NET gör det möjligt att kvickt utveckla Web Services-applikationer. För att effektivisera detta snabba utvecklingsarbete använder man sig av den automatiska generering som ingår i .NET-plattformen av så kallade 'wrappers' för Web Services. Dessa skall nyttjas för existerande system. Du kan utföra handlingar användandes SOAP, UDDI och SDL (föregångaren till WDSL). Programpaketet för utveckling, Visual Studio.NET, förser oss med så kallade 'wizards' som genererar Web Services (Vawter & Roman, 2001).

3.10.4.2 Support gällande stöd från återförsäljare efter köp av plattform

.NET
Gällande .NET så finns det på Microsofts hemsida (Microsofts hemsida d, 2004)
dels så att man kan få hjälp med främst säkerhetsfrågor från Microsofts community. Det finns också säkert mer att hämta även i de olika forumen som huserar under hemsidan. Det är den fria delen, sedan finns det följande avgiftsbelagda supporttjänster som är indelade efter olika kategorier (Microsofts hemsida d, 2004):

Premier Support Service
Premiär support erbjuder en omfattande årligt servicekontrakt som täcker alla Microsoft produkter inom kundens företag

Partner-Offered Support
Microsofts Gold Certifierade Partners för supporttjänster är en utvald grupp av företag som har uppfyllt strikta partnerskapskrav från Microsoft.

Personal Support
Personal Support täcker användarprodukter, personliga operativsystem, och desktop applikationer. Inhandla Personal Support i paket om fem (olycks-)händelser eller betala på en per-(olycks-)händelsebasis.

Professional Support
Professional Support täcker alla Microsoft-produkter. Köp Professional Support om fem (olycks-)händelser eller betala på en per-(olycks-)händelsebasis.

Finns dessutom en hjälp-kategori som kallas för '**Advisory Services'** men den är bara tillgänglig för personer som är bosatta i USA eller Canada och därför utelämnar jag den kategorin.

J2EE
För J2EE blir det en lite mer komplex historia med tanke på hur många återförsäljare det finns knutna till plattformen. Men för enkelhetens skull så visar jag här de supportmöjligheter som finns via Sun:s hemsida (Sun:s hemsida, 2004). Det finns även där en fri del som är ett

fullkomligt myller av hjälpmöjligheter med olika forum och möjligheter att söka och ställa frågor. Det som jag koncentrerade mig på främst liksom för .NET ovan är vad Sun erbjuder utöver det. Sun har där något som kallas för 'Support Contracts', vilka jag skall kort och översiktligt redogöra för på motsvarande sätt som för .NET ovan (Sun:s hemsida, 2004):

System Support Contracts
Har fyra täcknings-nivåer som förser kunderna med teknisk support, speciell service som kan nås av kunder med denna kontraktsform från Sun:s hemsida och mycket mer. varje supportkontrakt täcker systemets hårdvara och operativsystem samt inkluderar även uppdateringar av operativsystemet.

Software Support Contracts
Har två täcknings-nivåer som förser teknisk support, access till kunskapsdatabas och uppdateringar av mjukvara från Sun.

Pre-Owned Equipment Support Qualification
Ta system från Sun som du redan äger och specificera upp dem och kvalificera dem därigenom för att inkluderas av support från Sun.

Sun Time-to-Repair Service
Sun Time-to-Repair Service knyter Sun till att utföra hårdvarureparationer hos kund inom en specifik tidsperiod - antingen fyra, sex eller åtta timmar.

Vendor Support Alliances
Sun samarbetar med respekterade återförsäljare av Sun-lösningar för att därmed på ett effektivt sätt kunna underhålla kritiska komponenter på kundens affärs-IT-lösning. Detta gäller för projektets hela så kallade livscykel, det vill säga: design, implementation, samt underhåll på dagsbasis.

Förutom ovanstående kontraktsformer så fanns det på hemsidan en länk med följande rubrik: '**Contact Sun Support & Services**'. Bakom den så fanns det telefonsupport, support för Star-Office version 6 (motsvarande Microsofts office-paket), resurser för utvecklare samt online-support där man kan skicka iväg en fråga fast det krävs att man är registrerad användare, hur man tar kontakt med Sun gällande konsulttjänster och mycket mera.

3.10.4.3 Sammanfattning gällande support
Det verkar som om både J2EE och .NET erbjuder en omfattande mängd av support, både vad det gäller den fria delen i form av bland annat artiklar, forum och dylikt samt även den kontraktsbundna och/eller avgiftsbelagda formen. Den senare har här granskats något mer ingående om än fortfarande översiktligt på grund av att det är väl den form som är mest aktuell i en professionell situation. Men har även tagit med vissa former som gäller för privatpersoner bara för att ge en helhet av det utbud som finns.

Vilken plattform som erbjuder bäst stöd är omöjligt att säga efter en så här kort introduktion och kräver dessutom mycket djupare granskningar än vad jag har haft möjlighet att genomför för den här uppsatsen. Det här avsnittet får därför ses som en översikt så att man sedan kan gå vidare med egna djupare efterforskningar.

3.10.5 Mognad (hur gammal och beprövad tekniken är)
Allt material under detta avsnitt kommer från samma källa: (Vawter & Roman, 2001).

Organisationer som inbringar en Web Services-plattform till verksamheten måste beakta mognaden hos den underliggande plattformen. En mindre mogen plattform som härrör från en tidigare generation är mer benägen att orsaka problem och fel. J2EE är som en yta ovanpå existerande J2EE-lösningar. J2EE-verkställanden är vid liv och hälsosamma, samt underhåller en mängd av olika kritiska affärsproblem idag. Men om man granskar under denna yta så finner man att det finns några identifierbara riskfyllda områden där J2EE saknar mognad:

- EJB som är försedd med så kallad 'automatic persistence' är fortfarande omogen
- Java Connector Architecture (JCA) är nytt
- Allt stöd för Web Services är nytt

För Microsoft.NET, så är det en något annorlunda historia. En del av .NET är baserat på Windows DNA, vilket också handhar en rad olika kritiska webbsidor idag och gör det också mycket framgångsrikt. Men ändå:

- Med den nya CLR:en, så har en ordentlig bit av den underliggande .NET-plattformen blivit omfattande omskriven.
- C# är nytt
- Allt stöd för Web Services är nytt

Sammanfattningsvis så verkar det som om J2EE är den mer mogna plattformen av dessa två. Det är sant att vissa nya funktioner i J2EE är nya och riskfyllda. Men, själva den underliggande strukturen hos .NET är en omskrivning, och hela C#-språket är helt nytt. Det representerar en risk jämfört med vad de nya J2EE-funktionerna. Det bästa med .NET är det tar bort beroendet med COM-registret. Men det värsta med .NET är att det kastar ut den existerande infrastrukturen. Därför är det rekommenderbart att vänta och se angående .NET. Det handlar ju ändå i det fallet om en första generationen av en mjukvara.

3.10.6 Portabilitet
Det här avsnittet skall handla om ett begrepp som kallas portabilitet och hur det verkar under de båda plattformarna J2EE och .NET. Allra först inleds avsnittet med en definition av begreppet. Därefter följer ett avsnitt som tar upp hur man kan utvärdera portabilitet med utgångspunkt i tre tänkvärda scenarios. Därefter följs det av en genomgång av hur begreppet portabilitet fungerar under först J2EE och till sist följt av motsvarande under .NET.

3.10.6.1 Definition av portabilitet
Med portabel menas när det gäller mjukvara att den har förmågan att kunna exekveras på ett antal olika hårdvaruplattformar. Eller något annorlunda uttryckt, att mjukvaran inte är beroende av någon speciell typ av hårdvara (Webopedia, 2004).

3.10.6.2 Utvärdering av portabilitet
När det gäller att utvärdera den portabilitet som vi i detta avsnitt talade om, så är det tre scenarios som kan vara värda att tänka på (Vawter & Roman, 2001):

- Om din firma säljer mjukvara till andra företag, eller om du har en konsultverksamhet, och dina kunder använder sig av en mängd olika plattformar, så är det rekommenderat att använda sig av J2EE-arkitekturen. Om du inte kan garantera att varenda en av dina

kunder kommer att acceptera en Windows/.NET-lösning, så begränsar du den omfattning som du kan sälja från kunder som använder UNIX eller Mainframe-plattformar. Det här accepteras sällan hos de flesta oberoende mjukvaru-utvecklare eller konsultföretag.

- Om å andra sidan dina kunder nyttjar Windows-plattformen, då går det lika bra att erbjuda som J2EE som .NET, sedan båda är kompatibla med Windows. Det som gäller i ett sådant läge är att samla mycket information från dina kunder så att deras val av plattform verkligen blir rätt.
- Om du tillhandahåller dina egna lösningar, då kontrollerar du verkställandemiljön. Det gör det möjligt att välja J2EE liksom .NET. Du kan du då exempelvis standardisera på Win32-plattformen om du är villig att leva med de för- och nackdelar som den plattformen innebär. Med en sådan utgångspunkt så är plattformsneutralitet oviktigt och du skall överväga andra faktorer när det gäller valet mellan J2EE och .NET. Dock kan ett varningens ord vara på plats. Det är omöjligt att förutspå framtiden och den kan innebära att exempelvis en helt annan plattform blir gällande. Är man då som i det här fallet låst vid en enda så innebär det då en mindre gynnsam situation på grund av att den egna portabiliteten inte överensstämmer med den nya.

3.10.6.3 Portabilitet gällande J2EE

En viktig skillnad mellan J2EE och .NET är att J2EE är plattformsoberoende vilket innebär att det kan användas av en rad hårdvaror samt operativsystem, så som Win32, UNIX och Mainframe-system. Den här portabiliteten, är en absolut realitet idag på grund av att den så kallade ’Java Run Time’-miljön (JRE), på vilken J2EE är baserad, finns tillgänglig på alla plattformar (Vawter & Roman, 2001).

Det finns en andra, mer omdiskuterad aspekt av portabilitet. J2EE är en standard, och som en sådan så stödjer den en mängd implementationer som exempelvis BEA, IBM och Sun. Faran med en öppen standard som J2EE är att återförsäljare inte är strikt hållna att följa denna standard, och därmed kan applikationernas portabilitet gå förlorad. CORBA, till exempel, hade inte någon möjlighet att upprätthålla så att CORBA middleware verkligen rättade sig efter standarden, och därför så uppkom det ett antal problem gällande just portabiliteten. I J2EE:s ’barndom’ så drogs även det med liknande problem. (Vawter & Roman, 2001)

För att underlätta en sådan situation som ovan så har Sun byggt en J2EE-kompatibelt test-uppsättning, vilken garanterar J2EE-plattformen överensstämmer med standarderna. Den här testuppsättningen är verkligen viktig på grund av att den garanterar portabilitet hos applikationerna. Vid den tiden som den här källan skrevs, år 2001, så var det arton j2EE-kompatibla återförsäljare men tyvärr väldigt många som inte var det (Vawter & Roman, 2001).

3.10.6.4 Portabilitet gällande .NET

.NET fungerar bara på Windows men stödjer teoretiskt utveckling med ett antal programmeringsspråk (en gång när väl stöd har utvecklats för dem). Även .NET:s SOAP-möjligheter gör det möjligt för komponenter från andra plattformar att genomföra datautbyte med .NET-komponenter. Medan få av elementen i .NET som SOAP och dess ’discovery and lookup-protokoll’ är framställda som publika specifikationer, så är kärnkomponenterna av ramverket (framework) som IL runtime, interna ASP+ komponenter, Win formulär och kontrakt för Webb formulär-komponenter etcetera, tillhandahållna av Microsoft. Dessutom kommer Microsoft vara den enda återförsäljaren av kompletta .NET- utvecklings samt så kallad ’runtime’-miljöer. Det har förekommit en press från utvecklare att Microsoft skall

öppna sina specifikationer, men det skulle i så fall strida strikt mot deras standard praxis (Farley, 2000).

3.10.7 Prestanda

När inget annat anges under detta avsnitt så kommer materialet från denna källa: (Vawter & Roman, 2001).

En plattforms prestanda anses som bra om den presterar acceptabla svarstider under en viss specifik användarbelastning. Vad som anses vara 'acceptabelt' varierar för varje affärsproblem. För att åstadkomma acceptabel prestanda, så är det viktigt att den underliggande infrastrukturen för Web Services tillåter dig att bygga system med hög prestanda.

Den primära flaskhalsen gällande konstruktion av Web Services är vanligtvis integrationen med bakomliggande databassystem. Skälet för det är att de flesta applikationer som är affärsrelaterade även är datastyrda system med mycket mer datalogik än affärslogik. Under förutsättning att databasen vanligtvis är flaskhalsen, så kommer alla taktiker som kan reducera databasbelastningen att resultera i en signifikant vinst gällande prestanda. Exempelvis J2EE reducerar databastrafik genom följande två taktiker:

Stateful business processes tillåter dig att vidhålla aktuellt tillstånd hos en affärsprocess i minnet, snarare än att skriva ut det tillståndet till databasen vid varje förfrågan.

Long-term caching (erbjuds av vissa implementationer) gör det möjligt för datan från en databas att bli undanlagrat, 'cached', under långa tidsperioder, snarare än att återläsas till databasen vid varje förfrågan.

Det skall noteras att både ' Stateful business processes ' och 'Long-term caching' måste användas försiktigt, och kan resultera i problem om de aktuella utvecklarna inte är rätt utbildade när det gäller att bedöma när man skall, och inte skall, använda de har funktionerna. Det här är den fundamentala skillnaden mellan J2EE och .NET:s attityder beträffande konstruktion av Web Services; J2EE:s fördel är att det ger programmerare mer kontroll över lågnivå-tjänster som exempelvis 'caching'. Välutbildade utvecklare kan förbättra dessa funktioner så att de deras kvalitet beträffande verkställandet höjs. Men det är av högsta vikt att utvecklarna är ordentligt utbildade när de gör de här avvägningarna, annars kan fel lätt uppkomma i systemen.

Det skall även nämnas att de två beskrivna ovanstående funktionera inte finns hos .NET (Vawter & Roman, 2001). Men .NET har ändå en fördel beträffande prestanda mot J2EE. Det handlar om faktumet att J2EE är väldigt starkt knutet till Java och J2EE kan även implementeras i en Windows-miljö, men eftersom .NET är optimerad redan i dess inneboende konstruktion beträffande Windows-miljön så fungerar Java mycket bättre prestandamässigt under Windows än vad det gör i den miljön med J2EE (Gustafson).

3.10.8 Säkerhet

3.10.8.1 Web-services och säkerhet

I detta avsnitt så skall säkerhetsfrågorna hos Web Services presenteras. Avsnittet inleds med en allmän introduktion till ämnet och dess problematik. Därefter följer genomgångar av säkerhetslösningar under .NET- respektive J2EE-plattformarna.

3.10.8.2 Introduktion: Web Services och säkra API mekanismer
Traditionella säkerhetsteknologier som används på Internet är ofta inte tillräckligt nog för Web Services. Det huvudsakliga problemet är deras transportberoende. Till exempel, så är den mest omfattande använda tekniken, SSL (Secure Socket Layer) bunden till kommunikationsändpunkten i nätverket. Mer precist, identitet kan bara bli tilldelad med kommunikationsändpunkten som vanligtvis dessutom delas med många andra Web Services (Svoboda, 2002).

Andra säkerhetsteknologier vid namn GSS-API (Generic Security Service Application Programmers Interface) och säkerhetsmekanismer baserade på GSS-API som SPKM (Simple Public Key Mechanism) samt Kerberos, är tillverkade specifikt för att användas i löst sammansatta arkitekturer. GSS-API:n är oberoende av både transport- och säkerhetsmekanismer (oberoende av säkerhetsmekanism innebär att det underliggande teknologierna för kryptering, sådana som representerar identifiering, och datasignering är helt inkapslade). Den mest använda säkerhetsmekanismen som används idag i samband med Web Services är fortfarande SSL, något som även bekräftas av Clemens Vasters - Chief Technologist på newtelligence AG (Clemens refererad i Arnoldsson, Lupander & Jönsson, 2003), men upptagandet av SPKM och Kerboros ökar. SPKM använder sig av asymmetrisk kryptering medan Kerberos nyttjar symmetrisk kryptering (Svoboda, 2002).

Sammanfattningsvis gällande denna introduktion så kan man säga att för att kunna erhålla ett skydd för Web Services beträffande integritet, konfidentiellitet och säkerhet så är det av högsta vikt att man applicerar en omfattande säkerhetsmodell (Microsofts svenska hemsida 2b). Hur mjukvaruplattformarna .NET samt J2EE har löst sådana här och liknande frågor skall nedan presenteras.

3.10.8.3 Säkerhetsaspekter under .NET

GXA
Allt eftersom Web Services har blivit mer globala gällande dess omfattning och kapacitet, så blir det även allt mer viktigt att förse dem med ytterligare funktionalitet som skall värna om dess pålitlighet och säkerhet. Efter att ha uppmärksammat behovet av att standardisera de här extra funktionaliteterna så designade Microsoft och gav ut en uppsättning av specifikationer, som innefattar början till vad som kallas för ’Global XML Web Services Architecture’ , eller helt enkelt GXA. GXA ses även som en ny vision av XML Web Services som kan förbättras för att klara av den komplexitet och skalbarhet som finns inneboende i den distribuerade datamodellen. (Microsofts hemsida e, 2002)

GXA baseras på fyra designprinciper som inkluderar (Microsofts hemsida e, 2002):

Modularity: GXA är byggt på modulära komponenter som kan användas för att skapa lösningar som erbjuder exakt den uppsättning av funktioner som behöver för den givna kontexten. När nya eller förbättrade funktioner eller färdigheter behövs, så finns modulära protokoll för det.

General purpose. GXA är designad för en bred mängd av användningsområden för XML Web Services, inkluderandes business-to-business (B2B), business-to-consumer (B2C) och peer-to-peer program.

Federated: GXA behöver inte centrala servrar eller centraliserade administrativa funktioner. Arkitekturen tillåter att kommunikation sker över så kallade ’trust relationships’(en överenskommelse om att den som kommunicerar, utför transaktioner eller dylikt med dig är den som hon/han utger sig för att vara.) och oberoende entiteter. Modulerna och protokollen i GXA gör inga förutsättningar gällande implementationsteknologin vid informationsändpunkten.

Standards-based. GXA är byggt på XML Web Service:s specifikationer och protokoll, inkluderandes SOAP och UDDI. Även i framtiden så fortsätter Microsoft att arbeta med olika samarbetspartners för att kunna expandera och standardisera framtida specifikationer. Dessa är nödvändiga för att XML Web Services skall kunna utföra datautbyten mellan olika plattformar.

GXA och säkerhet
Enligt Microsoft (Microsofts hemsida e, 2002) så har Web Services under åren, 2000-2002, verkligen vuxit till sig som fenomen. Under samma period så har också uppkommit ett växande behov av en omfattande säkerhetsmodell för Web Services, ett ämne som länge har varit obehandlat.

I april år 2002, så tog företagen Microsoft, IBM samt VeriSign det första steget mot att finna en lösning på ovanstående behov genom att publicera en specifikation som kallades ’WS-Security’. Specifikationen definierar en standarduppsättning av SOAP-tillägg, eller meddelandehuvuden (message headers), som kan användas för att implementera integritet och säkerhet i ett Web Services-program. ’WS-Security ’ erbjuder standardmekanismer för att utbyta säkra och signerade meddelanden i en Web Services-miljö. Det förser även utvecklare med ett grundläggande lager som skall hjälpa dem att konstruera Web Services som är mer säkra och kan utföra ett brett datautbyte mellan olika plattformar. (Microsofts hemsida e, 2002)

Som tillägg till nämnda ’WS-Security’-specifikation så har Microsoft och IBM också publicerat något som de liknar vid en vägkarta för säkerhet gällande Web Services, och som omnämns som ’Security in a Web Services World’. Dokumentet beskriver en utvecklande attityd mot säkerhet och definierar ytterligare, relaterade säkerhetsfunktioner hos Web Services inom det ramverk som har etablerats av ’WS-Security’-specifikationen. Företagen planera att fortsätta utveckla den här specifikationen i nära samarbete med plattforms-återförsäljare, programutvecklare, återförsäljare av nätverk och infrastruktur samt kunder. (Microsofts hemsida e, 2002)

Organisationer kan införliva de här nya specifikationerna, vid behov, till olika nivåer hos deras Web Services-applikationer. Andra specifikationer som är på förslag (Microsofts hemsida e, 2002):

- WS-Policy, WS-Trust, och WS-Privacy. WS-Policy kommer att definiera hur möjligheterna och begränsningarna hos säkerhetspolicies skall beskrivas. WS-Trust kommer att beskriva modellen för etablering av både så kallade ’direct’- och ’broken trust relationships’ (direct innebär att man får lita på den som utger certifikatet själv och brokered innebär att det finns en mäklare som går i god för att certifikatet tillhör den som certifikatet anger skall inneha detsamma) inkluderandes tredje parter och mellanhänder. WS-Privacy kommer att definiera hur Web Services framställer och implementerar rutiner gällande den egna integriteten.

- WS-Secure Conversation, WS-Federation, och WS-Authorization. WS-Secure Conversation kommer att beskriva hur man hanterar och autenticerar utbyten av meddelanden mellan parter, inkluderande utbyten av säkerhetsrelaterad information samt etablerar och erhåller sessions-nycklar. WS-Federation kommer att beskriva hur hantering och mäklande av så kallade 'trust relationships' i en heterogen förenad (federated) miljö skall gå till, inkluderandes stöd för förenade identiteter. WS-Authorisation kommer att definiera hur Web Services hanterar autoriserings- data samt policies.

En modulär attityd gällande säkerhet i samband med Web Services är nödvändig med tanke på den mängd system som utgör nutidens IT-miljö. Allt medan användandet av Web Services ökar bland organisationer som i sin tur använder olika sätt att bemöta säkerhetsfrågan, så erbjuder de föreslagna så kallade 'trust'- och säkerhetsmodellerna ett flexibelt ramverk för organisationer så att de kan interagera med varandra på ett säkert sätt. Detta angreppssätt möjliggör utveckling för både den säkra teknologin och dess appliceranden i olika affärssammanhang. Därav, så beskriver den ovan nämnda så kallade 'vägkartan' hur man stödjer både nuvarande och framtida angreppssätt gällande säkerheten. (Microsofts hemsida e, 2002)

.NET Passport
Som en avrundning av .NET-delen gällande säkerhet så skall jag även kort berätta om .NET Passport, detta för att det ofta omnämns i säkerhetssammanhang i samband med .NET och att presentationen inte skulle kännas helt komplett utan att ha nämnt Passport.

Passport är ett licensierat protokoll som gör det möjligt för användare att logga in på multipla webbsajter genom att autensiera sig själva en enda gång i en vanlig server. Det främsta problemet som har framkommit i sambandet med detta är att servern i fråga innehas av Microsoft och därför så måste man anförtro alla sina privata data och dylikt till dem och den säkerhet som den servern har (Hillesley, 2001).

3.10.8.4 Säkerhetsaspekter under J2EE

Säkerhetsanvändande: Autenticering, auktorisation och dataintegritet
Autenticering, auktorisation och dataintegritet är tre huvudelement hos en säkerhetsarkitektur.

Beträffande J2EE så är det teknologier som JAAS (Java Authentication and Authorization Service) som används för autensiering och auktorisation, och om just JAAS så finns det nedan mer att läsa om den teknologin. Tillvägagångssättet hos JAAS tillåter helt transparent integration av Web Services med J2EE applikationsservrar som redan använder teknologin för just syften gällande Autensiering samt auktorisation (Svoboda, 2002).

Dataintegritet är den tredje funktionen enligt de ovanstående huvudelementen hos Web Services säkerhetsarkitektur. Dataintegritet försäkrar att mottagen data inte har blivit modifierad medan den transporterades, och har även hand om datakonfidentiellitet. Vanligtvis så krypterad data användandes någon symmetrisk krypteringsalgoritm (till exempel 'Tripple DES' eller RC5) på grund av att det är mycket snabbare än asymmetriska metoder. Generellt, så används bara asymmetriska krypteringsalgoritmer för säkert symmetriskt nyckelutbyte (Svoboda, 2002).

J2EE och säkerhetsteknologier
Som en utveckling av det som diskuterats ovan så anser J2EE att följande tekniker och teknologier, varav JAAS redan har nämnts ovan, är viktiga delar i affärsrelaterad mjukvara. Paketen som står inom hakparenteser är inte formellt del av J2EE, men de är ofta använda tillsammans med det eller övervägs att inkluderas i framtida specifikationer (P5EE - J2EE Summary Information):

- JAAS (Java Authentication and Authorization Services)
- [JCE] (Java Cryptography Extension)
- [JSSE] (Java Secure Socket Extension)

Nedan skall jag i tur och ordning ge en kort presentation av dessa tre tekniker/teknologier.

JAAS
'Java Authentication and Authorization Service' (JAAS) är en uppsättning av API:s som kan användas av två skäl:

- För att autensiera användare, och därmed bestämma på ett pålitligt och säkert sätt vem som för tillfället exekverar Javakoden, oavsett om koden används av en applikation, en så kallad 'applet', en böna eller en så kallad 'servlet'.
- För auktorisation av användare, och därmed bestämma på ett pålitligt och säkert sätt vem som för tillfället exekverar Javakoden, oavsett om koden används av en applikation, en så kallad 'applet', en böna eller en så kallad 'servlet'.

JAAS autenticering är utförd på ett 'plug-in'-sätt. Det tillåter Java-applikationer att kvarstå som oberoende från underliggande autenticeringstekonologier. Nya eller uppdaterade teknologier kan 'pluggas in' utan att man behöver ändra i själva programmet. JAAS autenticering utökar den existerande säkerhetsarkitekturen hos Java som använder sig av en säkerhetspolicy för att specificera vilka accessrättigheter som är tillåtna för att exekvera kod. Den arkitekturen, som den är introducerad i Java 2-plattformen, är så kallat 'code-centric' (ungefär 'kodspecifik'). Med det menas att, tillstånden beviljades baserat på karaktäristika hos koden: var koden kom från och om den antingen var digitalt signerad och ifall så, av vem. I samband med integrationen av JAAS in i Java 2 SDK, så hanterar Javas API för säkerhetspolicies principbaserade förfrågningar, och den ursprungliga policy-implementationen stödjer tillträden (inloggningar) som blivit beviljade dessa nya policies. (Java Authentication and Authorization Service (JAAS) Overview)

Java Cryptography Extension (JCE)
'Java Cryptography Extension' (JCE) är en uppsättning av paket som erbjuder ett ramverk och implementeringar för kryptering, nyckelgenerering och nyckel-'överenskommelse' (en slags överenskommelse mellan sändare och mottagare vid krypterade överföringar) samt så kallad ' Message Authentication Code' (MAC) –algoritmer. Stöd för kryptering inklusive symmetrisk, asymmetrisk, block, och så kallade 'stream'-algoritmer. Mjukvaran stödjer även säkra strömmar (secure streams) och slutna objekt (sealed objects) (Java Cryptography Extension (JCE)).

Java Secure Socket Extension (JSSE)
The Java Secure Socket Extension (JSSE) är en uppsättning paket som erbjuder säkra kommunikationer över Internet. Det implementerar en Java-teknologi version av SSL- samt ’Transport Layer Security’ (TLS) – protokollen. Det inkluderar funktionalitet för datakryptering, serverautenticering, meddelandeintegritet, och som tillval finns klientautenticering. (Java Secure Socket Extension (JSSE))

4. Empiri

I detta kapitel skall jag redogöra för mitt eget test som är i form av ett mindre prestandatest gällande Web Services på mjukvaruplattformarna J2EE och .NET. Det hela börjar med en beskrivning av den testutrustning som användes, både gällande hård- likväl mjukvara. Därefter går jag genom hur jag har förberett testerna för respektive plattform. Det följs av hur själva testerna har genomförts. Därefter tar jag upp vilka förväntningar jag har på resultatet och vilka krav och kriterier som fanns på detta test vilka inte blev så många på grund av testets föga omfattning. Slutligen så redogör jag för resultaten av testet genom att först gå genom i vilken grad reliabilitet samt validitet uppfylldes i testet. Detta följs av att jag för varje plattform berättar om de intryck jag fått under testet, vilka kvantitativa resultat jag har kommit fram till samt jämför även varje plattforms resultat med dem från de två större prestandatesten som jag presenterade i teoriavsnittet.

Allra sist så är det ett avsnitt som kallas för ’Resultatanalys’ där jag så att säga knyter ihop alla delarna av detta test och försöker göra en total analys av det som man kommit fram till under hela detta kapitel.

4.1 Beskrivning av testutrustning

4.1.1 Mjukvara

Under denna punkt skall mjukvaran som användes i undersökningen redovisas. Den kommer att göras så ganska övergripande. För mer utförlig information om denna mjukvara hänvisas läsaren till respektive hemsida på Internet.

Nedan beskrivs den mjukvara som behövdes för, eller direkt med, plattformarna som placerades på serverdatorn. Därefter kommer följaktligen en redogörelse av den mjukvara som förekom på klientdatorn.

4.1.1.1 Mjukvara för serverdatorn: .NET-plattformen

.NET Framework 1.1

.NET Framework utgör ett ramverk som inkluderar de tekniker som behövs för att konstruera program och Web Services. Denna miljö och dess ingående delar möjliggör autonomi från hårdvaran beträffande olika utvecklingsspråk. Som ett exempel så kan en klass från språket C++ användas i språket Visual Basic och detta precis som om den var skriven från början i det språket (Johansson & Ledin, 2001). Dock gäller detta under vissa förutsättningar, såsom att .NET-standarden måste stödjas av de ingående språken och att den måste först kompileras till mellanliggande så kallad MSIL-kod (Microsofts hemsida 2001h refererad i Johansson & Ledin, 2001)

.NET Framework ersätter vidare många väl utprovade teknologier som är i produktion idag, där ibland exempelvis Microsoft Transaction Server (MTS) och COM+, Microsoft Message Queue (MSMQ) samt Microsoft SQL Server databasen. Förutom ersättandet av dessa teknologier så innehåller .NET Framework ett webb Services-lager samt förbättrat språkstöd för olika utvecklingsspråk (Vawter & Roman, 2001).

Microsoft SQL Server 2000 Enterprise Edition
Valet av databasserver var ganska enkelt med tanke på att jag använde ett Microsoft-operativsystem. Valet överensstämde även med systemkraven som ställdes av databasen.

Helt kort så kan man presentera Microsoft SQL 2000 Server Enterprise Edition som jag använde som en kompetent och snabb server för såväl själva databashanteringen som analysen av densamma. Den är även enligt tillverkaren mycket skalbar till sin natur, vilket kan användas för exempelvis växande e-handelsföretag med bland annat stöd för underliggande datorer med flera CPU:er (Produktportfolio för Microsoftprodukter c).

Microsoft .NET Pet Shop 3.0 Referensimplementationsapplikation

Om programmet
Syftet med det ursprungliga .NET studien var att ta Sun:s original – J2EE-applikation, Sun Java Pet Store, och implementera samma programfunktionalitet med Microsoft.NET. Baserat på .NET-implementationen av Sun:s bästa prövoprogram för övning på dess funktioner, så kunde man direkt jämföra Microsoft.NET:s teknologi med J2EE-baserade applikationsservrar. Detta kunde ske på en rad frontdatorer medan man lär sig om likheter och skillnader mellan de rekommenderade designmönstren för att bygga Web-baserade applikationer. .NET-applikationen är nu inne på sin tredje utgåva och är designad för att visa hur man med .NET på bästa sätt kan konstruera flerskikts-applikationer riktade mot företag, vilka kan behöva stödja en mängd databas- samt verkställandemodeller. .NET Pet Shop 3.0 har blivit omgjord i dess arkitektur baserat på responsen från den community som omger programmet gällande version 2.0. Detta även i enlighet med Microsofts så kallade ’Prescreptive Architecture Guidance’ (rekommendationer gällande mjukvaruarkitektur) så som den är publicerad på MSDN. Denna tredje version av programmet är även fullt kompatibelt med det så kallade ’Middleware Company Application Server Benchmark Specification’ – specifikationen, se även avsnittet ’ 2.9.1 Middleware Companys specifikation för bland annat prestandatester’ för denna specifikation (Leake & Duff, 2003).

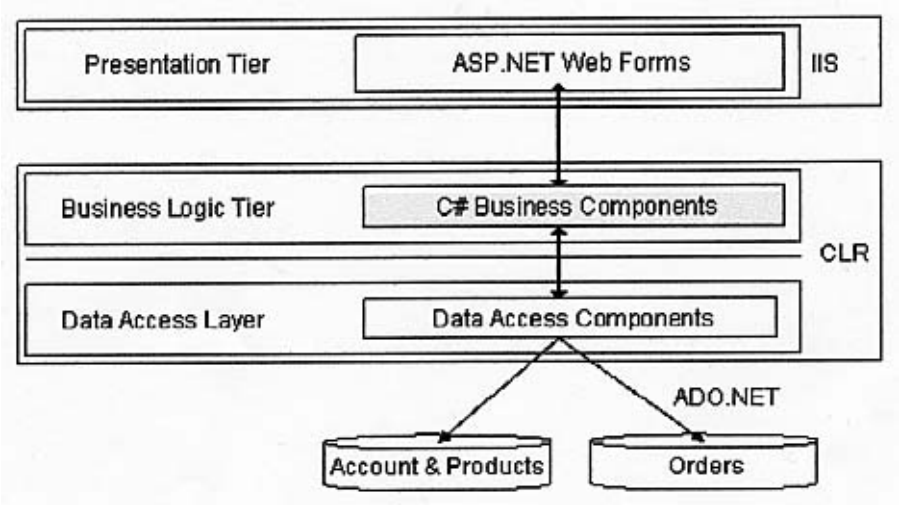
Beskrivning av programmet
Målet för .NET Pet Shop var att fokusera enbart på Java Pet Store (administrations- och mailkomponenter implementerades inte i .NET). Förutom att återskapa funktionaliteten hos Java Pet Store, så lades ytterligare två aspekter till (Leake & Duff, 2003):

- Jämföra och kontrastera koden och dess storlek på en verklig referensapplikation och hur det skiljer sig mellan .NET och J2EE.
- Ta fram data på hur många användare ett typiskt, väldesignat program kan stödja när det implementeras i J2EE och i .NET.

Den övergripande logiska arkitekturen av .NET Pet Shop visas i figur 4:1. Huvudfokus gällande designen låg på att använda ASP.NET Web formulär i presentationsskiktet vilket kommunicerar med C# affärskomponenter i ett logiskt mellanskikt. I sin tur så har affärskomponenterna tillgång till en back-end databas genom ADO.NET och en SQL Server hjälpklass. Dataaccessen är totalt abstraherad till ett ’Data Access Layer’ (DAL), som är separerat från ’Business Logic Layer’ (BLL). En nyhet med version 3.0 är att det finns ett DAL-skikt för databaserna Oracle 9i och SQL Server 2000. Så kallad ’Class loading’ av lämpligt DAL-skikt sker dynamiskt vid exekvering baserat på en programkonfigurationsinställning i inställningsfilen Web.Config. Notera att .NET Pet Shop 3

använder två backend databaser och behandlingen av ordrar involverar en distribuerad transaktion genom de här två databaserna. Användandes enkla programinställningar i inställningsfilen Web.Config, så kan användare genom verkställande av .NET Pet Shop få det att fungera tillsammans med enkla eller multipla backend databaser. Därmed kan användarna även fritt blanda SQL Server och Oracle backend databaser med den distribuerade transaktionen som hanteras av '.NET Service Components' med hjälp av COM+ 'Enterprise Services'(Leake & Duff, 2003).

Den tekniska beskrivningen ovan skall nedan illustreras i en logisk arkitektur över .NET Pet Shop. Med logisk menas här att den är abstrakt och visas bara som en generell modell över hur programmet fungerar.



Figur 4:1. .NET Pet Shop 3.0 högnivå logisk arkitektur.

(Källa: Leake & Duff, 2003)

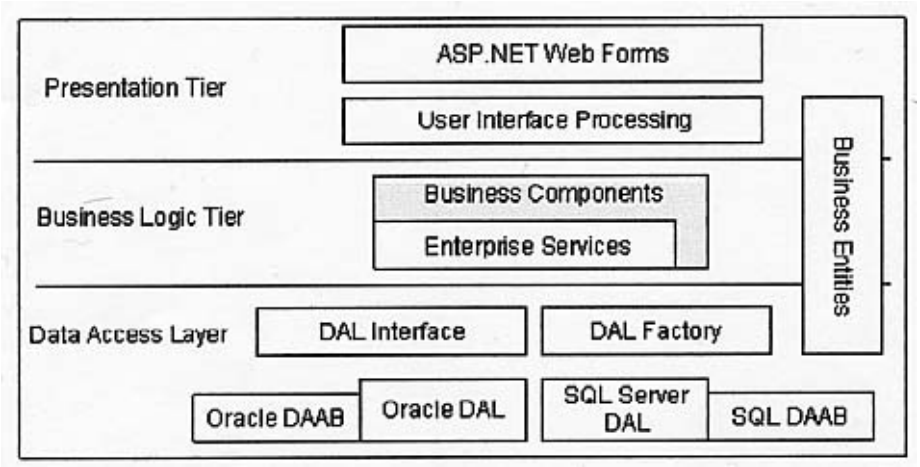
Funktionella krav för applikationen
Följande punkter beskriver de krav som ställs på programmet när det skall agera i exempelvis ett affärssystem (Leake & Duff, 2003):

- Programmet skall göra det möjligt för kunder att kunna söka i exempelvis en artikelkatalog antingen efter kategori eller enligt sökord.
- Programmet skall förse kunderna med en mekanism för att inhandla flera produkter genom en så kallad 'kundvagns'-modell.
- Applikationen skall erbjuda en enkel säkerhetsmodell så att kunderna är tvungna att logga in före de är tillåtna att köpa innehållet i 'kundvagnen'.
- Applikationen är tänkt att stödja en e-handelslösning av hög företags-klass som hanterar stora volymer av användare. Följaktligen skall programmet påvisa följande:
    - Hög prestanda, som mäts i form av antal betjänade kunder, och svarstider från användarna. Det vill säga hur lång tid det exempelvis tar från att en kund genomför ett moment tills det momentet når applikationen. För ett program som inte kan hantera stora volymer bra tar det längre tid och vice versa.
    - Möjligheten för expansion genom att öka antalet processorer.
    - Möjligheten att addera fler servrar och på så sätt bilda ett kluster.

- Med ett stort affärssystem så är det troligt att programmet måste ansluta sig till flera datalagringskällor, eftersom applikationen skall stödja distribuerade transaktioner.
- Programmet skall tillåta en flexibel verkställandestrategi. Initiellt så är applikationen konstruerad för att verkställa två fysiska maskiner, en applikationsserver och en databasserver men skall vara utbyggbar för att kunna jobba under andra verkställandemodeller. Programmet skall stödja många databasåterförsäljare. I denna beskrivning är det databaserna Microsoft SQL och Oracle som har valts.
- Programmet skall vara lätt att underhålla, vilket kommer att mätas i antalet kodrader i applikationen. Desto fler sådana desto mer komplext program och därmed svårare att underhålla.

.NET Pet Shop 3.0; Applikationsarkitektur

Nedan visas i figur 4:2 vilka applikationer som kan ingå i en .NET Pet Shop – arkitektur och det är som sagt var Microsoft SQL Server och Oracle som får stå som exempel på ingående databaser.



Figur 4:2. .NET Pet Shop 3.0 applikationsarkitektur.

(Källa: Leake & Duff, 2003)

Olika programområden inom Pet Shop – lösningen

I tabell 4:1 nedan så förklaras de applikationsområden som förekommer ovan i figur 4:2:s arkitektur:

Tabell 4:1. Applikationsområden inom Pet Shop-lösningen (Källa: Leake & Duff, 2003)

Område	**Syfte**	**.NET Implementation**
Komponenter för Användargränssnitt	Fånga data från användaren och visar returdata via back end system. De hanterar också enkel navigation. Omnämns som 'User Interface Components'.	ASP.NET Webb formulär, användar- och serverkontroller. Dessa konstruktioner möjliggör det för ren separation mellan HTML och UI – kod så som händelsehanterare av knappar.
Hantering av användargränssnitt	Kontrollerar navigation och processflöden med backend affärsobjekt. Hanterar också handhavandet av användares sessionsdata. Omnämns som 'User Process Components'	De här är implementerade som C# klasser. Sessionstatus-hanteringen sköts av ASP.NET
Affärskomponenter	Komponenter som implementerar affärslogiken för ett program	De här är implementerade som C# klasser.
Affärsentiteter	Tunna dataklasser för att skicka data mellan olika skikt i applikationen Omnämns som 'Entity components'	De här är implementerade som C# klasser med varje fält exponerat som en property (egenskap). Varje klass är markerad som 'Serializeable' (möjlig för serialisering) för att möjliggöra överföringen mellan processer
Komponenter för dataaccess-skiktet	Hanterar interaktionen med back end datalagring, inkluderandes databaser, meddelandesystem etc.	Hanterar interaktionen med back end datalagring, inkluderandes databaser, meddelandesystem etc. Implementeras som fyra C#-projekt: • En uppsättning av gränssnittsklasser för dataaccess-metod som vi vill exponera • En implementation av gränssnitten för SQL Server. • En Implementation av gränssnitten för Oracle 9i • En uppsättning av fabriksklasser för att ladda den korrekta implementationen, antingen SLQ Server eller Oracle.

4.1.1.2 Mjukvara för serverdatorn: J2EE-plattformen

J2EE 1.3.1

Det här är den version av J2EE-plattformen som jag använde mig av. Plattformen definierar enligt Sun (Java 2 Plattform, Enterprise Edition (J2EE)) en standard för komponentbaserade

multiskikts-applikationer med profilering mot affärsbruk. Den inkluderar även stöd för Web Services och utvecklingsverktyg, 'SDK'. För att läsa mer om J2EE-plattformen se avsnittet '3.6 Sun Soft J2EE '.

J2SDK 1.4.2
J2SDK (Java 2 Java 2 Software Development Kit) paketet innehåller den utvecklingsmiljö som jag nyttjade under mitt eget test: Sun:s utvecklingsmiljö i Java. Det är användbart för att utveckla Java-applikationer och erbjuder rätt exekveringsmiljö, 'runtime environment', för att exekvering av Javaprogram skall kunna fortlöpa smärtfritt (J2SDK-1.4.2). Jag använde denna utvecklingsmiljö just på grund av att jag behövde dess exekveringsmiljö.

JWSDP 1.3
JWSDP 1.3 (Java Web Services Developer Pack), är den version av Javas stöd för webservices under J2EE och som jag använde mig av i mitt egna test.

JWSDP är en fritt integrerad uppsättning av verktyg som är gjort för att javautvecklare skall kunna bygga och testa XML program, Web Services och Webb applikationer med Web Services teknologier och standardimplementationer (Java Web Services Developer Pack). I mitt fall så behövdes det här paketets stöd för att kunna nyttja Web Services.

Petstore 1.3.1 Referensimplementationsapplikationen
Allra först vill jag bara nämna att allt material under detta avsnitt kommer från denna källa: (Leake & Duff, 2003).

Java Pet Store är en referens-implementation av en distribuerad applikation som följer de 'ritningar', riktlinjer som upprätthålls av Sun. Referensprogrammet var från början konstruerat för att hjälpa utvecklare av arkitektur att förstå hur de skulle använda och kunna på verka J2EE-teknologier, samt hur varje J2EE-komponent passade ihop med de övriga i Java Pet Stores demoprogram. Applikationen inkluderar dokumentation, full källkod för den så kallade 'Enterprise Java Beans'- (EJB) arkitekturen, Java Server Pages (JSP), så kallade 'tag'-bibliotek, och Servlets som bygger applikationen. Som ett tillägg till detta så demonstrerar referensimplementsapplikationen vissa modeller och designmönster genom särskilda exempel.

Den fullständiga Java Pet Store innerhåller tre provapplikationer:

- Java Pet Store: Den huvudsakliga applikationen som följer de ursprungliga specifikationerna
- Java Pet Store Administrator: En administrationsmodul för Java Pet Store
- Blueprints mailer: En mini-applikation som presenterar några av J2EE specifikationernas riktlinjer gällande design i ett mindre paket.

Originalversion av Java Pet Store var designat för att fungera med följande databaser: Oracle, Sybase samt Cloudscape. IBM har tillverkade en DB2-port för programmet, alltså så man kan använda databasen DB2 med programmet.

Antagandet som gäller för huvud-programmet, Java Pet Store, är att det är en e-handelsapplikation där du kan köpa husdjur via Internet. När du startar programmet så kan du bläddra genom och söka efter olika typer av husdjur, allt från hundar till reptiler.

4.1.1.3 Mjukvara för serverdatorn: Operativsystemet Windows 2000 Server
Windows 2000 är den server som jag till slut kom fram till som den server som fungerade bäst ihop med de andra mjukvarorna jag använde samt tjänade även det syfte som jag ville uppnå.

Servern är mycket lätt att använda och har ett enkelt utförande men ändå de funktioner som man kan behöva för att driva en affärsrörelse idag, med tanke på dess Internet-intensiva anknytning. Tjänster av olika typer så som av: web-, nätverk-, fil- samt skrivkaraktär integreras enligt tillverkaren med ett handhavande och pålitlighet som sömlöst skall kunna verka genom hela verksamheten för en god integration av den egna affärsrörelsen med Internet (Produktportfolio för Microsoftprodukter a).

Mitt eget intryck av servern under den mycket korta period som den användes var att den i alla fall efter en första tids användande verkar uppfylla denna enkelhet i handhavandet. Vilket också stödjer min avsikt med den här uppsatsen där företagaren kanske inte alltid är extremt tekniskt insatt eller har råd att anlita konsulthjälp. Dock så vet jag av egen erfarenhet att de eventuella problemen först brukar uppkomma efter en längre tids användande då man hunnit pröva servern i fråga mer ingående.

4.1.1.4 Mjukvara för klientdatorn: Operativsystemet Windows Me och programmet Apache JMeter 1.9.1

Windows ME
Windows Millennium Edition eller Windows ME som det vanligtvis brukar kallas är ett mycket enkelt operativsystem nischat mot den genomsnittlige användare och är enligt min tolkning en mycket god ersättare till Windows 98SE. Även enligt tillverkaren så är dess mål att förenkla datorupplevelsen men ändå erbjuda ett brett utbud av stöd för hård- och mjukvara avsett för konsumentbruk.

Eftersom jag i mitt test ville försöka efterlikna verkliga förhållanden så mycket som möjligt med mina enkla resurser så tyckte jag att det här operativsystemet var ett mycket bra exempel på den miljö som den genomsnittlige användaren och därmed potentielle kunden för en företagare kan tänkas vistas inom (Produktportfolio för Microsoftprodukter b).

Apache Jmeter 1.9.1
Apache Jmeter ett renodlat Javaprogram för desktopbruk som är konstruerat för simulering av beteenden i testmiljöer samt även därvid utföra prestandamätningar. Det var från början designat för att testa webbapplikationer men har sedan dess expanderat till att inkludera även andra testfunktioner.

Jmeter kan användas till att testa både statiska och dynamiska källor (filer, Servlets, Perl script, Java objekt, Data baser och dito frågor, FTP servrar och mycket mer). Det kan också användas till att simulera exempelvis en kraftig användarbelastning på en server, ett nätverk eller objekt för att testa dess styrka eller att analysera allmän prestanda under olika typer av belastningar. Ett annat användningsområde är grafisk analys av prestanda för att testa beteenden hos servrar/objekt/skript under kraftig samstämmig belastning (The Apache Jakarta Projekt).

Skälet till varför jag valde just det här programmet var att det var ett av de få som jag kände till men det jag ändå hade hört talas om gällande programmet var att det skulle vara bra för

sådana här syften. Jag hade inte heller tid inom ramen för denna uppsats att testa fler program av den här karaktären och från dem välja det bästa.

4.1.2 Hårdvara

I denna undersökning har jag använt mig av följande datorer, och utrustning i dem, som är mina privata. På grund av att utrustning är min egen anser jag personligen också att den även är av ganska representativ karaktär för vad som man kan tänka sig att en egenföretagare/potentiell egenföretagare använder i sin rörelse i ett första skede. Det vill säga denne har liksom mig inte så avancerad eller omfattande hårdvara.

Dator som agerat som server i undersökningen:

Processor: Pentium 4
Processorkapacitet: 1,6 GHz
Arbetsminne: 512 MB RDRAM
Hårddiskens storlek: 60 GB

Dator som agerat som klient i undersökningen:

Processor: Pentium 2
Processorkapacitet: 400 MHz
Arbetsminne: 256 MB SDRAM
Hårddiskens storlek: 20 GB

Nätverkskort i båda datorerna är:

Märke och sort: 3COM Etherlink (PCI - kort)
Modell: 3C905CX-TX-NM
Kapacitet: 10/100 Mbps

4.2 Procedur

Inledningsvis för detta procedur-avsnitt skall jag redogöra för vad de ingående styckena innebär.

Mjukvaruinsamlande syftar till att införskaffandet av mjukvara inte alltid var det lättaste, i synnerhet som olika program krävde specifika andra program för att fungera. Det var minst sagt omständigt att hitta den rätta kombinationen men till slut lyckades jag.

Installation är som det låter kommentarer om hur väl det gick att installera programmen när jag väl hade funnit rätt kombination av dem. I föregående stycke prövade jag bara att se om det gick att installera, här genomfördes kompletta installationer.

Initialtest innebär ett första prov för att se hur de olika plattformarna fungerar, eller att de verkar fungera om man så vill. Man känner sig helt enkelt för en första gång.

JMeter-Problem är en rubrik som talar för sig själv. Ett av de mest komplexa problemen som uppkom under testen var att få JMeter att fungera och främst då att kunna göra anrop till server-datorn och webbservicen som fanns på denna. Jag bedömde detta som ett så viktigt moment att det således krävde ett eget stycke och rubrik.

4.2.1 Förberedelser och problem för .NET

4.2.1.1 Mjukvaruinsamlande
Införskaffandet av de mjukvaror som krävdes var allt annat än lätt på grund av att olika program krävde olika saker men kanske mer än det en svårighet att kunna anskaffa exakt det som krävdes. Först började jag med införskaffandet av .NET-mjukvaran och inledde helt logiskt med Petshop som är referensimplementationen av .NET. Enligt den så skulle operativtsystem som passade med den vara Microsoft Server 2003 eller Windows 2000 med SP3 (Sample Pack 3, paket med en rad förbättringar till operativsystemet). Båda dessa anförskaffades samt installerades på prov, dock fungerade bara Windows 2000. SQL server krävdes även och den version som införskaffades, Microsoft SQL Server 2000 Enterprise Edition, var tvunget att köras på ett operativsystem som benämndes som en server, sålunda så fick Windows 2000 kasseras. Efter mycket funderande provade jag ett mellanting som i och för sig var utanför systemkraven för applikationen men tänkte det kunde fungera, nämligen Windows 2000 Server med SP4. Mycket riktigt så fungerade denna. Övrig mjukvara för .NET såsom själva Petshop applikationen samt dess ramverk Framework 1.1 fanns att ladda ner från Internet.

4.2.1.2 Installation
Som tur var med tanke på ovan nämnda problem med mjukvaruanförskaffandena så gick själva installationsprocesserna smärtfritt.

4.2.1.3 Initialtest
Gällande serverdelen och själva Petshop så var det inga problem att lägga ett antal ordrar samt innan dess teckna två konton. Detta för att på så sätt placera poster i SQL databasen så att de kunde via webbservicen anropas med hjälpa av JMeter från klientdatorn.

4.2.1.4 JMeter-problem
Först infann sig ett problem med att förstå hur JMeter fungerar med avseende på Web Service- anrop. Första försöket skedde med en så kallad ’sampler’ för webbservice som automatiskt skulle generera SOAP anrop genom att tolka WSDL kod. Detta fungerade mycket dåligt på grund av otillfredsställande dokumentation gällande samplern och ej heller hjälpte upprepade försök att ändå få den att fungera.

Därför byttes den ut mot en annan sampler, den här gången med funktionaliteten att generera SOAP XML-RPC – anrop. Detta är en manuell metod som inte inkluderar ovan WDSL tolkande utan man matar in SOAP koden i samplern för att sedan genomföra anropet. Dock uppstod ett problem till när Web Service:n på servern klagade över utformningen av anropen med denna nya sampler. Det var enkelt uttryckt att den inte kunde förstå vilken typ av handling som skulle genomföras. Detta löstes med att lägga till ett så kallat konfigurationselement, till det befintliga anropet, som fanns i JMeter. Detta element bestod av kod som definierar den händelse som skulle utföras på servern, en så kallat SOAP action parameter. Efter detta kunde jag erhålla problemfria anrop, förutom att jag endast fick vad som verkade vara metadata. Detta var konfunderade med tanke på att man ej fick något felmeddelande i övrigt samt följt dokumentationen för förfarandet i fråga.

4.2.2 Förberedelser och problem för J2EE

4.2.2.1 Mjukvaruinsamlande
Det enda problematiken gällande införsamlandet var att det blev mer omständigt än vad jag hade tänkt mig från början på grund av felaktig dokumentation som förklaras i nästa stycke. All mjukvara fanns tillgänglig på Internet så ur den aspekten var det inga problem.

4.2.2.2 Installation
Den första installationen som gjordes av J2EE verkade till en början att fungera bra. Men när själva Petstore applikationen skulle installeras upptäcktes det att dokumentationen var för en äldre variant av J2EE som ej var kompatibel med den version av Petstore som vi skulle använda och dokumentationen medföljde till. För att åtgärda detta laddades ner en äldre version som var förenligt med den installationsinformation som gavs i dokumentationen. Därefter skulle webbservice-stödet för Petstore installeras och det var då som nästa problem uppstod. Problemet som gjorde sig gällande var att installationsinstruktionerna ytterligare än en gång gällde en äldre version av JWSDP (Webbservicesstödet) och icke-kompatibilitet uppstod. Än en gång fick därför förfarandet att från Internet ladda ner en version som överensstämde med Installationsinstruktionerna för Petstore utföras.

Intrycket av installationen var att det var mycket mer manuellt än dito för .NET. För att installationen skulle fungera var man tvungen att bland annat sätta egna miljövariabler för att på så sätt kunna köra skripten för installation av Petstore (så operativsystemet kunde hitta skripten). Kort sagt kan man säga att installationsintrycket av J2EE var att det var betydligt mer komplext att handha men samtidigt fanns det mer möjligheter till inställningar så som exempelvis flera javamiljöer (flera till exempel SDK:er).

4.2.2.3 Initialtest
När väl alla problem ovan var lösta och man fick mjukvara som överensstämde med Petstores dokumentation så var det bara att följa den. När det första försöket att pröva allting gjordes så fungerade systemet smärtfritt.

4.2.2.4 JMeterproblem
De webbservice-anpassningar som fanns från början i JMeter kunde man inte använda för att man inte hade exempel på SOAP-anrop. Det fanns helt enkelt inga instruktioner att följa som det fanns i dokumentationen till .NET . Detta omöjliggjorde tyvärr att kunna genomföra något test av J2EE-plattformen inom den snäva tidsram som fanns för denna uppsats.

Dock verkade webbservice-stödet för Petstore mer komplett än för .NET. Dokumentationen gav intrycket att all administration av databasen var möjlig via webbservices. Dokumentationen för .NET angav att det fanns bara en webbservice installerad som hämtar orderdata. Men det är mycket troligt att det bara gällde för testimplementationsapplikationen och om man köper ett fullfjädrat .NET system så finns det fler Web Services för det syftet.

4.2.3 Förfarande av tester
I JMeter kunde man bestämma något vars motsvarighet i tidigare test i teoriavsnittet har omnämnts som virtuella användare. Dess motsvarighet i JMeter var så kallade trådar, vilket motsvarar en process eller handling om man så vill. Det andra man kunde ställa in var hur

många gånger denna tråd, i fortsättning kallad virtuell användare för analogi med prestandatesterna under teoriavsnittet, kunde utföra handlingen. Min tanke från början innan jag blev lite insatt i hur JMeter fungerade var att man kunde öka antalet användare men ändå få ett konstant utfall, det vill säga att jag tänkte ha en, fem och tio virtuella användare och få dem att utföra en handling en gång. Alltså virtuella användare skulle bli variabeln och antalet handlingar konstanten, och ett lämpligt antal utfall på varje av dessa tre gruppar av virtuella användare bedömde jag till tjugo stycken. Ett antal som enligt min mening var tillräckligt högt för att ge reliabilitet. Det visades sig att antalet virtuella användare multiplicerades med antalet handlingar i JMeter vilket omöjliggjorde ett fast antal utfall per virtuell grupp. Det skulle ju innebära att när man skulle testa tio virtuella användare skulle man få tvåhundra utfall. Detta skulle dessutom generera en bias då det skulle vara olika antal värden, det vill säga antal utfall, i de olika mätningarna.

För att lösa detta problem så gjorde jag enligt följande. Vid varje test fanns alltså kravet om tjugo utfall. Dessa fördelades som så att antalet virtuella användare som anropades från JMeter anpassades så att dessa multiplicerat med antalet anrop utgjorde utfallsantalet till konstant tjugo stycken. Exempelvis för fem användare så gjorde således varje användare fyra anrop. Vid varje testtillfälle så användes en funktionalitet i JMeter som kallas listener som ”lyssnade” av svaren och den sort av listener som jag använde genererade en grafisk utskrift i form av en graf. För komplettering till grafen användes även en annan visualiseringsfunktion, en tabell, där svarstiderna kunde avläsas mer korrekt. Detta på grund av att de kunde ej så göras individuellt i grafen. Den hade helt enkelt inte den upplösning som krävdes för att göra noggranna avläsningar.

Till sist angående virtuella användare så skall det observeras att jag i den här undersökningen när det gällde en användare kallade det för just användare och ej virtuell användare. Detta på grund av att det i det fallet bara var en användare, det vill säga mig, och därmed inga virtuella diton.

4.2.4 Förväntat Resultat

Det är mycket svårt att säga exakt vad man förväntar sig av en undersökning som sker under så här begränsade förhållande och där jag inte hade någon tidigare erfarenhet utan har fått lära mig det mest nödvändiga väldigt snabbt för att så att säga pröva på att göra ett enkelt prestandatest med Web Services.

Dock har jag ändå några förväntningar på resultatet och det är att fler användare kommer att ge längre svarstider vilket jag även anser vara en relativt logisk förväntan. För .NET tror jag, i analogi med resultatet från Middleware Companys prestandatest gällande Web Services som jag har redovisat för under teoriavsnittet, att .NET kommer vara överlägset J2EE gällande prestanda även i den enklare kontext som mitt test äger rum inom.

4.3 Krav och kriterier

Antalet värden för varje test, det vill säg antalet utfall, bedömde jag som jag redan har beskrivit att tjugo stycken var ett lämpligt antal. Antalet tjugo därför som jag även det har nämnt innan, att det var ett bra antal att hantera praktiskt samt att det enligt min bedömning gav en god mån av reliabilitet.

4.4 Resultat
Till att börja med försvåras ju utlåtandet generellt sett med tanke på att den kvantitativa J2EE-delen av testen bortföll enligt nämnda skäl.

4.4.1 Reliabilitet
Reliabiliteten har jag inget konkret fastställande av gällande JMeter. Dock visade det sig att när jag för säkerhets skull ändå gjorde samma beräkning av svarstidernas medelvärde, median och standardavvikelse, som är de statistiska mått som JMeter producerade, med hjälp av Microsoft Excel så visade det på differenser. Vad dessa differenser berodde på visste jag inte men med tanke på att jag hade hört att Excel hade en relativt god precision, läs om detta lite längre fram i detta stycke, så verkade de värdena därför mer tillförlitliga. Jag visste inte exakt vad det här berodde på men kanske var det hur programmet internt hanterade de inkommande värdena som gjorde att de utkommande värdena som presenterades, det vill säga utfallen, blev felaktiga. Annat tänkbart var interna avrundningar som gjorde utfallen felaktiga. Men allt det här var endast spekulationer något konkret skäl kunde jag inte komma fram till. Men eftersom jag i mina tidigare studier hade använt mig av Microsoft Excel och det programmet hade fått från lärarna vid de kurserna i alla fall hyfsade omdömen gällande bland annat precision så använde jag mig av det för att kontrollräkna medelvärde etcetera och kom då fram till andra resultat än vad JMeter gjorde. Detta gör sammanfattningsvis att reliabiliteten vid mina tester enligt mina egna bedömningar inte är den bästa men det bör också observeras inte totalt förkastlig. Vid granskning av värdena får man dock ha en smula överseende men ej absolut enligt min mening totalt bortse från dem. Jag har i tabellerna nedan som presenterar testerna för en, fem och tio användare inkluderat värdena från både de ursprungliga testerna som JMeter presenterade samt de kontrollberäkningar med Excel som jag här har beskrivit. Detta för att läsare av denna uppsats själva skall kunna se den differens som jag här har berättat om.

4.4.2 Validitet
Avsikten med det här testet var att mäta prestandan hos en Web Service. Programmet JMeter (se även avsnitt om detta program) användes därför till att mäta den webservice som ingick i referensimplementationerna av Petshop respektive Petstore. Dessutom har samtlig dokumentation följts noga och det har använts en så kallad ’sampler’ i JMeter som genererar SOAP XML-RPC – anrop. SOAP är vidare ett XML-baserat protokoll avsett för just webbservices. Genom att ta hänsyn till alla dessa aspekter så är det min bedömning att jag har mätt det som avsågs att mäta och därmed uppnått god validitet.

4.4.3 Microsoft.NET

4.4.3.1 Intryck, upplevelser
Inledningsvis var det trasslig med tanke på de olika mjukvarornas systemkrav var svåra att förena för en gemensam fungerande kombination. Detta upplevdes som den mest problematiska delen gällande testgenomförande för .NET-plattformen. Även att få anropen att fungera med JMeter och så vidare upplevdes som en aning problematiskt men med tanke på sådana här systems generella komplexitet får man ändå hävda att det gick relativt smärtfritt totalt sett.

4.4.3.2 Siffror
Under detta stycke skall de kvalitativa resultaten redovisas tillsammans med förklaringar av de tabeller som de förekommer i.

Tabellförklaringar
Nedan följer förklaringar över de mått och värden som förekommer i tabellerna tillsammans med annan lämplig information.

De statistiska mått som finns i tabellerna är de som förekommer vid grafisk utskrift i JMeter och dels därav valet av just de måtten. Men även dels för att jag tyckte de gav en bra beskrivning av det som man kan få fram ur ett sådant här litet test. Man kunde använt andra statistiska mått som hade beräknats från värdena som genererades av JMeter. Men jag ansåg att eftersom det bara var ett litet test utan ambitioner på några större analytiska resultat, som dessutom mer skulle agera som en lätt jämförelse med de större testen så räckte det med de statistiska mått som presenterades av applikationen JMeter. Dessutom som man kan läsa sig till i denna uppsats så är förutsättningarna för de större prestandatesten helt annorlunda varav att mer ingående jämföra de kvantitativa resultaten ter sig meningslöst. Den tyngdpunkt som jämförelsen kommer att vila vid är snarare den från upplevelser och intryck. Det är enligt min bedömning mer direkt jämförbart och känns med tanke på de stora skillnader i förutsättningar som det område där testerna delar mest gemensam grund. Det är därför enligt min åsikt mest både relevant och meningsfullt att jämföra. Men hur som helst här följer alltså förklaringarna av de begrepp, mått med mera som förekommer i tabellerna från testerna:

Utfall.nr
I vilken ordning returvärdena kommer tillbaka till JMeter i samband med aktuellt anrop.

Anv antal/virt.anv antal
Vilket antal användare, benämns så när det endast är en användare, samt virtuella användare som förekommer i mätningen i fråga. Antalen som förekommer i den här undersökningen, det vill säga en, fem och tio användare fann jag representativa. Detta på grund av att jag tycker det är föga troligt för att inte säga väldigt sällan som mer än tio användare simultant anropar och dessutom utan betänketid (så kallad ’thinktime’). Förutom detta antagande fanns det heller inte tid med fler mätningar för att ta fram eventuella extremvärden eller dylikt statistiskt material.

Svarstid för anrop
Den tid angiven i millisekunder som det tar för JMeters anrop att färdas till destinationen (servern) och tillbaka (klienten).

Medel
Medelvärdet av alla anrops-svarstider. Anges i millisekunder. I tabellerna som presenteras här så är medelvärdet, liksom standardavvikelsen och median kontrollberäknade i kalkyleringsprogrammet Microsoft Excel. Medelvärdet i det här testet/fallet utgör ett mått på hur väl den applikation som testas i genomsnitt presterar.

Median
Beräknad median för de svarstider som JMeter erhållit. Anges i millisekunder. Median är det värde som mitt-observationen, svarstiden, antar. På grund av att medianen inte påverkas av extremvärden är den bra som ett komplement till medelvärdet (Körner & Wahlgren, 1996).

Standardavvikelse
Standardavvikelse för svarstiderna från JMeter. Anges i millisekunder. Enkelt uttryckt kan man säga att standardavvikelsen är ett samlingsmått för hur mycket observationerna, i det här fallet svarstiderna, avviker från det medelvärde som de alla har gemensamt. Brukar anses som

ett bra mått vid statistisk slutledning. (Körner & Wahlgren, 1996). Om den avviker allt för mycket från medelvärdet så kan det vara en indikation på att något inte stämmer.

Throughput medelv.
Ett andra mått som här presenteras på hur väl en applikation presterar är det så kallade 'Throughput'-värdet. Throughput brukar vanligtvis beskrivas som ett mått för antalet behandlade webbsidor per sekund, eller översatt till vårt fall antal anrop per sekund (Middleware Case Study, 2003; Nile Ecommerce Benchmark, 2001). JMeter anger detta som ett medelvärde av antal anrop per minut, medan det i andra prestandatester är vanligt att man anger exempelvis hur många webbsidor som kan behandlas i genomsnitt per sekund (Middleware Case Study, 2003). Troligtvis är detta relaterat till vilket program man använder för prestandatestet och dess kapacitet. Om denna är hög är det kanske lättare att ange det i sekunder för att på så sätt få mer hanterbara siffror.

Tabell 4:2. JMeter-mätning med en användare som gör tjugo anrop. Inkluderar kontrollberäkning med hjälp av Microsoft Excel

Utfall.nr	Anv.antal	Svarstid för anrop
1	**1**	170
2	**1**	220
3	**1**	220
4	**1**	170
5	**1**	220
6	**1**	160
7	**1**	270
8	**1**	220
9	**1**	170
10	**1**	170
11	**1**	160
12	**1**	160
13	**1**	220
14	**1**	280
15	**1**	160
16	**1**	170
17	**1**	170
18	**1**	220
19	**1**	160
20	**1**	220
Excel	**Medel**	195,5
	Median	170
	Std.avv	37,76311205
JMeter	**Medel**	210
	Median	220
	Std.avv	47
	Throughput medelv	269,66

Tabell 4:3. JMeter-mätning med fem virtuella användare som gör fyra anrop var. Inkluderar kontrollberäkning med hjälp av Microsoft Excel

Utfall.nr	Virt.anv. antal	Svarstid för anrop
1	**5**	380
2	**5**	220
3	**5**	330
4	**5**	610
5	**5**	220
6	**5**	220
7	**5**	160
8	**5**	330
9	**5**	380
10	**5**	330
11	**5**	110
12	**5**	270
13	**5**	600
14	**5**	110
15	**5**	820
16	**5**	990
17	**5**	280
18	**5**	110
19	**5**	270
20	**5**	160
Excel	**Medel**	345
	Median	275
	Std.avv	238,0291888
JMeter	**Medel**	340
	Median	280
	Std.avv	232
	Throughput medelv	437,95

Tabell 4:4. JMeter-mätning med tio virtuella användare som gör två anrop var. Inkluderar kontrollberäkning med hjälp av Microsoft Excel

Utfall.nr	Virt.anv. antal	Svarstid för anrop
1	**10**	220
2	**10**	280
3	**10**	270
4	**10**	220
5	**10**	220
6	**10**	770
7	**10**	380
8	**10**	870
9	**10**	1090
10	**10**	1090
11	**10**	1090
12	**10**	220
13	**10**	550
14	**10**	1320
15	**10**	170
16	**10**	270
17	**10**	220
18	**10**	390
19	**10**	280
20	**10**	470
Excel	**Medel**	519,5
	Median	330
	Std.avv	373,6234383
JMeter	**Medel**	518
	Median	380
	Std.avv	364
	Throughput medelv	485,83

4.4.4 Sun Soft J2EE

4.4.4.1 Intryck, upplevelser
Det totala intrycket av J2EE är att det kändes aningen föråldrat på grund av nyttjandet av installationsskript som tillhörde den äldre versionen vilka jag var tvungen att använda. Där användes enbart kommandoradsinmatning vid installation av Petstore. Detta kändes onödigt komplicerat och i och för sig har det ju blivit ersatt med ett GUI (Graphical User Interface), det vill säga ett grafiskt gränssnitt, i den nyare versionen men då ej korrekt dokumentation medföljde så gjorde det att användandet av detsamma försvårades betydligt.

På samma gång som man fick dessa negativa intryck så kunde det konstateras att systemet i stort verkade mycket flexibelt om man bland annat betänker antalet webbservices som följer med denna testapplikation. Dock lämnar jag en viss reservation för denna observation med tanke på att det är bara testapplikationer vi talar om och förhållandena i ett 'skarpt' system kan vara annorlunda. Dock gäller tyvärr inte denna reservation för dokumentationen.

4.4.4.2 Siffror
Uteblev på grund av nämnt skäl.

4.4.5 Jämförelse med tidigare prestandatester av.NET och J2EE
Under denna rubrik skall jag försöka att se på vilka skillnader det finns mellan det som jag har upptäckt och upplevt i mitt eget test med motsvarande för de två prestandatester som jag har beskrivit under teoriavsnittet beträffande de två plattformarna J2EE och .NET.

Först och främst så fanns det inte i någon av prestandatesterna någon beskrivelse över hur installationsförfaranden och dylikt har gått till, utan man beskriver mer andra tekniska skillnader som vilka olika implementationer, kodbaser och dylikt som ingår. Detta är även gemensamt för både J2EE och för .NET. Så där är direkt en skillnad mot mina tester för båda plattformarna jämfört med de två större testerna som jag har återgivit.

Beträffande .NET så är den mest märkbara skillnaden den att beträffande Web Services som bland annat testades i Middleware Companys test och som jag därför där lade tyngdpunkten vid gällande återgivningen av det testet, vida utklassade J2EE. När det gäller test av mer ordinära, det vill säga ej inkluderandes Web Services, webbapplikationer så visade testresultatet från Middleware Company att både .NET och den snabbaste J2EE-plattformen var ungefärligen jämställda gällande den formen av test. Denna uppgift motstreds av Nile-rapporten där testen av webbapplikation hade ett helt annat resultat och där .NET utklassade J2EE. Dock bör det sägas att för göra en helt rättvis och korrekt bedömning så måste man ta hänsyn till en rad faktorer och göra en mycket djupare analys än vad som kan göras här. Men jag tycker ändå trots differenser i konfiguration och dylikt mellan testerna att det är värt att noteras. Själv kan jag tyvärr inte vara med i just denna del av jämförelsen mellan de två plattformarna på grund av att J2EE-delen av mitt test bortföll.

Det mest suspekta av allt är att det förklaras gällande skillnaderna mellan dessa plattformar, alltså J2EE och .NET, i betydelsen rena prestandaresultat aldrig uttalat vad som ligger bakom de kvantitativa resultaten utan man konstaterar mer eller mindre att det bara är så. Nu vill jag lägga en viss reservation mot det här hävdandet på grund av att min erfarenhet av att tolka prestandatest inte är så stor och har aldrig gjort det innan jag började jobba med den här uppsatsen. Men jag har ändå granskat de båda prestandatesterna flera gånger och efter min tolkning och bedömning så finns det åtminstone ingen sådan information som man kan

lättvindigt hitta eller begripa från det de återger i sina testrapporter. Visserligen har ju kunnat konstaterats, som man kunnat läsa här i min uppsats gällande återgivningen av Middlewares rapport, att de har en fri attityd och de trycker på att just formella resultat som siffror och liknande inte säger allt, utan upplevelser, diskussioner med mera även är minst lika viktigt. Men även dessa mer ’mjuka’ värden representeras inte i den grad som man får intrycket av att de borde göra enligt deras egen specifikation utan förekom mycket sparsmakat i den rapport som jag har återgett under teoriavsnittet.

Nile-rapporten är lika sparsmakad den och där finns även inte den attityd som jag personligen gillar hos Middleware Company. Samma attityd som jag beskrev i föregående stycke och som i har sin grund i deras specifikation. Niles rapport har mer den klassiska prestandatest-atmosfären och därför mer stel och inriktar sig om ej inte enbart så i mer utsträckning på siffror, men även där fortfarande utan att förklara skillnaderna mellan J2EE och .NET. Det är den uppfattnings som jag har fått av de rapporterna och bör väl sägas att Nile ändå inte är extema när det gäller den kvantitativa utformningen av sin rapport men jag får ändå känslan att den präglar dess natur.

För att återgå och jämföra med mina egna kvantitativa resultat och upplevelser så kan jag ju på grund av att halva mitt test bortföll, J2EE-delen närmare bestämt, inte redovisa de kvantitativa resultaten från detsamma på grund av det. Sedan har jag konstaterat att reliabiliteten för JMeter i sig är tveksam i sin pålitlighet vilket gör att jag anser att det inte är någon idé att diskutera siffror här. De är ändå svåra att jämföra på grund av de stora skillnaderna i förutsättningar som finns mellan mitt och de två andra testerna. Som exempel på detta så använder de sig av 100 klientdatorer mot sina servrar och dessutom adderat till det simulerar de än fler virtuella användare. Jag hade som bekant totalt sett två datorer varav en var klient. Därför skall jag nu försöka jämföra lite upplevelser istället, vilket känns mer relevant.

Nåväl för att ändå ge mig på ett försök gällande upplevelser trots allt, så kan jag säga att .NET-plattformen rent intuitivt verkar likt flertalet andra Microsoft-produkter lättare att jobba med. Det har vidare ett mer tilltalande gränssnitt än J2EE, vilket även har kunnat konstaterats ovan under emperi-kapitlet. J2EE känns mer som man måste kunna mer innan man ens börjar med det men att möjligheterna när man väl har skrapat på ytan är desto fler. För att i sammanhanget använda sig av en närliggande parallell så stämmer det här i så fall precis överens med hur mina upplevelser mellan operativsystemen i Windowsfamiljen upplevs och Linux-diton, även om Linux har blivit bättre på användarvänlighet och gränssnitt. Det är faktiskt inte en så dum analogi med tanke på att det även där är Microsoft-världen mot open-source diton där J2EE-är en av dess representanter. Samma känsla får man när man jobbar med dem i alla fall. Med detta sagt så var min erfarenhet av hur det blev redan efter ett litet tag med .NET, som jag redan nämnt under empirin, likvärdig men den från samma företags operativsystem. Det verkar bra och lätt i början men alltmer som man jobbar med det och kanske redan bara efter en mycket kort tid så börjar man så smått upptäcka svårigheterna. Dessutom som jag också skrev under empirin så upplevdes .NET väldigt lätt i början men mycket snart så uppkom ju även där problem.

Sammanfattningsvis så är det svårt att säga att någon av dessa plattformar skulle vara bäst som Ni säkert förstår utifrån min väldigt korta tid med dem. Det kräver nog att man studerar dem mer och framför allt jobbar mer med dem under olika omständigheter och i olika kontexter innan man mer säkert kan uttala sig om något. Men det här var i alla fall mitt försök

att sammanföra de upplevelser, om än korta sådana, av plattformarna och jämföra dem med hur de framfördes på de redan existerande prestandatesterna som ingår i mitt teoriavsnitt.

Olyckligtvis så bortföll den bredd som jag hade hoppas på under denna jämförelse. Detta på grund av att det mesta av J2EE-delen gällande mitt egna test bortföll. Jämförelsen av upplevelser blev inte heller så omfattande till följd av deras föga förekomst i de prestandatest som jag har återgett under uppsatsens teoriavsnitt.

Men jag tycker ändå att det har varit upplyftande inblickar som har framkommit under det här avsnittet även om en direkt jämförelse mellan de båda tidigare rapporterna och mina upplevelser skulle ha genererat ett mycket bättre resultat.

4.5 Resultatanalys

4.5.1 Konkret resultat

Förvisso har väl redan resultatet presenterats ovan under J2EE respektive .NET-rubrikerna. Jag skall här ändå försöka sammanfatta dessa båda samt säga något övergripande om resultatet. Det som kan sägas är att jag får bedöma det sammanfattade resultatet av mitt test som gott. Detta trots alla problem jag har haft samt om man betänker under vilka enkla förhållanden som jag har gjort testerna.

Att exempelvis det inte fanns dokumentation till J2EE så att jag kunde haft större möjlighet att få med den plattformens kvantitativa värden i testet är sådana olyckliga omständigheter som händer. Samma gäller för strulet med att en äldre dokumentation till J2EE följde med en nyare version av det programmet. Det gör att resultaten är inte så utförliga eller breda som de kunde ha varit på grund av att mycket av jämförelsen gick bort. Men trots att man inte lyckades få igång J2EE så fick man i alla fall ett omfattande första intryck under installationsprocessen och fram tills det att det inte gick att fortsätta längre. I enlighet med Middlware Companys specifikation, som framhåller mjukvare värden, så kan jag säga att dessa upplevelser har jag tolkat som minst lika värdefulla som någon kvantitativ data.

4.5.2 Insamlat materials relevans

Enligt min åsikt så ligger det insamlade materialets relevans att i att det ger en insyn i hur ett enklare prestandatest kan utformas samt genomföras. Detta helt även i linje med Middleware Companys specifikation som också framhöjer kvalitativa och ej endast kvantitativa värden som exempelvis beskrivelser av hur man har upplevt testet.

Men just att det i mitt test verkligen tas upp upplevelser och hur det kändes för mig som testare när jag genomförde det tror jag även är en stor pluspoäng för min målgrupp. Detta därfär att jag tror de gärna ser ett mer familjärt uttryck än enbart tekniska termer och till exempel diagram. Visserligen blir det alltid en balansgång mellan dessa två världar men syftet har ändå varit att försöka få med den mer emotionella biten och inte endast den rent tekniska.

En annan aspekt som är betydande för detta tests relevans är dess avsaknad av externt inflytande från intressenter som exempelvis sponsrande företag. Det är ju i fallet sponsorer knappast troligt att man starkt kritiserar sin egen sponsors produkter. I synnerhet inte om det förekommer exempelvis stora summor pengar. Det gör ju att material som bedömer och beskriver produkter där sponsorer eller för den delen andra lojalitetsintressen finns med i bilden starkt kan ifrågasättas. Detta oberoende är relevant för uppsatsens målgrupp när de

skall välja mjukvaruplattform. I synnerhet om de saknar erfarenhet och därför inte själva kan göra en bedömning om yttre påverkan har förekommit eller inte.

4.5.3 Insamlat materials originalitet jämfört med existerande material

Mitt materials originalitet ligger i det faktumet att prestandatester på den här nivån, det vill säga under mycket enkla förhållanden, är något ovanligt. Dessutom det som ändå gör det verkligen originellt är att det har skett utan påverkan från några externa ekonomiska- eller andra lojalitetsintressen mot exempelvis en sponsrande leverantör eller dylikt. Visst finns det mindre prestandatester i var och varannan datortidning bara för att nämna ett exempel, men där kan det tänkas att det finns andra intressen bakom som kan vinkla testerna som till exempel ekonomiska ägandeberoenden. Allt som en människa någonsin gör vinklas ju på något sätt men en akademisk uppsats gjord av en vanlig student utan några ekonomiska eller andra bakomliggande intressen bör väl enligt min åsikt vara bland de kontexter som är mest befriat från sådant.

Slutligen att mitt test dessutom i kombination med det jag redan nämnt har en inriktning mot en såpass ny och aktuell teknologi som Web Services gör ju att originaliteten framhävs än mer. Det är dessutom min övertygelse efter att ha genomfört denna uppsats att det behövs än mer forskning inom detta relativt nya ämnesområde (se även avsnittet ’ 5.4 Fortsatt forskning’).

5. Slutsats

5.1 Konklusioner

Den här uppsatsen syfte var ju att fungera som en guide för främst företagare som utgör målgruppen men även naturligtvis för vilka intressenter som helst. Avsikten var också att dess normativa metodik skulle generera en bred och informativ grund med rekommendationer och dylikt. Dessa skulle sedan fungera som en bas för deras beslutstagande över vilken av de plattformar som jag i denna uppsats har valt att inrikta mig på, det vill säga antingen J2EE eller .NET, som var bäst för deras syften. Det gäller då naturligtvis dessa plattformar utrustade med Web Services som har varit ytterligare ett delområde i den här uppsatsen. Allt det här för att svara på uppsatsens frågeställning:

Vilken mjukvaruplattform som inkluderar Web Services är det bästa valet för en företagare?

Som det har kunnat konstateras på flertalet ställen i den här uppsatsen, som till exempel under styckena ’3.8.3.6 Val av plattform ’av Wavter och Roman (Wavter & Roman, 2001) samt David Chapell (Chapell refererad från Gustafson) eller under stycket ’3.9.4.4 Slutsatser som kan dras från denna, likväl som från prestandastudier i allmänhet’ av The Middleware Company (Middleware Company Case Study, 2003), så kan man inte enkelt svara på den frågan, utan får se till hur det egna företaget ser ut, vilka kunder man har, vilka krav de har, vad man själv har råd med och många andra faktorer. Dock är det en sak i sammanhanget som har återkommit gång på gång i den teori som jag har granskat, och det är att man skall inte stirra sig blind på kvantitativa resultat från exempelvis prestandatester och dylikt. Detta är något som i synnerhet Middleware Company hävdar i sin rapport och specifikation (se exempelvis avsnittet ’3.9.4.3 Middleware:s mjuka värderingar i enlighet med deras specifikation’). Det de säger där är att många andra värden minst är lika viktiga även om man för den skull inte heller skall bortse från just siffervärden. Några av dessa ’mjukare’ värden som nämns av dem är debatterna och diskussionerna mellan de olika deltagarna av prestandatesten. En enhetlig slutsats till som jag har kunnat dra är att ingen av plattformarna J2EE eller .NET kan därför enkelt kåras till någon slags vinnare, det är helt enkelt omöjligt. För ser man till uppsatsens målgrupp vilket är företagare med i många fall en stram budget och kanske också inte så stor kunskap inom ämnet så finns det ju mycket att tänka på vilket gör att det kanske inte finns något självklart val. Visst Microsoft kan te sig, vilket även som jag har skrivit var mitt första intryck gällande differenser mellan plattformarna, enklare gällande gränssnitt, handhavande och så vidare. Men samtidigt så får man ju se saken i ett längre perspektiv för det som kan verka vara svårt i början kanske kan fungera och vara en vinnare i längden. Min egen erfarenhet av J2EE är att det verkar vara mer komplicerat men verkar erbjuda ett större utbud av lösningar. Men det är mer spekulationer för det kräver än mer erfarenhet och insupande av teori för att kunna göra sådana konkreta och mer säkra antaganden än vad det har funnits tid till att göra under denna uppsats.

Ekonomiskt sett så kan man tycka att J2EE är ett bättre val. Detta motsägs dock på grund av nämnda komplexitet som kan göra att det behövs till exempel en konsult som installerar och undervisar köparna av systemet hur det fungerar. Det kan i sin tur bli hemskt dyrt, vilket delar av uppsatsens målgrupp med största sannolikhet inte ens kan klara av ekonomiskt. Ett sådant faktum kan därför motivera Microsofts högre pris. Sedan finns ju systemen i olika utföranden som också påverkar priset och eftersom J2EE har en mängd olika återförsäljare så uppkommer även problematiken med vem man skall välja som sin återförsäljare beträffande den plattformen. Vad det gäller Microsoft vilket jag även har nämnt tidigare så har de ju under senare år kommit med en ny typ av supportavtal, som också är en viktig bit att tänka över vid

val av plattform och återförsäljare. Dessa gör att man skall binda sig till dem för en massa uppdateringar och blir därmed låst till den enorma takt vilket det företaget presenterar nya produkter, uppdateringar och så vidare under den tid som avtalet gäller. Även J2EE har en rad olika utformningar av supportavtal som jag har skrivit om en del i uppsatsen. Det har även i uppsatsen tagits upp saker vid detta val som säkerhet där jag tycker plattformarna är relativt lika, mognad där en fördel nog måste gå till J2EE med tanke på att den plattformen har funnits längre och har hunnit utvecklas (mogna) vilket .NET på grund av det är mycket nyare inte har gjort än, samt prestanda.

Just prestanda har jag ju även valt att koncentrera mig till gällande uppsatsens empiriska del. Inte för att man skall koncentrera sig vid enbart det benhårt som jag precis här har dragit en slutsats att man inte skall göra. Men om man ändå skall granska den så har jag fått en antydan just när det gäller Web Services och de två granskade plattformarna att .NET kan eventuellt vara överlägset (för teoretiskt underlag se avsnitten: ’ 3.9.4.2 Bakgrund’, ’ 3.9.4.5 Rapport (punkten 'Web Services-test')’ samt ’ 3.9.4.11 Web Services test resultat’). När det gäller mer ordinära Webbapplikationer utan Web Services funktionalitet så har jag inte fått något så entydigt resultat från de två prestandatester jag granskat i teoriavsnittet utan två olika svar. Men för att återgå till Web Services så har det i uppsatsen enligt Hanson (Hanson, 2002) framkommit att .NET:s fördel i stort, och inte specifikt för prestanda, beror på att plattformen redan från början är konstruerad att fungera med Web Services och att J2EE är mer av karaktären att man ovanpå det befintliga systemet har byggt på med fler API:er (se avsnitet ’ 3.8.3.6 Val av plattform’). Så trots att jag bara har Middlewares rapport gällande prestanda och Web Services att gå efter så kom de ändå i år 2002:s rapport fram till samma resultat, det vill säga att .NET-plattformen var överlägsen i prestanda, som de gjorde i 2003 års prestandatest. Så just för Web Services så kan .NET-plattformen vara överlägsen J2EE (för teoretiskt underlag se avsnitten: ’ 3.9.4.2 Bakgrund’, ’ 3.9.4.5 Rapport (punkten 'Web Services-test')’ samt ’ 3.9.4.11 Web Services test resultat’). Men, och det är ett stort men, jag har som sagt var ändå bara lyckats få fram tidigare prestandatest gällande Web Services från en enda källa. Beträffande mitt eget prestandatest så är det främst av så annorlunda omfattning men också så bortföll ju J2EE halvan vilket omöjliggjorde en jämförelse fullt ut. Därför är det vanskligt att dra slutsatsen att .NET skulle vara överlägsen J2EE beträffande Web Services men en viss känsla av att det inte är en omöjlighet får man.

Så kort sagt en ren slutsats kan vara svårt att dra här men en vag antydan, med tanke på vad som sagts i föregående stycke, skulle jag ändå vilja säga att det finns gällande .NET:s överlägset inom prestandaområdet för Web Services i dagsläget. Men det är viktigt att observera att dessa antaganden är baserade på den ringa erfarenhet som jag både teoretiskt och praktiskt har erhållit genom den här uppsatsen. Det skulle krävas mycket längre studier och erfarenhet för att kunna göra några mer säkra uttalanden. Därför skall man sålunda inte dra för stora växlar av dessa uttalanden utan de får mer ses som spekulationer och känslor av hur det verkar efter att jag har gjort den här uppsatsen, inte hur det nödvändigtvis faktiskt är. Men för att återgå till själva diskussionen så kan man säga att .NET:s till synes överlägsenhet kan bero på dess nyare konstruktion, som nämndes i föregående stycke, vilket jag har fått en lätt aning om utan att hitta någon konkret bekräftan för det i den teori jag har funnit. Kanske man som sagt ännu skall låta detta vara osagt även om det kan vara en möjlighet.

En ytterligare sak som jag måste flika in med här och som skall observeras är att i uppsatsen så finns det fler aspekter gällande mjukvaruplattformar än de som har tagits upp här under slutsatser, men jag har därmed inte uteslutit dem på något sätt. Men däremot tar jag bara upp vissa exempel på aspekter, för de genomgående slutsatserna man ändå kan dra och den känsla

man får efter att ha arbetat med samtliga av dessa aspekter är ändå de samma. Slutsatserna som dras här gäller alltså även för de aspekter som inte nämns här. Det känns ifall man skulle ta upp alla aspekter att det bara skulle bli en massa upprepningar och onödigt omfattande.

Avslutningsvis så kan jag inte dra någon annan slutsats än den som jag först drog under det här avsnittet för min målgrupp, det vill säga att ingen av plattformarna J2EE eller .NET kan enkelt kåras till någon slags vinnare. Många i målgruppen kanske skulle tyckt varit bekvämt om det nu var möjligt, för att på så sätt underlätta det egna valet och ta bort en del av frustrationen som allt beslutsunderlag kan innebära. Det är dock som jag redan har kunnat konstatera i nämnd slutsats beroende helt enkelt på vad som är viktigast för just den enskilde företagaren i fråga och det är nog den viktigaste och mest betydelsefulla slutsats som man kan göra ifrån denna uppsats. Den här uppsatsen skall fungera som en guide på vägen men något absolut val åt någon kan den som sagt var omöjligtvis göra.

Men eftersom det här ändå är en normativ uppsats och det därför är avsikten att den skall ge rekommendationer åt uppsatsens målgrupp, så skulle mitt råd till sist och med tanke på de slutsatser som jag här har kommit fram till vara att målgruppen gör en egen liten analys. I denna analys tar de fram och tänker efter vilka behov de har, vilka deras kunder har, vilka deras kunder är, vilka framtida eventuella expansionsmöjligheter som det finns, vilka möjligheter som finns inom den egna budgeten, vilka plattformar de skall arbeta under och liknande frågor. Kort sagt analysera vilka de själva är, vilka deras kunder är och vad allas behov är och kan tänkas vara. Samt naturligtvis än en gång: fatta inga beslut enbart baserat på statiska kvantitativa resultat härstammandes från ett typiskt prestandatest eller dylikt, utan se mer till andra värden som finns inom en organisation.

5.2 Diskussion

Det första man kan säga om de resultat som har framkommit är att de i stort är ganska entydiga och pekar åt samma håll. Vidare så är även just det synsätt som därmed har kommit fram detsamma som Middleware Company i sin specifikation propagerar för. Det vill säga att mer i prestandatester och även anda sammanhang inkludera samt ta hänsyn till mjukare värden som informella samtal och inte enbart redovisa eller framhäva siffror och dylikt.

Om prestandatestet från Nile finns väl inte direkt så mycket att säga om. Det är mer av klassiskt snitt och faktiskt i stort av den karaktär som Middleware Company kritiserar. Det vill säga i stort för mycket inriktat på att framställa siffror.

Beträffande den omfattande teorin i övrigt så tycker jag i enlighet med vad jag tidigare har nämnt i uppsatsen att den fyller sitt syfte mycket väl som en bred och informativ grund. Med den eventuella reservationen som jag också tidigare har nämnt att det har varit en mycket svår balansgång mellan att det skall vara någorlunda lätt att läsa och förstå samtidigt som den tekniska substansen inte får gå förlorat på grund av det. Med risk att verka alltför repetitiv så har jag nämnt det här innan i uppsatsen men jag tycker ändå att det tål att poängteras än en gång. Detta på grund av att det är något som har varit en av uppsatsens stora svårigheter men som jag ändå hoppas att jag har lyckats med. Det vill säga att jag hoppas att jag har lyckats att förmedla även de svårare avsnitten till så många som möjligt både inom och utom uppsatsens målgrupp.

En annan sak om jag har nämnt förut men även den tål att ytterligare poängteras är bortfallet av J2EE-delen i mitt eget prestandatest under empirin. Det som är intressant att spekulera i är hur utfallet av den här uppsatsen skulle ha blivit om man hade fått med J2EE plattformen.

Vidare också kanske haft mer tid att lära sig båda plattformarna ordentligt och genomföra testet mer noggrant och grundligt. Det skall dock sägas att efter den föga tid som jag hade till mitt förfogande så gjordes det så korrekt jag kunde. Men som en ren spekulativ tankelek så är det intressant att tänka sig vart uppsatsen kunde ha landat och vad som kunde ha blivit resultatet då.

Dessutom gällande mitt eget test så skulle nog också utprovning av flera andra motsvarande program till JMeter vara på sin plats. Även det var något som gick bort av tidsbrist vilket gjorde att mätningarna och resultaten kanske kunde ha gjorts bättre med ett annat program. En tanke har ju även varit att trots att man granskade den dokumentation som medföljde JMeter så kunde kanske om man blivit mer bekant med programmet och fått provat det mer resulterat i annorlunda resultat. Möjligheten att ändå misstag gällande inställningar har förekommit kan man inte utesluta. Trots att JMeter ändå är ett förhållandevis enkelt program i sammanhanget så är teknologin det vill säga Web Services något mycket komplext vilket jag tror inte någon tvivlar på efter att ha läst genom den här uppsatsen. Just därför finns det möjligheter att vissa inställningar kunde kanske ha gjorts bättre.

Något som jag dock hade önskat varit med i större omfattning skulle varit open source alternativ. Det finns ett avsnitt där de tas upp specifikt och sedan i resten av uppsatsen nämns de lite här och där. Det finns en hel del intressant att utforska inom just den delen av mjukvaruplattformar samt att de representerar ett intressant alternativ. Dock så verkar det som om att just hitta material om dessa alternativ kräver mer arbete än för J2EE och .NET. Det är inte lika lättillgängligt men kanske just därför skulle det ha varit intressant att ha med mer. Men med tanke på den här uppsatsens omfattning så kanske det just beträffande den här uppsatsens utformning får räcka att open source alternativen figurerar som ett komplement som tas upp på ett begränsat sätt.

5.3 Metodutvärdering

Enligt min mening har den normativa metod som jag har använt mig av fungerat mycket tillfredsställande. Målet var ju att insamla en bred bas av information samt filtrera den och presentera den på ett sätt som var av mesta möjliga nytta för uppsatsens målgrupp. Jag tycker att den här uppsatsens bredd inom de områden som den har behandlat fyller sitt syfte till fullo och likaså sättet som den presenteras på. Det jag bör inflika här är väl den mycket svåra balansgång som finns i just presentationen av något så här komplext när det skall göras inför människor som kanske har en mycket föga kunskap om ämnet. Jag har därför bland annat förutom jobbat mycket på att förenkla svåra avsnitt men ändå utan att förlora substansen även inkluderat en omfattande ordlista som skall täcka det behov som kan uppkomma efter förklaring av den terminologi som förekommer i uppsatsen.

5.4 Fortsatt forskning

Nedan presenterar jag kort tre stycken förslag på fortsatt forskning som jag tycker kan tjäna som goda förlängningar av den här uppsatsen innehåll.

1) Forskning som är lik den här i sin utformning men där det finns mer tid att lägga på den empiriska delen. Detta så att man därmed kan bedöma den delen bättre på grund av att man får mer tid att sätta sig in i hur plattformarna fungerar. Vidare också kanske än mer hur man praktiskt exempelvis hanterar Web Services. Dessutom få mer tid att göra testerna mer utförliga och på så sätt kanske bland annat erhålla en god reliabilitet vilket jag inte lyckades med på grund av just tidsbrist så att jag inte hann utprova andra mätprogram än JMeter. Att

därmed alltså vid ett sådant test pröva flera stycken program av samma karaktär som JMeter för att på så sätt kunna få fram ett mer optimalt mätinstrument.

2) Forskning kring de open source alternativ som finns främst gällande Web Services och kanske även mjukvaruplattformar i stort. Jag har här i den här uppsatsen tagit upp till exempel en open source implementationen av .NET vid namn Mono. Jag skulle anta att det inte är det enda som finns, och även om man bara kommer upp med Mono så kunde dess funktionalitet utforskas mer ingående. Jag har ju valt J2EE och .NET på grund av att de är de mest tillgängliga och mest förekommande och på så sätt lättast för min målgrupp att skaffa och ta till sig. Men just därför känner jag att behovet av att utforska och lägga fram det som inte är så 'vanligt' i det allmännas ljus är desto större.

3) Någon form av forskning där användningsområdena av Web Services studeras på ett mer djupare och ingående plan än vad som har gjorts i den här uppsatsen. Visst har jag tagit upp en rad användningsområden men ingen djupare analys har gjorts eftersom det har legat utanför de områden som den här uppsatsens har behandlat. I synnerhet tycker jag det skulle vara mycket intressant om det i den forskningen lades extra fokus på nya spännande användningsområden, gärna från saker som än så länge bara är på ett experimentellt stadium. Det som är nytt och innovativt och som sagt var gärna ovanligt tror jag kan tjäna som en nyttig inspirationskälla för andra. De kan kanske sedan bidraga med sina idéer till de forskningsprojekt som det kan tänkas det skrivs om eller till och med utveckla dessa i egna riktningar.

6. Ordlista

Här nedan har jag listat termer som förekommer i uppsatsen. Vissa av dessa termer kan exempelvis redan vara förklarade i uppsatsen men då på ett sätt som jag tyckte krävde ett ytterligare separat förtydligande. Ibland förekommer en förkortning före dess utskrivna form och ibland det motsatta förhållandet. Detta beror på att termerna förekommer i nedanstående lista i samma form som de förekommer i uppsatsen, detta för att undvika förvirring.

.NET-identiteten
.NET-ramverket inkluderar en enkel men mycket flexibel identitetsbaserad säkerhetsmekanism. Genom att använda den kan man ha mycket finkänslig kontroll över vilka som har tillåtelse att använda program och vilka funktioner som de användarna får bruka (Sabbadin, 2003).

abstraktion
Abstraktion är en förenklad beskrivning, eller specifikation av ett system som lägger tonvikt vid någon eller några av ett systems detaljer medan det döljer andra. En bra abstraktion är en sådan som lägger vikt vid detaljer som är viktiga för användaren och ställer sådana som är oväsentliga eller tveksamma i bakgrunden (Shaw refererad från ett citat i Berard).

ActiveX
En löst definierad uppsättning av teknologier och utvecklad av Microsoft. ActiveX är en vidareutveckling av två andra Microsoft-teknologier: OLE och COM (se även dessa begrepp). AcriveX kan vara väldigt förvirrande på grund av att det syftar till en hel uppsättning av COM-baserade teknologier. De flesta tänker dock på det som AciveX-kontroller vilket är ett särskilt sätt att implementera ActiveX-teknologier (ActiveX).

ADO.NET dataläsare (databaser)
ADO.NET skapar en dataström som bara kan läsas och då endast framåt det vill säga man kan inte backa och gå tillbaka i strömmen. Exempelvis så säg att den returnerar tre poster då kan man bara gå igenom posterna framlänges och kommer du på vid post tre att du vill läsa post ett igen så går det därmed inte. Finessen med det här är att det går snabbare samt det går att optimera minnet som man redan har gått över, den datan behövs inte längre (ADO.NET dataläsare).

API (Application Program Interface)
En uppsättning av rutiner, protokoll och verktyg för konstruktion av mjukvaruprogram. Ett bra API underlättar för utvecklaren att skapa program genom att förse denne med lämpliga ”byggnadsblock”. Programmeraren sätter sedan samman dessa och kan på så sätt bilda program.

Flertalet operativsystem Innhåller API:s. Dessa gör det lättare att skapa program i det aktuella operativsystemets speciella miljö. Även om API:s i första hand är avsedda för programmerare så innebär de också en fördel för användare pga att alla program som använder ett generellt API kommer att ha liknande gränssnitt och därmed lättare att som användare känna igen sig (API).

Applet
En applet är ett program som är skapat för att endast kunna köras inom ett annat program. Olikt ett reguljärt program så kan inte applets köras direkt från operativsystemet. Användningsområdena för applets är många.

Webbläsare vilka ofta är utrustade med så kallad JVM:s (se detta begrepp) kan interpretera (se detta begrep) applet:s just inifrån webbläsaren. På grund av att applet:s har en liten filstorlek, är plattformsoberoende, och mycket säkra (går ej att använda för att nå access till användares hårddiskar), så är de idealiska för små Internet-program som kan nås med hjälp av en webbläsare (Applet).

Array
Inom programmering så betecknar en array en serie av objekt (se även detta begrepp) vilka alla är av samma storlek och typ. Varje objekt i en array kalas för ett array-element. Det viktiga för att känna igen en array är:

- Varje element har samma datatyp (fast de kan ha olika värden)
- Hela array:en är lagrad i en samlad följd i minnet, dvs det finns inga luckor mellan de olika elementen.

(Array)

ASP (Active Server Pages)
En specifikation för en dynamiskt genererad webbsida med en .ASP-extension som använder sig av ActiveX (se även detta begrepp) skript vanligtvis skrivna med Visual Basic- eller Jscript kod. När en webbläsare anropar en ASP-sida, så genererar webbservern en sida med HTML (se även detta begrepp) kod och sänder tillbaka det till webbläsaren (Active Server Pages).

Asynkron dataöverföring
Dataöverföring där varje tecken i överföringen förses med start och stoppbitar. Där förekommer inte heller några synkroniserade klocksignaler, vilket innebär att datan som kan vara i form av paket kan komma lite hur som helst (Asynkron; Thorell, 1995).

B2B (business-to-business)
B2B innebär utbytet av tjänster, information och/eller produkter från en affärsidkare till en annan (B2B).

B2C (business-to-consumer)
B2C innebär tjänster, information och/eller produkter från en affärsidkare till en konsument (B2C).

Back-end
Front-end och back-end är begrepp som karaktäriserar gränssnitt (se även detta begrepp) och tjänster relativt till användaren av dessa (’användaren’ kan vara en människa eller ett program). Ett front-end program är ett som användarna kommunicerar direkt med. Ett back-end program fungerar indirekt som ett stöd för front-end tjänsterna, vanligtvis genom att vara närmare den begärda källan eller genom att ha förmågan att kunna kommunicera med den eftersökta källan. Back-end applikationen kan interagera direkt med front-end applikationen eller kanske mer vanligt att det är ett program som anropas av ett mellanliggande program som förmedlar mellan front-end och back-end-programmen (Back-end).

Bindning
Bindning kallas det när man tilldelar ett värde till en symbolisk så kallad ’placeholder’(reserverar adresser i en dators minne). Under kompilering (omvandling från skriven kod till maskinkod) till exempel, så tilldelar kompileraren en symbolisk adress till vissa variabler och instruktioner. När programmet är bundet, så ersätt den symboliska representationen med en riktig så kallad maskinadress (fast minnesadress i en dator) (Bindning).

En annan och något enklare förklaring som är mer direkt knutet till tjänster och XML är: En bindning förenar en abstrakt definition av en tjänst med ett specifikt XML-baserat protokoll samt transport. Ett exempel av bindning är protokollet SOAP över, eller kanske rättare sagt användandes av, HTTP (Hanson, 2002).

Cache
Cache är en speciell mekanism för höghastighets-lagring av data. Den kan antingen bestå av en reserverad minnessektion eller av ett oberoende lagringsmedia. Två varianter av caching som är vanliga i persondatorer är: minnes-caching och disk caching.

Minnes-cache är en del av ett minne bestående av den snabba minnestypen Static Random Access Memory (SRAM) istället för det snabbare och billigare minnes typen dynamic RAM som används för det huvudsakliga arbetsminnet i en dator. Minnes-caching är effektivt på grund av att de flesta program använder samma data eller instruktioner om och om igen. Genom att ha så mycket av den frekvent använda datan i det snabbare SRAM-minnet så snabbas hanteringen upp avsevärt än om man endast skulle hämta datan från det långsammare DRAM-minnet.

Disk-caching fungerar enligt samma princip som minnes-caching, men istället för att använda SRAM som ovan används det konventionella minnet i datorn. Den senast hämtade datan från disken (vanligtvis hårddisken) lagras i en minnesbuffert. När ett program sedan behöver hämta data från disken, så kollar det först om den eftersökta datan finns i denna minnesbuffer, cachen. Denna typ av caching kan öka prestandan avsevärt hos applikationer, på grund av att hämta en byte med data från arbetsminnet går avsevärt mycket snabbare än om man skulle ha gjort motsvarande från exempelvis en hårddisk (Cache).

Class loading
Inom objektorienterade utvecklingsspråk som exempelvis Java så är en så kallad ‘Class loader’ ett objekt som är ansvarigt för att ladda konstruktorer (se även detta begrepp) dynamiskt för olika objekt (se även detta begrepp). Enklare uttryckt kan man säga att en ’Class loader’ är ansvarig för att ladda klasser (se även detta begrepp). Givet ett namn för en klass så kan den försöka att lokalisera eller generera den data som utgör definitionen för en klass. Till exempel så kan en applikation skapa en nätverks- ’Class loader’ för att ladda ner ’klassfiler’ från en server (Class loading).

CMP2 (Container Managed Persistance, version 2)
Entitetsbönor (se motsvarande avsnitt i uppsats) delegerar ansvaret för så kallad ’ persisting data’ , data som finns kvar i minnet så kallad minnesresident data, till den container (se även detta begrepp) som har ansvar för bönan och därav begreppet CMP (CMP2 (Container Managed Persistance, v2)).

Cobalt (hårdvaru-servers)
Cobalt är hårdvaruservers som tillverkas numera av företaget Sun. Sun köpte år 2000 upp Cobalt, en taktiskt manöver på grund av att deras egna billigare servers inte sålde tillräckligt mycket gällande marknaden för lagring och hantering av Webbsidor (Webbserver och dylikt). Cobaltmaskinerna använder sig av linuxoperativsystemet, AMD eller numer alltmer Intel processorer. Sun:s egna servrar använder operativsystemet Solaris och UltraSparc-processorer (Shankland).

COM (Component Object Model)
En mjukvaruarkitektur som har utvecklats av Microsoft för konstruktion av komponentbaserade applikationer. COM-objekt är diskreta komponenter, där varje komponent har en unik identitet, vilka visar upp gränssnitt som gör det möjligt för program och andra komponenter att nå dess funktioner. COM-objekt är mer mångsidiga än de så kallade Win32 DLL (se även detta begrepp) med tanke på att de är totalt språkoberoende, har en inbyggd process för internkommunikation samt att de passar lätt in i en objektorienterad programdesign (COM).

Common Language Infrastructure (CLI)
Skapad av Microsoft som en grundläggande del i deras .NET-plattform. CLI är en så kallad ECMA-standard (ECMA-355) som tillåter att applikationer kan skrivas i en rad högnivåspråk och dessutom exekveras i en mängd olika miljöer.

Program som tillkännager sig till CLI kan därmed också bli kompilerade till samma mellanliggande språk (’intermediate Language’, IL, se även detta begrepp) samt metadata (se även detta begrepp). IL kompileras sedan ytterligare till det språk som vid tillfället används och anpassas därmed också till det språkets arkitektur (CLI).

Common Language Runtime (CLR)
En så kallar ‘runtime environment’ (ungefär exekveringsmiljö) som tillhandahar exekveringen (se även detta begrepp) av programkod inom Microsofts plattform .NET. Miljön erbjuder även tjänster som minnes- och så kallad undantagshantering, ’debugging’ (söka samt rätta till fel) samt säkerhet. CLR är en viktig del i Microsofts så kallade .NET-framework (CLR).

Community (ungefär samhälle)
En virtuell samlingsplats på Internet för människor som har ett gemensamt intresse (exempelvis ett professionellt, socialt eller demokratiskt sådant) för att engagera sig i en tvåvägskommunikation med ’samhällets’ andra medlemmar för att dela idéer, kunskap, information och åsikter. Ett ’samhälle’ består även vanligtvis av ett eller flera forum (Community).

Container
I Sun:s komponentarkitektur samt i Microsoft:s COM (se även detta begrepp), är en container ett applikationsprogram eller subsystem i vilket det program-byggnadsblock som är känt som komponent exekveras (se även detta begrepp). Till exempel, en komponent, så som en knapp eller annat grafiskt gränssnitt (se även detta begrepp) eller en liten mjukvaru-fickräknare, kan utvecklas genom att använda så kallade Java bönor som kan köras i Netscape containers som browsers eller om man använder Microsoft så kan man köra komponenten i dess containrar såsom Internet Explorer, Visual Basic och Word (Container).

CORBA (Common Object Request Broker)
CORBA är en generell öppen industristandard för distribuerade objekt. Med CORBA kan man koppla ihop objekt eller applikationer oavsett programspråk, plattform, hårdvara eller geografisk plats (koppling sker via Internet) (Balic, Björklund, & Jägmark, 2001).

DCOM (Distributed Component Object Model)
DCOM är Microsofts distribuerade version av COM (Common Object Model) komponenter. Denna teknik möjliggör client/server-lösningar som kan koppla ihop flera klienter mot en server. Det är en möjlighet att utveckla en DCOM-server i flera olika språk så länge att formen stödjer COM tjänster. DCOM används främst på Windows-plattform men går att använda i UNIX-, Linux- och stordatormiljö om platt formen stödjer COM-tjänster (Balic, Björklund, & Jägmark, 2001).

DHTML (Dynamic HTML)
1) Refererar till ett webbinnehåll som förändras varje gång det beskådas. Till exempel så kan samma URL (se även detta begrepp) resultera i att olika sidor visas beroende på en rad parametrar, såsom:
    - Läsarens geografiska lokation
    - Tid på dagen
    - Vilka sidor som läsaren har tidigare tittat på
    - Läsarens profil

Några av de vanligaste teknologierna för att producera dynamisk HTML är: CGI skript, Server Side Includes (SSI), cookies, Java skript och ActiveX (se även detta begrepp).

2) När termen skrivs med stora bokstäver, så brukar termen referera till en ny typ av HTML-extensioner som gör det möjligt för en webbsida att reagera på användarinmatningar utan att sända dessa inmatningar till Webservern i fråga (DHTML).

Discovery and lookup-protokoll
Vid webservices och SOAP så använder man UDDI för att berätta var servicen finns och hur man gör för att ’använda’den. UDDI är sålunda ett exempel på ett ’Discovery and lookup-protokoll’ (Discovery and lookup-protokoll).

Distribuerade Datamodeller
En typ av datahantering i vilket olika komponenter och objekt kan innefattas inom en applikation. Denna applikation kan sedan finnas på olika datorer som i sin tur är sammanlänkade av ett nätverk. Så, exempelvis så kan en ordbehandlingsapplikation berså av en komponent för redigering på en dator, en komponent för stavningskontroll på en annan och så vidare. På vissa distribuerade system så är det även möjligt att någon av datorerna använder sig av olika operativsystem.

En nödvändighet för distribuerade system är att man använder sig av en uppsättning av standarder som bestämmer hur man skall kommunicera med varandra. Två sådana standarder är: CORBA och DCOM (se även dessa begrepp) (Distribuerade Datamodeller).

Vid utveckling av så kallad Middleware så är idag de vanligaste distribuerade datamodellerna: DCOM, CORBA och RMI (Balic, Björklund & Jägmark, 2001).

DLL (Dynamic Link Library)
Ett bibliotek med exekverbara funktioner eller data som kan användas av Windows-applikationer. Generellt sett så erbjuder en DLL en eller flera specifika funktioner som ett program kan nå genom att skapa en statisk eller dynamisk länk till DLL:n. En statisk länk ändras inte när program körs medan en dynamisk länk skapas av det program som behöver DLL:n. DLL:er kan även innehålla bara data.

Flera applikationer kan simultant använda samma DLL. Vissa DLL medföljer Windows operativsystem och är därmed tillgängliga för alla Windows program. Andra DLL:er är skrivna för ett särskilt program och installeras med det programmet (DLL).

DTD (Document Type Definition)
En fil som definierar elementen och datastrukturen som finns i ett XML-dokument (DTD).

Dynamic output caching (med ASP.NET)
Hellre än att återskapa dynamiska webbsidor vid varje förfrågan av desamma, så kan ASP.NET använda sig av cache (se även detta begrepp) funktionalitet som gör medför att hela sidor eller delar därav kan 'cachas'. Detta medför en avsevärd ökning av skalbarheten genom att användningen av cache reducerar belastningen på servern. Microsoft kallar den här typen av 'cachning' för 'dynamic output caching'.

För att även kunna 'cacha' dynamiska webbsidor vars innehåll inte är statisk utan ändras mer eller mindre ofta, som exempelvis en söksida, så har ASP.NET löst det genom att variera de 'cachade' utgående datan med hjälp av en uppsättning av parametrar så som frågesträngen. Om parameterns värde förändras så lägger ASP.NET till en ny kopia av den utgående datan (Dynamic output caching).

EAI (enterprise application integration) funktionalitet
EAI kallas processen för obegränsat utbyte av data eller affärsprocesser inom ett nätverk eller datakällor hos en organisation (EAI).

ebXML (electronic business Extensible Markup Language)
En modulär (se även detta begrepp) uppsättning av specifikationer för att standardisera XML globalt och därmed underlätta för handel mellan organisationer oavsett deras storlek. Specifikationen erhåller företagen med en standardiserad metod för att utbyta XML-baserade affärsmeddelanden, sköta handelsrelationer, generell datakommunikation samt definiera och registrera affärsprocesser (ebXML).

Enterprise
Inom databranschen är det ett begrepp som används för att beskriva en stor organisation som nyttjar datorer. Ett intranet till exempel, är att bra exempel på ett så kallat ’enterprise computing system’ (Enterprise).

Entitet (Entity)
En entitet är ett namn avsett för en del av en mängd data så att denna kan refereras till med ett namn. Datan kan vara vad som helst från simpla bokstäver till uppsättningar av DTD- (se

även detta begrepp) uttryck. Entitetsparametrar kan exempelvis vara generella, externa eller intern data. En entitet kan även liknas vid en variabel i ett programmeringsspråk (Entitet).

Exekvera
Med exekvering innebär det att man utför en handling, så som att köra ett program eller skriva ett kommando (Exekvera).

Fail-over
En så kallad Application-Server middleware (se även detta begrepp) - tjänst. Fail-over används för att förbättra feltoleransen mellan klienter och applikationsservern. Om en server inte fungerar kopplas klientens anrop till en annan fungerande server. (Benfield 1999)

Vissa applikationsservrar har möjlighet att återskapa data vid ett serverhaveri med hjälp av ’session-level-fail-over’. Detta innebär att tillstånd- och sessionsdata kopieras till en andra server eller placeras i databas. På så sätt blir användarens tillstånd tillgängliga på alla servrar (Benfield 1999 refererad i Balic, Björklund, & Jägmark, 2001).

Förenade identiteter
Identitet är en mycket viktig webbtjänst. Det är kärnan i företagens möjligheter att känna sina kunder och partner, och nå dem på sätt som är av värde för alla parter.

Förenade identiteter innebär vidare att exempelvis många företag använder standardiserade sätt att logga in på webben med enkla förenade konton som flera tillverkare stödjer. Alla dessa företag använder vanliga identitetslösningar. Denna förening kan vidare även innebära samverkan mellan system så att länkning av konton utan användarens medgivande och förenklad inloggning kan användas. De utgör den första uppsättningen specifikationer för öppna, förenade nätverksidentiteter (Förenade identiteter, 2002).

Gnome-projektet (GNU Network Object Model Environment)
GNOME är en del av det så kallade GNU-projektet och även en del av den fria mjukvaru- även kallat ’open source’-rörelsen. GNOME i sig är ett Windows-liknande så kallat ’desktop’ (skrivbords-) -system som opererar under UNIX samt UNIX-liknande system. Det är inte beroende av någon så kallad ’windows-manager’ (mjukvara som har hand om fönstersystem) (GNOME).

Gränssnitt (interface)
Kan beskrivas som en brygga som sammanbinder två oberoende system, så att de kan mötas och kommunicera med varandra. Inom datorteknologin så finns det flera olika sorters gränssnitt

- Användargränssnitt – som kan vara tangentbordet, musen eller menyerna hos ett datorsystem. Gränssnittet tillåter användaren att kommunicera med operativsystemet.
- Mjukvarugränssnitt – språken och koderna som applikationer använder för att kommunicera med varandra och med hårdvaran.
- Hårdvarugränssnitt – kablarna, kontakterna och socklar som hårdvara använder för att kommunicera med varandra (Gränssnitt).

GSS-API (Generic Security Service Application Programmers Interface)
Ett API (se även detta begrepp) som erbjuder generella säkerhetstjänster till dem som anropar. Kan stödjas med en mängd underliggande mekanismer och teknologier samt följaktligen så tillåts portabilitet på källkodsnivå gällande applikationer i olika miljöer (GSS-API).

GUI (Graphical User Interface)
Ett gränssnitt (se även detta begrepp) hos program som drar fördel av en dators grafiska möjligheter och på så sätt underlätta hanteringen av programmet i fråga för användaren (GUI).

HTML (HyperText Markup Language)
Ett så kallat markeringsspråk som används för att skapa dokument på World Wide Web. HTML definierar strukturen och layouten hos ett webbdokument genom att använda olika så kallade taggar (kommandon som specificerar formatering av dokument) och attribut (egenskaper). Taggarna kan även användas för att specificera så kallade hypertextlänkar (URL:er, se även detta begrepp). Dessa tillåter Webbutvecklare att leda användare vidare till andra webbsidor bara genom att klicka på ett ord eller en bild (HTML).

HTTP (HyperText Transfer Protocol)
Det underliggande protokollet som använda av World Wide Webb. HTTP definierar hur meddelanden är formaterade och sända, och vilka åtgärder som webservrar och webbläsare skall utföra som svar på olika kommandon. Till exempel, när du matar in en URL (se även detta begrepp) på din webbläsare, så sänder det egentligen iväg ett http-kommando till den webbserver dit adressen går och som svar sänder denna server tillbaka den begärda webbsidan (HTTP).

HTTPS (Secure HyperText Transfer Protocol Secure)
En säker version av HTTP (se även detta begrepp). Webbsidor som använder sig av en SSL- (se även detta begrepp) anslutning inleder sin URL (se även detta begrepp) med https istället för http. HTTPS är till skillnader från SSL konstruerat för att endast sända individuella säkra meddelanden medan SSL sätter upp en komplett säker anslutning mellan klient och server (SSL).

Identitet (Identity)
Inom datavärlden kännetecknar en identitet ett unikt namn på en person, eller ett tillbehör eller dylikt eller en kombination av de båda vilka båda är igenkännbara av ett system (Identitet).

Implementation
Sätta i drift, tillämpa, införa system (Implementation).

Inkonsistent data
Data som är osammanhängande eller ofullständig (Kroenke, 2000, s. 395-397).

Internal Language (IL)
Program som tillkännager sig till det av Microsoft skapade ICL (se även detta begrepp) kan också bli kompilerat till Microsoft:s variant av ett så kallat mellanliggande språk (’intermediate Language’, IL) samt metadata (se även detta begrepp). IL kompileras sedan ytterligare till det språk som vid tillfället används och anpassas därmed också till det språkets arkitektur (CLI).

Internet-ORB Protocol (IIOP)
Ett protokoll som har utvecklats av Object Management Group (OMG) för att implementera CORBA (se även detta begrepp) -lösningar via World Wide Web. IIOP gör det möjligt för webbläsare och servrar att utbyta integers, array:er (se även detta begrepp), och även mer komplexa objekt, till skillnad från HTTP (se även detta begrepp) som bara tillåter överföring av text (IIOP).

Interpreterare
Ett program som exekverar (se även detta begrepp) skrivna i ett högnivåspråk. Det finns två sätt att exekvera ett högnivåspråk. Det mest vanliga är att kompilera ett program, den andra metoden är att låta programmet i fråga passera genom en interpreterare. En interpreterare översätter vid exekvering ett högnivåspråk till en så kallad mellanliggande form, vilket den sedan exekverar. I kontrast till en kompilerare så översätter inte en kompilerare direkt till maskinkod. Därför blir interpreterade program generellt långsammare än kompilerade. Fördelen med kompilerade program är bland annat att de för en utvecklare kan vara lätta att testa, när själva källkoden i dess ursprungliga form finns tillgänglig (Interpreterare).

iPlanet
Iplanet-plattformen är en mjukvarumiljö som är skapad för att göra underlätta hanteringen av snabb ansamling och tillhandahavande av skalbara Internettjänster. Baserade på utprovade och avancerade teknologier så inkluderar plattformen bland annat Meddelande-, Fil-, Webb- samt applikationsservrar (iPlanet).

ISAPI (Internet Server API)
Ett API (se även detta begrepp) för Microsoft:s IIS (Internet Information Server) Webb server. ISAPI erbjuder programmerare att utveckla webb-baserade program som kan köras mycket snabbare än konventionella CGI-program på grund av att de är i mycket högra grad integrerade med Webbservern. Servrar av flertalet andra märken än Microsoft stödjer också ISAPI (ISAPI).

ISO (International Organization for Standardization) – standarden
Notera att ISO inte är en förkortning, istället kommer begreppet från det grekiska ordet iso, vilket betyder motsvarande. Grundat 1946 så är ISO en internationell organisation bestående av nationella standardiseringsorgan från över sjuttiofem länder (ISO).

Java Connector Architecture (JCA)
J2EE-arkitekturen erbjuder en Java-teknologi för lösningen beträffande problemet att ansluta många applikationsservrar med nuvarande affärsinriktade informationssystem (Enterprise Information Systems, EIS) (JCA).

Java Database Connectivity (JDBC)
Ett java API (se även detta begrepp) som möjliggör för Java-program att exekvera SQL (se även detta begrepp) -strängar. Detta tillåter Java-program att interagera med alla databaser som stödjer SQL (JDBC).

Java Runtime Environment (JRE)
Java är ett objektorienterat språk, liknande C++, men förenklat för att eliminera språkegenskaper som orsakar vanliga programmeringsfel. Javas källkodsfiler är kompilerade till ett format som kallas ’bytecode’, vilket sedan kan bli exekverat av en Java interpreterare. Kompilerad Javakod kan köras på de flesta datorer på grund av att Java interpreterare och exekveringsmiljöer (JRE:s), kända som ’Java Virtuell Machines’ (JVM, så även detta begrepp), finns för de flesta operativsystem, inkluderandes Unix, Macintosh OS samt Windows. Bytekod kan även bli konverterad direkt till maskinspråks-instruktioner med en ’just-in-time’-kompilerare (JIT, se även detta begrepp) (JRE).

Java VM (JVM, Java Virtual Machine)
Programvara som måste finnas för att till exempel webbläsare skall kunna använda Java-program. Den virtuella dator som exekverar (se även detta begrepp) javakoden. Sun står för specifikationerna över hur en JVM ska fungera men det är fritt fram att göra sin egen vilket både Netscape och Microsoft gjort (JVM). Läs även under begreppet ’JRE’.

jBoss
En applikationsserver skriven i Java som kan agera värd för affärskomponenter utvecklade i Java. I grund och botten är jBoss en open source (se även detta begrepp) –implementation (se även detta begrepp) av J2EE och tillförlitar sig på Enterpris Java Bean-specifikationen för dess funktionalitet (jBoss).

JNI (Java Native Interface)
Ett gränssnitt (se även detta begrepp) för programmering, eller API (se även detta begrepp) om man så vill, som ger utvecklare tillträde till ett värdsystems språk för att på så sätt bestämma hur Java skall integrera med den inhemska koden på det systemet (JNI).

JSP (Java Server Page)
En teknologi som huserar på Serversidan och som även är en utveckling av Java Servlet (se även detta begrepp)-teknologin som även den är utvecklad av Sun.

JSP har dynamiska skriptmöjligheter som fungerar tillsammans med HTML- (se även detta begrepp) kod, separerandes en webbsidans logik (’tankesättet’ bakom sidan) från dess statiska element, det vill säga sidan faktiska design och hur den visas upp, för att på så sätt göra HTML-koden mer funktionell som exempelvis kunna hantera dynamisk databasfrågor.

JSP anpassas till Java Servlet innan den exekveras och därmed hanterar HTTP -(se även detta begrepp) anrop samt genererar svar likt vilken Servlet som helst. Men, JSP-teknologin erbjuder ett mer bekvämt sätt att koda en Servlet. Översättningen äger rum första gången som applikationen exekveras. En JSP översättare sätter igång processen när den märker att filextensionen är ’.jsp’ i en URL (se även detta begrepp). JSP har en fullständigt samverkansförmåga med Servlets. Man kan inkludera utdata från en Servlet och skicka vidare utdatan till en Servlet, och det går även att inkludera utdatan från en JSP eller skicka vidare utdatan till en JSP.

JSP:er är inte begränsade till någon specifik plattform eller server. Det var ursprungligen konstruerat som ett alternativt till Microsoft:s ASP (se även detta begrepp) (JSP).

Just In Time (JIT) kompilerare
En kodgenerator som konverterar javas bytekod till maskinkods-instruktioner. Java som kompileras med en JIT är oftast mycket snabbare än när koden är exekverad med en interpreterare (se även detta begrepp) (JIT).

Kerberos
Kerberos är ett autenticeringsprotokoll för nätverk. Det är konstruerat för att erbjuda stark autenticering för klient/server-applikationer genom att använda så kallad 'secret key' – kryptografi (Kerberos).

klass
1) Inom objektorienterad programmering så innebär en klass en viss kategori av objekt. Till exempel kan det vara en klass som kallas form där det ingår objekt vilka är cirklar, rektanglar, och trianglar. Klassen bestämmer alla vanliga egenskaper hos de olika objekten som tillhör klassen.
2) Gällande språk inom Microsofts plattform .NET, så är klasser mallar för att definiera nya typer. Klasser beskriver både egenskaperna och beteenden hos objekten (Klass).

kodbas
Generellt så är en kodbas en fristående CVS- (Concurrent Versions System, ett system för att hantera konfigurering av mjukvara) modul för ett specifikt paket.

Kodbaser specificerar de mobila kod-URL:erna som en komponent behöver för att kunna exekveras. Länkarna måste vara av sorten HTTP (Kodbas).

Komponent
En komponent är en standardiserad byggsten i en organisation som används för att utveckla applikationer. För att en komponent skall vara återanvändbar så måste den vara oberoende av den applikation som den byggs för. Det går att återanvända komponenter som är beroende av en applikation, men ju mer beroende desto mer måste komponenten anpassas till den applikationen som man vill återhämta den på. Exempel på komponenter som måste anpassas är fönster, filhanterare m.fl. (Jabocsen et al 1996. s 294-295 refererad i Johansson & Ledin, 2001)

Konstruktor
I objektorienterade utvecklingsspråk som exempelvis Java så anropas automatiskt vid skapande av objekt (se även detta begrepp) en så kallad konstruktor för det aktuella objektet. Konstruktorn är en slags metod med vilken man gör de initieringar, det vill säga ungefärligen 'grundinställningar', som kan behövas för det nya objekt som har skapats (Skansholm, 1999, s. 21).

Load Balancing
En så kallad Application-Server middleware -tjänst. Med load balancing menas att en samling applikationsservrar finns tillgängliga och anropas via en dispatcher (komponent som styr datatrafiken). Anropet från klienten dirigeras vidare tiol den minst upptagna servern och klienten kan kommunicera med den applikationsservern som den har blivit hänvisad till av dispatchen (Benfield 1999 refererad i Balic, Björklund, & Jägmark, 2001).

Logisk vy
Gällande mjukvaruarkitekturer så görs det en skillnad mellan olika nivåer av abstraktion (se även detta begrepp), eller synvinklar om man så vill, från vilket en beskrivning av ett system är möjligt. En av dessa synvinklar gällande beskrivning av en mjukvaruarkitektur är den logiska vyn, vilken beskriver de funktionella behoven hos systemet i fråga (Logisk vy).

Marshalling (data marshalling)
Marshalling kallas processen som innefattar att samla ihop data och omvandla det till ett standardiserat format före det sänds via ett nätverk så att datan kan överskrida nätverkets begränsningar. För att ett objekt (se även detta begrepp) skall kunna flyttas runt i ett nätverk, så måste det först konverteras till en dataström som motsvarar paketstrukturen i nätverkets överföringsprotokoll. Denna omvandling kallas alltså som sagt 'data marshalling'. Olika delar av datan samlas ihop i en meddelande-buffer före de är 'marshaled'. När datan överförs, så omvandlar den mottagande datorn den datan som genomgått processen tillbaka till ett objekt igen. Denna process är nödvändig när man använder sig av utgående dataparametrar från ett program skrivet i ett språk som ingående dataparametrar till ett program skrivet i ett annat språk (Marshalling). Se även begreppet 'Un-Marshalling'.

Metadata
Metadata kan beskriva som data om data. Metadata beskriver hur, när och av vem en specifik ansamling av data har blivit hanterad, och även hur datan är formaterad. Metadata är har blivit en oerhört viktig del i XML-baserade applikationer (Metadata).

Microsoft Host Integration Server 2000
Microsoft Host Integration Server 2000 utökar Microsoft Windows till att integrera med andra system genom att erbjuda applikations-, data- och nätverksintegrering. Host Integration Server öppnar upp för nya affärsmöjligheter samtidigt som det bevarar den befintliga infrastrukturen i företaget (Microsoft Host Integration Server 2000).

Microsoft Message Queue (MSMQ)
Microsoft Message Queuing-teknologin låter applikationer köras vid olika tillfällen för kommunikation över heterogena nätverk och för system som för tillfället ej är uppkopplade på nätet. Programmen sänder och läser meddelanden till/från köer (MSMQ).

Microsoft Transaction Server (MTS)
MTS är ett komponentbaserat system som behandlar transaktioner. Systemets syfte är att bygga, verkställa och administrera serverapplikationer avsedda för Internet och Intranet (Combs).

Modeller
Med modell avses en avbildning av verkligheten. En avbildning som också är en förenkling av det man avbildar, för att på ett abstrakt sätt kunna visa hur något ser ut (Balic, Björklund, & Jägmark, 2001).

Modulär
Refererar till designen hos ett system som består av separata komponenter som kan kopplas samman. Finessen med en modulär arkitektur är att du kan byta ut vilken som helst av komponenterna utan att det påverkar resten av systemet. Termen modulär kan appliceras både på hårdvara likväl som på mjukvara (Modulär).

MSXML (Microsoft Extended Markup Language)
MSXML är en uppsättning tjänster som erbjuder funktioner för att hantering av XML. En av de viktiga användningsområdena för MSXML är att parsing (se även detta begrepp), generering, validering samt att konvertera XML-dokument så att dess information kan visas, lagras eller manipuleras (MSXML).

Namnrymd
Namnrymd har många betydelser och användningsområden, här följer några av dessa:

1) En logisk gruppering av namn som används inom ett program. Även kallat ’name scope’.
2) Ett bibliotek med klasser i .NET.
3) XML namnrymd: I XML, så är en namnrymd en samling av namn, identifierade av en URL (se även detta begrepp)-referens, som används i XML-dokument som element-typer och attributnamn. För att XML-dokument skall kunna använda element och attribut som har samma namn men som kommer från olika källor, så måste det finnas ett sätt att särskilja mellan ’markup’-elementen (taggarna) som kommer från olika källor (Namnrymd).

Objekt
Generellt så är ett objekt vilket föremål som helst som kan individuellt väljas och manipuleras. Det kan inkludera former, bilder som förekommer på en bildskärm likväl som mindre konkreta mjukvaru-entiteter (Objekt).

OLE DB (Object Linking and Embedding DataBase)
OLE är en sammansatt dokumentstandard som är utvecklad av Microsoft. Den tillför möjligheten att skapa objekt (se även detta begrepp) med en applikation och sedan länka eller bädda in objektet i en annan applikation. Inbäddade objekt behåller deras ursprungliga format och är länkade till den applikation som skapade dem. Databashantering som i det här fallet är ett av de användningsområden som denna standard kan appliceras på (OLE DB).

Open source
Med open source menas generellt ett program för vilket källkoden är tillgänglig för allmänheten att fritt och utan kostnad använda eller modifiera. Sådan kod är vanligtvis skapas som ett kollektivt samarbete mellan utvecklare i en så kallad community (se även detta begrepp)där de skapar och förbättrar mjukvaran ifråga. Det är även en motreaktion mot den kommersiella mjukvara som finns. Det anses att bra och felfri mjukvara skall finnas tillgänglig för alla utan kostnad (Open source).

Output caching
Output caching är en server-teknik som tillåter dig att mellanlagra (’cache’, se även detta begrepp) resultatet från ett anrop i minnet gällande säg en webbsida, efterföljande anrop för den sidan tas därmed fram från mellanlagring och kan därmed hanteras mycket snabbare vilket resulterar i bättre prestanda (Output caching)

Parsing
Parsing är handlingen när ett dokument avläses, ’scannas’, och informationen som dokumentet Innhåller filtreras till elementens kontext i vilken informationen är strukturerad. Enklare uttryckt så anpassas det avlästa dokumentets innehåll till den miljö som det skall

förekomma. En annan betydelse är när en kompilerare (se även detta begrepp) analyserar en program-källfil efter syntax-fel (Parsing).

RDBMS (Relational Database Management System)
En form av databashanteringssystem (DBMS) som lagrar data i form av tabeller som är relaterade till varandra. Relationsdatabaser är kraftfulla på grund av att de behöver få förutsättningar gällande hur datan ska lagras eller hur den kommer att extraheras från databasen. Som en följd av det så kan en databas granskas på en mängd olika sätt. En annan fördel är att en databas kan vara spridd över flera olika tabeller (RDBMS)

RMI (Remote Method Invocation)
RMI är en distribuerad objektmodell som är en del av Suns Java Standard. RMI är integrerad med utvecklingsspråket Java. RMI kan bara användas av objekt som är skrivna i Java (Balic, Björklund, & Jägmark, 2001).

Sampler
En kontrollfunktion hos programmet Jmeter (se motsvarande avsnitt i uppsats). Samplern talar om för JMeter att den skall sända förfrågningar till en server. Genom att addera konfigurationselement till samplern kan den än mer formas så som man vill att den skall fungera (Sampler).

Serialisering
Förfarandet att omsätta ett objekts tillstånd till en informationsström kallas för serialisering (Cerami, 2002).

Som ett konkret exempel på serialisering så kan nämnas att i Webbspråket PHP så innebär serialisering av data att man från datan gör en sträng så att det därmed lättare kan lagras på en disk eller i en databas (Serialisering).

Servlet
Ett litet program som körs på en server. Termen refererar vanligtvis till en Java applet (se även detta begrepp) som exekveras inom en webbserver-miljö. Det här är analogt med en Java applet som körs inom en webbläsarmiljö.

Servlets är minneskonstanta, det vill säga att när väl programmet har startats en gång så är det kvar i minnet och kan uppfylla multipla anrop. Denna egenskap gör dem snabba på grund av att ingen tid slösas bort på att starta och avsluta processer (Servlet).

Shared context
Utgör en del av vad som idag (år 2004) är en vision om att kunna dela innehåll/kontext i betydelsen att exempelvis kunna hålla reda på en persons inloggningsinformation, eller dylik information, från ett antal webbsidor, eller andra källor, så att denne person slipper hålla reda på allt det själv vilket idag är något som kan kännas mycket jobbigt. En av de springande frågorna är naturligtvis den centrering av information gällande en person som då kommer uppkomma och den eventuella säkerhetsrisk det också i så fall kommer att innebära (Shared context).

Slutna objekt (sealed objects)
Ett slutet objekt är ett objekt som skyddas när det skickas över nätverk från påverkan. Man kan inte hämta ut innehållet i objektet utan att ha en nyckel som kan öppna objektet (Slutna objekt (sealed objects)).

SOAP Action Parameter
En del av HTTP implementationen av SOAP och förekommer vid SOAP anrop som en del av anropen. Kan användas till att tala om vilket syfte som förfrågan har. Är specifikt för version 1.1 och tidigare av SOAP-protokollet (SOAP action parameter).

SPKM (Simple Public Key Mechanism)
Ett lågnivå protokoll för att kommunicera säkert över öppna nätverk t.ex. Internet (SPKM).

SQL/J (Structured Query Language/Java)
SQL är ett standardiserat så kallat frågespråk som används för att efterfråga information från en databas. En av de egenskaper som har gjort SQL så populär är att de stödjer distribuerade databaser, det vill säga databaser som är spridda över flera datorsystem.

Fastän det finns flera dialekter av SQL, varav SQL/J är Javas dialekt, så är det ändå det närmaste som finns för en standard gällande frågespråk (SQL).

SSL (Secure Socket Layer)
Ett protokoll som är utvecklat av Netscape för att överföra privata dokument via Internet. SSL använder sig av en så kallad privat nyckel för att kryptera data som sänds via SSL-anslutningen. Båda webbläsarna Netscape Navigator och Internet Explorer stödjer SSL, och många webbsidor använder protokollet för att erhålla konfidentiell användarinformation, såsom kreditkortsnummer. En konvention i sammanhanget är att sidor som avsänder sig av en SSL-anslutning inleder sin URL (se även detta begrepp) med https istället för http (se även detta begrepp) (SSL).

State (management)
State betyder status eller läge, och syftar på nuvarande eller den senast kända statusen hos en applikation eller process. Termerna ’maintaining state’ eller ’managing state’ refererar till att hålla ordning på i vilket skick, status/läge, som processen är (State).

Säkra strömmar (secure streams)
Säker (data)ström kallas en dataström då den krypteras för att på så sätt kunna kommuniceras på ett säkert sätt över öppna nätverk som till exempel Internet (Säkra strömmar (secure streams)).

Tablet PC
Tablet PC är en typ av så kallad ‘notebook’-dator som har en LCD-skärm på vilken användaren kan skriva om denna nyttjar en för ändamålet specialkonstruerad penna, även kallad ‘stylus’. Det som skrivs för hand kan digitaliseras och kan även konvertrars till standardiserad text genom en process som kallas för ’handwriting recognition’ eller det handsskrivna kan behålla sin ursprungliga form. Vanligtvis är även denna form av dator utrustad med gränssnitt som tangentbord och/eller mus för inmatning från användaren (Tablet PC).

Tomcat
Tomcat är en server som också är en Javabaserad applikations-container (se även detta begrepp) som skapades för att kunna köra Servlets och Java Server Pages (Se även dessa begrepp) (Tomcat).

Transparent Fail-over
Transparent
Transparent kan liknas vid att vara osynlig. I datorsammanhang så är en handling transparent om den äger rumt utan att någon synlig effekt. Transparens är vanligtvis ansett att vara en bra egenskap hos ett system på grund det avskärmar användaren från systemets egentliga komplexitet.

Fail-over
En back-upp åtgärd som automatiskt växlar till en databas, server eller nätverk som står ständigt redo ifall det primära systemet skulle krascha eller temporärt behöva stängas av för exempelvis service. Failover är en viktig så kallad feltolerans-funktion för kritiska system som är beroende av att kunna erbjuda konstant access till dem. Fail-over omdirigerar automatiskt och transparent (se ovan) användaren från det systemet som inte fungerar längre till back-up diton som härmar det ursprungliga systemets göromål och funktionalitet (Transparent Fail-over).

Un-marshalling
Omvänd process av ’Marshalling’, se detta begrepp.

URI (Uniform Resource Identifier)
Detsamma som en URL (se även detta begrepp). Helt enkelt en webbadress (URI).

URL (Uniform Resource Locator)
URL utgör det globala adress-system som appliceras på dokument och resurser på World Wide Web. Den första delen av adressen visar vilket protokoll som används, den andra specificerar den så kallade IP-adressen eller det domännamn som anger vart den eftersökta resursen finns (URL).

Utvecklingsspråk
Det finns ett antal olika utvecklingsspråk med en rad olika egenskaper. De vanligast förekommande språken vid utveckling är bland annat C++, Delphi, Java och Visual Basic. Val av utvecklingsspråk styrs av bland annat kraven på applikationens prestanda, plattform som applikationen ska implementeras på eller kraven på utvecklingstakt av applikationen. (Balic, Björklund, & Jägmark, 2001).

Visual Studio.NET
Visual Studio.NET är ett programpaket som används för att utveckla webbapplikationer och datadrivna lösningar under Microsofts .NET-plattform (Visual Studio.NET).

WML (Wireless Markup Language)
WML är ett så kallat markeringsspråk som är specifikt utvecklat för trådlösa applikationer. WML är baserat på XML. WML är språket som gör det möjligt att visa webbsidor på handdatorer och mobiltelefoner. WML är den del av det så kallade ’Wireless Access Protocol’ (WAP). Det är en kusin till HTML (se även detta begrepp) (WML).

Wrapper
Wrapper är en mjukvara som kompletterar resurser eller annan mjukvara i syfte att förbättra bekvämlighet, kompatibilitet eller säkerhet. Exempelvis kan en wrapper användas för att komprimera och kryptera mjukvara som säljs över Internet (Wrapper).

XHTML (Extensible Hypertext Markup Language)
XHTML är ett så kallat markeringsspråk. Det är en hybrid mellan HTML och XML (se även de begreppen) och specifikt utvecklat för att kunna visa upp webbsidor på olika hårdvarors bildskärmar eller dylikt (’displayer’)

XHTML har beskrivits som både det nästkommande steget inom HTML (se även detta begrepp) samt som en XML-applikation. XML är nämligen både en strukturerad uppsättning a regler som definierar hur särskild data skall delas ut på webben men på grund av att det även är självbeskrivande så kan det också beskriva förekomsten av en webbsida. Därmed så blir det logiskt att involvera HTML med XML, med resultatet av en XML-applikation som även utför det jobb som vanligtvis HTML gör (XHTML).

XML-namrymd
Se under begreppet ‘Namnrymd’.

XML Scheman
Ett sätt att definiera hur XML-dokumenten skall se ut. Utifrån ett XML Schema kan man validera dokumentet att det är korrekt utformat. Tanken är att det flexiblare XML Schema-språket skall ersätta den gamla DTD- (se även detta begrepp) definition (XML Scheman).

Ändpunkt (endpoint)
Ändpunkt (endpoint) är för närvarande synonymt med en port; En association mellan en bindning (se även detta begrepp) och en nätverksadress, specificerat av en URI (se även detta begrepp), som kan användas för att kommunicera med en instans av en tjänst. En port indikerar en specifik plats där access till en tjänst med hjälp av ett särskilt protokoll samt dataformat är möjlig (Ändpunkt).

7. Referenser

Böcker (Elektroniska böcker anges med: [Online])
Backman, Jarl. (1998). *Rapporter och uppsatser*. Lund : Studentlitteratur.

Britton, Chris. (2000*). It Architectures and middleware*. Harlow, UK: Addison-Wesley Publishing Company.

Celander, Lars. (2002). *XML, en skonsam men effektiv introduktion*. [Online]
Göteborg: Azelia AB.
<http://www.azelia.se/xml.pdf>
Senast besökt den 2004-02-06

Kroenke, David, M. (2000). *Database Processing*.(Sjunde upplagan). New Jersey, USA: Prentice-Hall.Inc.

Körner, Svante., & Wahlgren, Lars. (1996). *Praktisk Statistik*. Lund: Studentlitteratur.

Nordstedts Svenska Ordbok. (1990). Språkdata, Sture Allén och Nordstedts förlag 1990.

Skansholm, Jan. (1999). *Java direkt*. (Andra upplagan).Lund: Studentlitteratur.

Sommerville, Ian. (2001). *Software Engineering*,
(Sjätte upplagan). Harlow, UK: Addison-Wesley Publishers Limited

Thorell, Jerker. (1995). *IT & Datalexikon 1995*. (Första upplagen). Växjö: Liber Utbildning AB.

Artiklar
Cerami, Ethan. (2002). Top Ten FAQs for Web Services [Online]. *Webservices.xml.com.* Copyright [O'Reilly & Associates, Inc.] 2002-02-12.
<http://www.oreillynet.com/pub/a/webservices/2002/02/12/webservicefaqs.html>
Senast besökt den 2004-02-06

Erlanger, Leon. (2001). .NET may be the biggest change to Microsofts strategy since it introduced Windows 3.0 [Online]. *Looksmart.*
<http://www.findarticles.com/cf_0/m0DXS/6_7/71722533/p1/article.jhtml>
Senast besökt den 2004-02-06

Gustafson, Johan. .Net eller Java – det är inte frågan [Online]. *Microsoft executive circle.*
<http://www.microsoft.com/sverige/business/archive/articles/David_Chappell.asp>
Senast besökt den 2003-12-27

Hammarberg, Pär. (2002). Så är .NET uppbyggt - från XML till UDDI [Online].
Copyright [IDG, International Data Group AB] 2002-10-11.
<http://www.idg.se/ArticlePages/200210/10/20021010174320_IDG.se103/20021010174320_IDG.se103.dbp.asp>
Senast besökt den 2004-02-06

Hoffman, Allan. Web services and your skills [Online]. *Monster Technology*.
< http://technology.monster.com/articles/webserv2/**>**
Senast besökt den 2003-12-27

Loney, Matt. Midcrosoft .Net: The alternatives [Online]. *Zdnet UK*.
<http://insight.zdnet.co.uk/specials/xmldotnet/0,39021182,39115501,00.htm>
Senast besökt den 2003-12-27

Peterson, Gunnar. (July 2002). Component Security Design Considerations for J2EE and .Net – An Architectural ViewPart 3 [Online].
<http://downloads.securityfocus.com/library/ISB0707GP.pdf>
Senast besökt den 2004-01-28

Sabbadin, Enrico. (2003). .NET Identity and Principal Objects [Online]. *Inform IT*.
<http://www.informit.com/isapi/product_id~%7BE2C25438-25DA-476D-86A6-2623F97E27B0%7D/content/index.asp>
Senast besökt den 2004-02-20

Vetenskapliga artiklar

Aoyama, Mikio., Lea, Doug., Maruyama, Hiroshi., Sullivan, Kevin., Szyperski, Clemens., & Weerawarana, Saniiya. (2002). Web services engineering: promises and challenges. *International Conference on Software Engineering***.** *Proceedings of the 24th international conference on Software engineering*, s. 647-648.

Bravetti, Maroi., Gorrieri, Roberto ., Lucchi, Roberto., & Zavattaro, Gianluigi. (2004). Web Services for E-commerce: guaranteeing security access and quality of service. *Symposium on Applied Computing***.** *Proceedings of the 2004 ACM symposium on Applied computing,* s. 800-806.

Chen, Shiping., Gorton, Ian., Jiang, Ning., Liu,Anna., & Liu, Jan. (2002). Designing a test suite for empirically-based middleware performance prediction. *ACM International Conference Proceeding Series***.** *Proceedings of the Fortieth International Confernece on Tools Pacific: Objects for internet, mobile and embedded applications*, volym 10, s. 123-130

Engelen, Robert Van. (2004). Code generation techniques for developing light-weight XML Web services for embedded devices. *Symposium on Applied Computing.*
Proceedings of the 2004 ACM symposium on Applied computing, s. 854-861.

Gordon, Andrew.d., & Pucella, Riccardo. (2002). Validating a Web service security abstraction by typing. *Workshop On XML Security***.** *Proceedings of the 2002 ACM workshop on XML security*, s. 18-29.

Li, Chen., & Pahl, Claus. (2003). Security in the Web Services Framework. *ACM International Conference Proceeding Series Proceedings of the 1st international symposium on Information and communication technologies*, s. 481-486.

Vetenskapliga rapporter
Garlan, David & Shaw, Mary. (1994). *An introduction to software architecture* (Teknisk rapport nummer CMU-CS-94-166). Carnegie Mellon University, Pittsburgh Pennsylvania, USA.

White papers
Berard, Edward V. *Abstraction, Encapsulation, and Information Hiding*.
<http://www.toa.com/pub/abstraction.txt>
Senast besökt den 2004-02-21

Butler, Brian., & Farell, Bernard. *The White Papers.Benchmarking Basics.*
<http://www.dlt.com/quest/pdf/high%20availability/ecommerce/benchmarking%20basics.pdf>
Senast besökt den 2004-04-22

Box, Don., Ehnebuske, David., Kakivaya, Gopal., Layman, Andrew., Mendelsohn, Noah ., Frystyk Nielsen, Henrik., Thatte, Satish., & Winer, Dave. (2000). *Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000*. Copyright [DevelopMentor, International Business Machines Corporation, Lotus Development Corporation, Microsoft, UserLand Software] 2000-05-08.
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
Senast besökt den 2004-02-06

Farley, Jim (2000). *Microsoft vs J2EE: How Do They Stack Up?*
< java.sun.com/features/2000/11/dotnetvsms.html?frontpage-banner>
Senast besökt den 2003-12-27

Hanson, J.Jeffrey. (2002). *.NET versus J2EE Web services. A comparison of approaches*.
< http://www.webservicesarchitect.com/content/articles/hanson01print.asp >
Senast besökt den 2003-12-27

Kirtland, Mary. (2001). A Platform for Web Services [Online]. *Msdn*.
Om nedanstående länk inte fungerar gå till: 'http://msdn.microsoft.com/archive/' och sök på 'A Platform for Web Services'
<http://msdn.microsoft.com/archive/default.asp?url=/archive/enus/dnarxml/html/websvcs_platform.asp >
Senast besökt den 2004-02-06

Svoboda, Zdenek. (2002). *An Introduction To Web Services: Web Services Security*.
<http://www.theserverside.com/articles/article.jsp?l=Systinet-web-services-part-3>
Senast besökt den 2004-02-05

Watkins, Damien. (2000). Handling Language Interoperability with the Microsoft.NET Framework [Online]. *Msdn*.
<http://msdn.microsoft.com/library/techart/interopdotnet.htm>
Senast besökt den 2004-02-06

Vawter, Chad., & Roman, Ed. (2001). *J2EE vs Microsoft.NET. A Comparison of building XML-based web services.*
< http://www.theserverside.com/resources/article.jsp?l=J2EE-vs-DOTNET >
Senast besökt den 2003-12-27

Zukowski, John. *Introduction to the JavaBeans API.*
<http://developer.java.sun.com/developer/onlineTraining/Beans/JBeansAPI/shortcourse.html#BeansOverview>
Senast besökt den 2004-02-06

Populärpress

Computersweden. (1997). Tillämpningsservrar nästa webbtrend. Krävs att man anger CS webbkod. Copyright [ComputerSweden] 1997-12-17.
<http://computersweden.idg.se/text/971217-CS5>
Senast besökt den 2004-02-06

Computersweden. (2001). Webbtjänster hetast under Suns Javaone. Krävs att man anger CS webbkod. Copyright [ComputerSweden] 2001-06-05.
<http://computersweden.idg.se/text/010605-CS9>
Senast besökt den 2004-02-06

Hemrajani Anil. (1999). The state of Java middleware, Part 2: Enterprise JavaBeans [Online]. *JavaWorld.com*. Copyright [JavaWorld.com] April 1999.
<http://www.javaworld.com/javaworld/jw-04-1999/jw-04-middleware.html>
Senast besökt den 2004-02-06

Hillesley, Richard (2001). Tempted by .NET? [Online]. *Linuxuser*, Oktober utgåvan. Klicka på länken ’ Cover feature - Tempted by .NET?’ på nedanstående adress för att erhålla dokument.
<http://www.linuxuser.co.uk/articles/issue15>
Senast besökt den 2003-12-27

Jacobsson, Henrik. (1999). Sun vill knyta ihop javas olika delar [Online]. *Computer Sweden*. Krävs att man anger CS webbkod. Copyright [ComputerSweden] 1999-04-02.
<http://computersweden.idg.se/text/990402-CS2>
Senast besökt den 2004-02-06

Kempe, Lotta (2003-03-11). Amerikanska myndigheter går med i Liberty Alliance. *Computer Sweden* [online].
<http://www.idg.se/ArticlePages/200303/11/20030311115936_CS/20030311115936_CS.dbp.asp >
Senast besökt den 7 januari 2004

Lindberg, Magnus. (2001). .NET-tredje generationens utvecklingsmiljö. *Datormagazin*, nr 3, s. 136-138.

Lotsson, Anders (20020411). Microsoft skrotar ’My Services’. *Computer Sweden* [online].
<http://www.idg.se/ArticlePages/article404.asp?404;http://www.idg.se/ArticlePages/200204/11/20020411164400963_CS29/20020411164400963_CS29.dbp.asp>
Senast besökt den 7 januari 2004

Lurie, Jonathan., & Belanger, R. Jason. (2002). The great debate: .NET vs J2EE. Does one web services platform dominate the other? *JavaWorld* [online].
< http://www.javaworld.com/javaworld/jw-03-2002/jw-0308-j2eenet_p.html>
Senast besökt den 2003-12-27

Studentarbeten

Arnoldsson, Martin., Lupander, Erik., & Jönsson, Peter. (2003). *Skalbar och flexibel systemarkitektur med Web-services och J2EE. En Undersökning inom det starkt föränderliga och dynamiska domännamnsregistreringsområdet.* (Magistereuppsats, Handelshögskolan vid Göteborgs Universitet. Institutionen för informatik).
<http://www.handels.gu.se/epc/archive/00002932/01/Nr9_MA,EL, PJ.pdf>
Senast besökt den 2003-12-27

Balic, Michael., Björklund, Nicklas., & Jägmark, Magnus. (2001). *Application-server middleware. Hur utvecklas ett bra applications-server middleware?* (Kandidatexamen, Högskolan Borås. Institutionen för data- och affärsvetenskap).

Bergquist, Jonas., & Ericsson, Anders. (2001). *Enterprise JavaBeans ur ett systemutvecklingsperspektiv*. (Magisteruppsats, Göteborgs Universitet).
<http://www.handels.gu.se/epc/archive/00001694/01/bergqvist.ericsson.pdf>
Senat besökt den 2004-02-06

Johansson, Fredrik., & Ledin, Caroline. (2001). *Återanvändning av kod – kan .NET underlätta?* (Kandidatuppsats, Högskolan Borås. Institutionen för data- och affärsvetenskap).

Kajbrink, Janni., & Lorentsson, Johan. (1999). *Client/Server is dead! En studie av skiktade arkitekturer fokuserad på säkerhet, tillgänglighet och dynamik*. (Magisteruppsats, Göteborgs Universitet. Institutionen för Informatik).
<http://www.handels.gu.se/epc/archive/00002017/01/kajbrink.lorentsson.pdf>
Senast besökt den 2004-03-07

Krakowski, Ralf., & Dahlgren, Johan. (1998). *Viktiga egenskaper hos en applikation vid migrering till intranet*. (Magisteruppsats. Institutionen för informatik).
<http://www.handels.gu.se/epc/archive/00002156/01/Dahlgren.Krakowski.IA7400.pdf>
Senast besökt den 2004-02-06

Lidén, Jonas., & Svensson, Tommy. (2001). *Mjukvaruklalité och visuella programmeringsverktyg, en analys av Delphi och JBuilder*. (Kandidatuppsats, högskolan Borås. Institutionen för data- och affärsvetenskap).

Ljunggren, Eva., & Viker, Joakim. (2001). *Prestanda och portabilitet för komponentbaserade system*. (Magisteruppsats, Göteborgs Universitet).
<www.math.chalmers.se/~kentp/Xjobb/ejb.pdf>
Senast besökt den 2004-02-06

Prestandatester och specifikation för prestandatest
Middleware Company Case Study. (2003) J2EE and .NET (RELOADED). *Yet another performance study.*
<http://www.middleware-company.com/casestudy/tmc-performance-study-jul-2003.pdf>
Senast besökt den 2003-12-17

Nile Ecommerce Benchmark. (2001) *Microsoft .NET vs. Sun Microsystems J2EE. The Nile Ecommerce Application Server Benchmark.*
<http://www.gotdotnet.com/team/compare/Nile%20Benchmark%20Results.doc>
Senast besökt den 2003-12-17

The Middleware Company Application Server Plattform Baseline Specification
<http://www.middlewareresearch.com/endeavors/030519TMCBASESPEC/endeavor.jsp>
den Senast besökt den 2003-12-17

Information om applikationer, plattformar och tjänster
iPlanet. Tridions hemsida: iPlanet
<http://www.tridion.com/com/partners/partnerarchive/file_263220010412123006.asp>
Senast besökt den 2004-02-22

J2SDK-1.4.2.Introduction to j2sdk
http://www.fr.linuxfromscratch.org/view/blfs-cvs/general/j2sdk.html
Senast besökt den 2004-02-14

Java 2 Platform, Enterprise Edition (J2EE)
<http://java.sun.com/j2ee/>
Senast besökt den 2004-02-14

jGuru. (2000). Enterprise JavaBeans Fundamentals. *Java.sun.com* [online].
Copyright[java.sun.com] Maj 2000.
<http://developer.java.sun.com/developer/onlineTraining/EJBIntro/EJBIntro.html>
Senast besökt den 2004-02-06

Johnson, Mark. (1997). A walking tour of JavaBeans. *JavaWorld.com* [online]
Copyright [JavaWorld.com] Augusti 1997.
<http://www.javaworld.com/javaworld/jw-08-1997/jw-08-beans-p2.html>
Senast besökt den 2004-02-06

Jones, Hans. (2001). *n-tier-mts*.
Copyright [Hans Jones] 2001-05-08.
<http://users.du.se/~hjo/vb/n-tier-mts.pdf|>
Senast besökt den 2004-02-06

Leake, Gregory., & Duff, James. (2003). *Microsoft .NET Pet Shop 3.x: Design patterns and architecture of the .Net Pet Shop*. Dokument erhålles efter att ha klickat på länken ‘new .NET Pet Shop 3.0’ under nedanstående adress.
<http://www.gotdotnet.com/team/compare/default.aspx>
Senast besökt den 2003-12-27

Microsofts hemsida a). *Smart Devices and .NET*
<http://www.microsoft.com/net/products/devices.asp>
Senast besökt den 7 januari 2004

Microsofts hemsida b). *Microsoft Servers and .NET*
<http://www.microsoft.com/net/products/servers.asp>
Senast besökt den 7 januari 2004

Microsofts hemsida c). *Developer Tools and .NET*
<http://www.microsoft.com/net/products/tools.asp>
Senast besökt den 7 januari 2004

Microsofts hemsida d). *Gällande supportformer för .NET*
<http://members.microsoft.com/partner/support/default.aspx?nav=ln>
Senast besökt den 2004-02-03

Microsofts hemsida e). (2002). *Security with Microsoft .NET*: AnOverview. Gå till nedanstående länk och ladda där ner filen 'security_net.doc'.
<http://www.microsoft.com/net/business/security_net.asp>
Senast besökt den 2004-02-05

Microsoft Host Integration Server 2000
<http://www.microsoft.com/hiserver/evaluation/overview/default.asp>
Senast besökt den 2004-02-22

Microsofts svenska hemsida 1) Frågor och svar om .NET.
< http://www.microsoft.com/sverige/net/thisis/qa/default.asp#whatisnet >
Senast besökt den 2003-12-27

Microsofts svenska hemsida 2a). *Defining the Basic Elements of .NET*
< http://www.microsoft.com/net/basics/whatis.asp>
Senast besökt den 2003-12-27

Microsofts svenska hemsida 2b). *What Are Web Services?*
< http://www.microsoft.com/net/basics/webservices.asp>
Senast besökt den 2003-12-27

Newport, Billy. JMX.
<http://www.theserverside.com/resources/article.jsp?l=JMX>
Senast besökt den 2004-02-06

P5EE - J2EE Summary Information; Technologies/Techniques.
<http://www.officevision.com/pub/p5ee/j2ee.html>
Senast besökt den 2004-02-05

Produktportfolio för Microsoftprodukter a) Ladda ner ett dokument innehållandes information om produkten *Windows 2000 Server* från platsen för nedanstående länk
<http://web01.microsoft.se/portfolio/browse.asp?Folder=/product%20information%20sheets>
Senast besökt den 2004-02-15

Produktportfolio för Microsoftprodukter b) Ladda ner ett dokument innehållandes information om produkten *Windows Millenium Edition* från platsen för nedanstående länk
<http://web01.microsoft.se/portfolio/browse.asp?Folder=/product%20information%20sheets>
Senast besökt den 2004-02-15

Produktportfolio för Microsoftprodukter c) Ladda ner ett dokument innehållandes information om produkten *Microsoft SQL Server 2000 Enterprise Edition* från platsen för nedanstående länk
<http://web01.microsoft.se/portfolio/browse.asp?Folder=/product%20information%20sheets>
Senast besökt den 2004-02-15

SPEC Webbsida. Standard Performance Evaluation Corporation.
<http://www.spec.org/benchmarks.html>
Senast besökt den 2004-04-22

SpiritSoft. *SpiritWave Message Server, JMX Management Guide – Version 5.2*.
<http://www.beyondjms.com/documentation/wave/5.2.0/JMX_Managment_Guide_5.2.p
df>
Senast besökt den 2004-02-06

Sullins, Benjamin .G., & Whipple, Mark B. (2002). *JMX in Action*.
Manning Publications Company, November 2002.
<http://developer.java.sun.com/developer/Books/javaprogramming/jmx>
Senast besökt den 2004-02-06
Sun:s hemsida. Gällande kontrakt för support
<http://www.sun.com/service/warrantiescontracts/index.html>
Senast besökt den 2004-02-03

The Apache Jakarta Projekt (info om programmet JMeter)
<http://jakarta.apache.org/jmeter/index.html>
Senast besökt den 2004-02-14

The J2EE Tutorial - What Is a Message-Driven Bean? (2002).
Copyright [java.sun.com] 2002-04-24.
<http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts5.html#64640>
Senast besökt den 2004-02-06

The J2EE Tutorial – What Is an Enterprise Bean. (2002).
Copyright [java.sun.com] 2002-04-24.
<http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts2.html>
Senast besökt den 2004-02-06

Referenser kopplade till ordlista (kapitel 6)

Active Server Pages
<http://www.pcwebopaedia.com/TERM/A/Active_Server_Pages.html>
Senast besökt den 2004-02-21

ActiveX
<http://www.pcwebopaedia.com/TERM/A/ActiveX.html>
Senast besökt den 2004-02-21

ADO.NET dataläsare
<http://msdn.microsoft.com/data/DataAccess/adonet/default.aspx?pull=/library/en-us/dndotnet/html/adonetprogmsdn.asp#adonetprogmsdn_topic5>
Senast besökt den 2004-03-10

API
<http://www.pcwebopaedia.com/TERM/A/API.html>
Senast besökt den 2004-02-21

Applet
<http://www.pcwebopaedia.com/TERM/a/applet.html>
Senast besökt den 2004-02-21

Array
<http://www.pcwebopaedia.com/TERM/a/array.html>
Senast besökt den 2004-02-21

Asynkron
<http://www.itv.se/~a1000/dataordlista.html>
Senast besökt den 2004-02-21

Axis, “Axis User's Guide”.
<http://ws.apache.org/axis/java/user-guide.html>
Senast besökt den 2004-02-06

B2B
<http://www.pcwebopaedia.com/TERM/B/B2B.html>
Senast besökt den 2004-02-21

B2C
<http://www.pcwebopaedia.com/TERM/B/B2C.html>
Senast besökt av 2004-02-21

Back-end
<http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci212161,00.html>
Senast besökt den 2004-02-21

Bindning
<http://www.webopedia.com/TERM/B/bind.html>
Senast besökt den 2004-02-22

Cache
http://www.pcwebopaedia.com/TERM/c/cache.html
Senast besökt den 2004-02-22

Class loading
<http://java.sun.com/j2se/1.3/docs/api/java/lang/ClassLoader.html>
Senast besökt den 2004-03-10

CLI
<http://www.pcwebopaedia.com/TERM/C/Common_Language_Infrastructure.html>
Senast besökt den 2004-02-22

CLR
<http://www.pcwebopaedia.com/TERM/C/CLR.html>
Senast besökt den 2004-02-22

CMP2 (Container Managed Persistance, v2)
<http://www.informit.com/articles/article.asp?p=20944>
Senast besökt den 2004-03-10

COM
<http://www.pcwebopaedia.com/TERM/C/Component_Object_Model.html>
Senast besökt den 2004-02-22

Combs, Bob. Windows NT-Microsoft Transaction Server
<http://www.interex.org/pubcontent/enterprise/mar99/15winnt/15winnt.html>
Senast besökt den 2004-02-22

Community
<www.ezboard.com/help/glossary.html>
Senast besökt den 2004-03-06

Container
<http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci211833,00.html>
Senast besökt den 2004-02-22

DHTML
<http://www.pcwebopaedia.com/TERM/d/dynamic_HTML.html>
Senast besökt den 2004-02-22

Discovery and lookup-protokoll
<http://www.developer.com/services/print.php/10928_1014371_2>
Senast besökt den 2004-03-10

Distribuerade Datamodeller
<http://www.webopedia.com/TERM/D/distributed_computing.html>
Senast besökt den 2004-02-22

DLL
<http://www.pcwebopaedia.com/TERM/D/DLL.html>
Senast besökt den 2004-02-22

DTD
<www.rlg.org/redlightgreen/glossary.html>
Senast besökt den 2004-03-06

Dustin. (2003). Prisuppgifter från 2003-05-16
<http://www.dustin.se>
Senast besökt den 2004-03-07

Dynamic output caching
<http://www.javaworld.com/javaworld/jw-07-2002/jw-0726-j2eevsnet2-p2.html>
Senast besökt den 2004-03-02

EAI
<http://www.pcwebopaedia.com/TERM/E/EAI.html>
Senast besökt den 2004-02-22

ebXML
<http://www.pcwebopaedia.com/TERM/e/ebXML.html>
Senast besökt den 2004-02-22

Enterprise
<http://www.pcwebopaedia.com/TERM/e/enterprise.html>
Senast besökt den 2004-02-22

Entitet
<www.tldp.org/LDP/LDP-Author-Guide/glossary.html>
Senast besökt den 2004-03-06

Exekvera
<http://www.pcwebopaedia.com/TERM/e/execute.html>
Senast besökt den 2004-02-22

Förenade identiteter. (2002)
Publicerat 16 juli 2002.
<http://se.sun.com/nyheter/feature_stories/2002/020716/ >
Senast besökt den 2004-03-06

GNOME
<http://www.pcwebopaedia.com/TERM/G/GNOME.html>
Senast besökt den 2004-02-22

Gränssnitt
<http://www.pcwebopaedia.com/TERM/i/interface.html>
Senast besökt den 2004-02-22

GSS-API
<http://www.ietf.org/rfc/rfc2078.txt>
Senat besökt den 2004-02-22

GUI
<http://www.pcwebopaedia.com/TERM/G/Graphical_User_Interface_GUI.html>
Senast besökt den 2004-02-22

HTML
<http://www.pcwebopaedia.com/TERM/H/HTML.html>
Senast besökt den 2004-02-22

HTTP
<http://www.pcwebopaedia.com/TERM/H/HTTP.html>
Senast besökt den 2004-02-22

Identitet
<http://www.pcwebopaedia.com/TERM/i/identity.html>
Senast besökt den 2004-02-22

IIOP
<http://www.pcwebopaedia.com/TERM/I/IIOP.html>
Senast besökt den 2004-02-22

Implementation
<http://www.itv.se/~a1000/dataordlista.html>
Senast besökt den 2004-02-22

Interpreterare
<http://www.pcwebopaedia.com/TERM/I/interpreter.html>
Senast besökt den 2004-02-22

ISAPI
<http://www.pcwebopaedia.com/TERM/I/ISAPI.html>
Senast besökt den 2004-02-22

ISO
<http://www.pcwebopaedia.com/TERM/I/ISO.html>
Senast besökt den 2004-02-22

Java 2 Platform, Enterprise Edition (J2EE) Overview. (2002).
Copyright [java.sun.com] 2002-08-13.
<http://java.sun.com/j2ee/overview.html>
Senast besökt den 2004-02-06

Java Authentication and Authorization Service (JAAS) Overview.
<http://java.sun.com/products/jaas/overview.html>
Senast besökt den 2004-02-05

Java Cryptography Extension (JCE)
<http://java.sun.com/products/jce/>
Senast besökt den 2004-02-05

Java Management Extensions (JMX) Specification.
<http://www.jcp.org/en/jsr/detail?id=3>
Senast besökt den 2004-02-06

Java Secure Socket Extension (JSSE)
<http://java.sun.com/products/jsse/>
Senast beskt den 2004-02-05

Java Web Services Developer Pack
<http://java.sun.com/webservices/downloads/webservicespack.html>
Senast besökt den 2004-02-14

jBoss
<http://www.pcwebopaedia.com/TERM/J/JBOSS.html>
Senast besökt den 2004-02-22

JCA
<http://java.sun.com/j2ee/connector/>
Senast besökt den 2004-02-22

JDBC
<http://www.pcwebopaedia.com/TERM/J/JDBC.html>
Senast besökt den 2004-02-22

JIT
<http://www.pcwebopaedia.com/TERM/J/JIT.html>
Senast besökt den 2004-02-22

JNI
<http://www.pcwebopaedia.com/TERM/J/JNI.html>
Senast besökt den 2004-02-22

JRE
<http://www.pcwebopaedia.com/TERM/J/Java.html>
Senast besökt den 2004-02-22

JSP
<http://www.pcwebopaedia.com/TERM/J/JSP.html>
Senast besökt den 2004-02-23

JVM
<http://www.itv.se/~a1000/dataordlista.html>
Senast besökt den 2004-02-22

Kerberos
<http://web.mit.edu/kerberos/www/#what_is>
Senast besökt den 2004-02-22

Klass
<http://www.pcwebopaedia.com/TERM/c/class.html>
Senast besökt den 2004-02-22

Kodbas
<incubator.apache.org/drafts/glossary.html>
<www.openwings.org/openwings-0.9.2/tutorial/Trail_Glossary/01_Glossary.html>
Senast besökta den 2004-02-22

Logisk vy
<http://www.cs.vu.nl/~eliens/online/oo/II/7/architecture.html>
Senast besökt den 2004-03-10

Marshalling
<http://www.pcwebopaedia.com/TERM/d/data_marshalling.html>
Senast besökt den 2004-02-22

Metadata
<http://www.pcwebopaedia.com/TERM/m/metadata.html>
Senast besökt den 2004-02-22

Modulär
<http://www.pcwebopaedia.com/TERM/m/modular_architecture.html>
Senast besökt den 2004-02-22

MSMQ
<http://www.microsoft.com/windows2000/technologies/communications/msmq/default.asp>
Senast besökt den 2004-02-22

MSXML
<http://www.2000trainers.com/article.aspx?articleID=131&page=2>
Senast besökt den 2004-02-22

Namnrymd
<http://www.pcwebopaedia.com/TERM/n/namespace.html>
Senast besökt den 2004-02-22

Objekt
<http://www.pcwebopaedia.com/TERM/o/object.html>
Senast besökt den 2004-02-22

OLE DB
<http://www.pcwebopaedia.com/TERM/O/OLE.html>
Senast besökt den 2004-02-22

Open Source
<http://www.webopedia.com/TERM/O/open_source.html>
Senast besökt den 2004-02-22

Output caching
<http://gangadhara.europe.webmatrixhosting.net/WeDox/NET/OutputCaching.htm>
Senast besökt den 2004-02-22

Parsing
<orworld.uni-paderborn.de/downloads/glossary/glossary.html>
<www.cs.ukc.ac.uk/people/staff/djb/book/glossary.html>
Senast besökta den 2004-02-23

RDBMS
<http://www.pcwebopaedia.com/TERM/R/RDBMS.html>
Senast besökt den 2004-02-23

Sampler
<http://jakarta.apache.org/jmeter/usermanual/test_plan.html#samplers>
Senast besökt den 2004-03-10

Serialisering
<http://p2p.wrox.com/archive/beginning_php/2002-05/10.asp>
Senast besökt den 2004-03-01

Servlet
<http://www.pcwebopaedia.com/TERM/s/servlet.html>
Senast besökt den 2004-02-23

Shankland, Stephen . *Sun promotes Java on Cobalt servers*
Senast uppdaterad 4 Juni, 2001.
<http://news.com.com/2100-1001-267813.html?legacy=cnet>
Senast besökt den 2004-02-22

Shared context
<http://www.w3.org/2001/03/WSWS-popa/paper29>
Senast besökt den 2004-03-06

Slutna objekt (sealed objects)
<http://www.javaworld.com/javaworld/jw-11-2000/jw-1117-howto.html>
Senast besökt den 2004-03-10

SOAP action parameter
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383528>
Senast besökt den 2004-03-06

SPKM
<http://www.faqs.org/rfcs/rfc2025.html>
Senast beskt den 2004-03-06

SQL
<http://www.pcwebopaedia.com/TERM/S/SQL.html>
Senast besökt den 2004-02-23

SSL
<http://www.pcwebopaedia.com/TERM/S/SSL.html>
Senast besökt den 2004-02-23

State
<http://www.pcwebopaedia.com/TERM/s/state.html>
Senast besökt den 2004-02-23

Säkra strömmar (secure streams)
<http://www.wedgetail.com/technology/crypto.html>
Senast besökt den 2004-03-10

Tablet PC
<http://www.pcwebopaedia.com/TERM/t/tablet_PC.html>
Senast besökt den 2004-02-23

Tomcat
<www.itdimensions.com/support/faq/general/terms_glossary.html>
Senast besökt den 2004-02-23

Transparent Fail-over
<http://www.pcwebopaedia.com/TERM/t/transparent.html>
<http://www.webopedia.com/TERM/F/failover.html>
Senast besökta den 2004-02-23

URI
<www.gerbilbox.com/newzilla/glossary.php>
Senast besökt den 2004-03-03

URL
<http://www.pcwebopaedia.com/TERM/U/URL.html>
Senast besökt den 2004-02-23

Webopedia (2004). Begreppet Portable (gällande mjukvara)
<http://www.pcwebopaedia.com/TERM/p/portable.html>
Senast besökt den 2004-02-02

Visual Studio.NET
<http://www.faqts.com/knowledge_base/view.phtml/aid/13912>
Senast besökt den 2004-02-23

WML
<www.3gnewsroom.com/html/glossary/w.shtml>
<www.10meters.com/communications_glossary.html>
Senast besökta den 2004-02-23

Wrapper
<http://www.pcwebopaedia.com/TERM/w/wrapper.html>
Senast besökt den 2004-02-23

XHTML
<http://www.pcwebopaedia.com/TERM/X/XHTML.html>
<www.dorsai.org/~walts/xglossary.html>
Senast besökta den 2004-02-23

XML Scheman
<http://www.w3.org/XML/Schema>
Senast besökt den 2004-03-06

Ändpunkt
<www.w3.org/TR/ws-gloss/>
Senast besökt den 2004-03-03