Handelshögskolan vid Göteborgs universitet
Institutionen för Informatik
VT 2001
Magisteruppsats 20p

Java applets

- en bra teknik för klientbaserad programmering?

Abstrakt

Den här magisteruppsatsen behandlar begreppet klientbaserad programmering som en teknik för att utveckla webbapplikationer. Undersökningen fokuseras kring frågeställningen: Är Java applets en bra teknik för att utveckla klientbaserade applikationer, med avseende på plattformsoberoende, säkerhet och funktionalitet? För att svara på denna fråga utformades tre delsyften: kartlägga bakgrunden till klientbaserad programmering, jämföra teknikerna Java applets och ActiveX samt utveckla en exemplifierande applikation med Java applets. Undersökningen utfördes på uppdrag av IT-företaget Resco som hade intresset och behovet av en klientbaserad filuppladdnings- och nedladdningsapplikation. Undersökningen är en kvalitativ studie med en, i huvudsak, deskriptiv undersökningsansats och bygger på litteraturstudier samt egna erfarenheter av utvecklingen av en Java applet. Resultatet av denna undersökning är att Java, som programmeringsspråk, har ett starkt utvecklat stöd för att utforma plattformsoberoende applikationer på ett, för användaren, säkert sätt och Java applets är därför en bra teknik för utveckla klientbaserade applikationer.

Författare: Ulrika Bramer
Handledare: Agneta Ranerup

INNEHÅLLSFÖRTECKNING

1 Inledning

Klientbaserad programmering och mobil kod är idag inga nya begrepp. Begreppen refererar till olika teknologier som utökar en webbläsares kapacitet, från att enbart presentera statisk information till att exekvera programkod. Klientbaserad programmering ger webbutvecklare möjligheten att skapa dynamiska webbapplikationer som effektivt utnyttjar tillgängliga resurser, samt ökar interaktiviteten med användare. Bakgrunden till klientbaserad programmering är den explosionsartade utvecklingen av Internet och World Wide Web (WWW) och den nya applikationsmiljö det har erbjudit. WWW har, enligt Hayes [2000], genomgått tre distinkta generationsväxlingar när det gäller utveckling och gruppering av applikationer:

- Statisk HTML: Från början såg man WWW som ett sätt att presentera och länka dokument. Webbapplikationer användes enbart för att förflytta användare från en statisk webbsida till en annan, utan någon form av interaktivitet.
- Serverbaserade applikationer: När man började inse WWW:s stora potential startade utvecklingen av serverbaserade applikationer, s k CGI-program. Dessa applikationer körs på en webbserver och gör det möjligt att skapa dynamiska webbapplikationer och användarinteraktivitet.
- Nätverksapplikationer: Tredje generationens webbapplikationer är både server- och klientbaserade. Utvecklare delar upp webbapplikationen på ett sätt som effektivt utnyttjar datormiljö och resurser. Detta angreppssätt ger en ökad datorkapacitet eftersom det bygger på det faktum att den dator där webbläsaren körs besitter outnyttjad datorkraft.

Klientbaserad programmering tillhör, enligt ovanstående indelning, tredje generationens webbapplikationer, vilka ger webbutvecklare möjligheten att utnyttja resurser hos klienten, t.ex. det lokala filsystemet. Att ge en webbapplikation tillgång till resurser hos klienten kan även innebära en säkerhetsrisk. Det kan innebära att klientdatorn utsätts för fientliga angrepp och attacker. Därför är tekniker för klientbaserad programmering, t.ex. Java applets, ActiveX, omgärdade av rigorösa säkerhetskontroller. Detta för att undvika att utnyttjandet sker på ett okontrollerat och otillbörligt sätt[1]. I vissa fall kan det dock vara önskvärt att utveckla webbapplikationer som tar sig igenom säkerhetskontrollerna och utför operationer på webbläsarens, d.v.s. klientens, lokala datorsystem. Ett exempel på en sådan applikation är ett program som sköter uppladdning och nedladdning av filer mellan server och klient. Denna uppsats kommer att behandla just detta område, d.v.s. hur man med hjälp av en klientbaserad Java applet kan ladda upp och ned filer mellan klient och server.

Uppsatsen består av en praktisk och en teoretisk del där utvecklingen av denna Java applet utgör den praktiska delen. Den teoretiska delen består av en redogörelse av hur klientbaserad programmering har växt fram, en beskrivning av några av de tekniker som finns för att åstadkomma klientbaserade applikationer samt en jämförelse av två tekniker, Java applets och ActiveX.

[1] Hayes, 2000

Uppsatsen utförs på uppdrag av företaget Resco AB. Resco är ett konsultföretag som specialiserat sig på att erbjuda helhetslösningar när det gäller bl.a. affärssystem, verksamhetsutveckling, systemintegration, IT-arkitektur, webbapplikationer och webbdesign[2]. I ett av Göteborgskontorets kundprojekt uppstod behovet av en applikation som gjorde det möjligt för projektdeltagare att ladda ner en fil från en webbsida, arbeta med den på den lokala datorn och sedan ladda upp den igen. En applikation, uppbyggd av Java applets, hade köpts in för att lösa uppgiften. Denna applikation visade sig inte fungera på ett tillfredställande sätt och idén att utveckla en egen variant växte fram. Resco var även intresserade av att se vilka andra tekniker och möjligheter som fanns för att lösa uppgiften samt om och varför Java applets var att föredra. Utformningen av problemområde, syfte och frågeställningar skedde i samråd med min handledare på Resco, tillika projektledare för ovan nämnda kundprojekt, Markus Östlund. Under dessa diskussioner var tekniken ActiveX ett återkommande begrepp när det gällde att hitta andra tekniker och möjligheter för att lösa uppgiften. Eftersom många jämförelser mellan Sun Microsystems Java och Microsofts ActiveX visade sig ha gjorts av anhängare och utvecklare av respektive föreföll det naturligt att avslutningsvis koncentrera uppsatsens teoretiska del kring dessa två tekniker.

Undersökningen huvudsakliga målgrupp är uppdragsgivaren Resco och de webbutvecklare som deltar i det aktuella kundprojektet. Undersökningen syftar dock till att ge en allmän bild av klientbaserad programmering och klargöra om det är en användbar teknik i liknande webbutvecklingar. Undersökningen riktar sig därför till webbutvecklare i allmänhet och en del tekniska begrepp och förklaringar kan därför anses som otillräckliga för ”oinsatta” läsare.

1.1 Syfte och frågeställning

Resco var i behov av en plattformsoberoende filuppladdning/nedladdning-sapplikation. Eftersom denna applikation skulle vara klientbaserad, d.v.s. laddas ned och exekveras på klienten, var även säkerheten och funktionalitet två intressanta aspekter. Jag valde därför att utforma uppsatsens övergripande frågeställning enligt följande:

Är Java applets en bra lösning för att utveckla en klientbaserad applikation, med avseende på plattformsoberoende, säkerhet och funktionalitet?

För att svara på frågeställningen har jag satt upp tre delsyften.

- Undersöka begreppet klientbaserad programmering och beskriva några av de tekniker som används för att utveckla klientbaserade applikationer.
- Jämföra två tekniker, Java applets och ActiveX, med avseende på plattformsoberoende, säkerhet och funktionalitet?
- Utveckla en klientbaserad filuppladdning/nedladdningsapplikation med Java applets för att kartlägga eventuella problem och begränsningar med denna teknik.

[2] www.resco.se

1.2 Uppsatsens disposition

Uppsatsens består av följande 8 kapitel:
Kapitel 1, Inledning, ger en bakgrundsbild till varför studien genomförs och därefter en beskrivning av uppsatsens syfte och frågeställningar.
Kapitel 2, Metod, beskriver olika undersökningsansatser och preciserar vilken ansats som används i denna uppsats. Vidare ges en beskrivning av undersökningens utformning samt en utvärdering av källkritik och undersökningens kvalitet. Metodavsnittet avslutas med en problemavgränsning.
Kapitel 3, Distribuerade system, har avsikten att ge läsaren en bakgrundsbild till klientbaserad programmering och beskriva hur utvecklingen av datorsystem har lett fram till en framväxt av denna teknik.
Kapitel 4, Klientbaserad programmering, beskriver begreppet klientbaserad programmering och ger exempel på olika tekniker för att åstadkomma denna typ av programmering. Kapitlet avslutas med en utförlig beskrivning av Java applets och ActiveX.
Kapitel 5, Resultat består av två delar. Den första delen innehåller jämförelsen mellan teknikerna Java applets och ActiveX, utifrån flyttbarhet, säkerhet och funktionalitet. Denna del avslutas med en diskussion kring vilken teknik Resco bör använda sig av och varför. Den andra delen beskriver tillvägagångssättet och resultatet av utvecklingen av den klientbaserade applikationen. Denna del avslutas med en diskussion kring de svårigheter och begränsningar jag har stött på under utvecklingen.
Kapitel 6, Sammanfattning och slutsats, ger en övergripande sammanfattning av de resultat jag redovisat och avslutas med de slutsatser jag kommit fram till.
Kapitel 7, Litteraturförteckning, innehåller en förteckning av det material som använts i och under arbetet med uppsatsen.
Kapitel 8, Bilagor, innehåller koden till applikationen FileApplet.

2 Metod

Metod är ett verktyg för att lösa ett problem. Avsikten med detta kapitel är att beskriva det praktiska tillvägagångssättet vid insamlingen av den information som behövdes för att uppfylla uppsatsens frågeställning och syften. För att utröna vilken metod som är lämplig för detta ändamål beskrivs först kortfattat de metoder som finns, för att senare motivera uppsatsens metod. Tillvägagångssättet vid datainsamlingen kommer även att beskrivas samt en diskussion kring uppsatsens kvalitet och tillförlitlighet.

2.1 Undersökningsansatser

Det finns en rad olika metoder att tillgå för att undersöka ett problemområde. Olika undersökningar kan klassificeras utifrån hur mycket information eller kunskap det finns inom det specifika problemområdet och den, för uppsatsen, valda undersökningsansatsen specificerar viken typ av information som ska samlas in och vilka datakällor som används vid denna insamling[3]. Nedan beskrivs två vanligas undersökningsansatser, explorativ och deskriptiv ansats .

När det finns luckor i kunskapsbasen för problemområdet kommer undersökningen att vara utforskande, d.v.s. explorativ. Det främsta syftet med explorativa undersökningar är att inhämta så mycket information som möjligt om ett bestämt problemområde för att få en allsidig bild. En explorativ undersökning kan ligga till grund för vidare studier eftersom den handlar om att identifiera problem och man använder sig ofta av flera olika tekniker för att inhämta information.

När det redan finns en viss mängd kunskap om problemområdet kommer undersökningen att få en mer beskrivande karaktär, en s.k. deskriptiv ansats. Vid deskriptiva undersökningar begränsar man sig till att undersöka några aspekter av det fenomen man är intresserad av och de beskrivningar som görs är detaljerade och grundliga. Ofta används endast ett fåtal tekniker för att samla information vid en deskriptiv undersökning[4].

2.1.1 Vald undersökningsansats

I denna uppsats används i huvudsak, en deskriptiv undersökningsansats. Detta görs p.g.a. att problemområdet, klientbaserad programmering, inte är ett nytt fenomen och mycket kunskap och information finns att tillgå. Studier och insamlingar av bakgrundsinformation om klientbaserad programmering kommer att göras, dels för att svara på hur denna teknik har växt fram och dels för att ge exempel på olika verktyg för att åstadkomma den här typen av programmering. Vidare kommer undersökningen att koncentreras på två tekniker, ActiveX och Java applets. Utifrån tidigare undersökningar och artiklar kommer en jämförelse mellan dessa att göras. Undersökningen kommer även att innefatta en utveckling av en klientbaserad applikation med Java applets. Denna utveckling har avsikten att ge en praktisk bild av hur klientbaserad programmering kan användas och vilka problem och

[3] Patel, Davidsson, 1994
[4] Ibid

begränsningar man kan stöta på. Jämförelsen mellan ActiveX och Java applets samt utvecklingen av en egen applikation ger undersökningen, förutom en deskriptiv, även en utvärderande karaktär.

2.2 Undersökningens utformning

För att uppfylla syftet med uppsatsen och för att undersökningens resultat ska kunna analyseras på ett rättvisande sätt är det viktigt att veta vilka typer av data och insamlingsmetoder som används[5].

2.2.1 Primär- och sekundärdata

Den data som samlas in kan vara av två olika typer, primär- och sekundärdata. Primärdata är den data som samlats in av uppsatsskrivaren själv, d.v.s. data som inte existerar innan undersökningen påbörjas. Sekundärdata är data som redan existerar, t.ex. tidigare undersökningar, offentlig statistik m.m.[6]. Insamling av primärdata sker vanligtvis genom egna experiment medan sekundärdata bör utnyttjas för att kartlägga ett visst problemområde. Ofta ger inte sekundärdata en fullständig bild av det informationsbehov som undersökningen ska tillfredställa och kan då ofta kompletteras med primärdata. Svårigheten med sekundärdata är att avgöra om den framtagna informationen är relevant för undersökningen. Det kan även vara svårt att bedöma källans tillförlitlighet och tidsförankring[7].

2.2.2 Kvalitativt och kvantitativt förhållningssätt

Kvalitativt och kvantitativt förhållningssätt kan, något förenklat, sägas syfta till hur man väljer att bearbeta och analysera den information som samlats in. Med kvantitativ forskning avses forskning som använder sig av statistiska bearbetnings- och analysmetoder och med kvalitativ avses forskning som använder sig av verbala analysmetoder. Det som är avgörande för valet av förhållningssätt i en uppsats är hur undersökningsproblemet är formulerat. Om man i huvudsak är intresserad av att svara på frågor som rör ”Var?”, ”Hur?”, ”Vilka är skillnaderna?” eller ”Vilka är relationerna?” bör man använda sig av ett kvantitativt förhållningssätt. Om problemet däremot handlar om att förstå människors upplevelser eller om man vill ha svar på frågor som rör ”Vad är detta?”, ”Vilka är de underliggande mönsterna?” så är ett kvalitativt förhållningssätt att föredra[8].

2.2.3 Val av undersökningens utformning

Under arbetet med denna uppsats används såväl primär- som sekundärdata. Sekundärdata utgörs av insamlad information, i form av böcker, rapporter och artiklar, om klientbaserad programmering, olika tekniker samt jämförelser mellan ActiveX och Java applets. Primärdata utgörs av de erfarenheter jag anskaffar mig

[5] Patel, Davidsson, 1994
[6] Wiedersheim, Eriksson, 1997
[7] Ibid
[8] Patel, Davidsson 1994

under utvecklingen av en klientbaserad applikation. Undersökningen bygger på tolkningar och slutsatser av insamlad sekundärdata samt upplevelser och erfarenheter under utvecklingen av en Java applet. Eftersom undersökningen av klientbaserad programmering dessutom till stor del har syftet att svara på frågan ”Vad är detta?” används ett kvalitativt förhållningssätt.

2.3 Undersökningens genomförande

Som tidigare nämnts hade Resco köpt in en applikation för att lösa ett problem i ett kundprojekt. Eftersom denna applikation inte fungerade på ett tillfredställande sätt uppstod ett kunskapsbehov om klientbaserad programmering samt ett behov av en exemplifierande klientbaserad applikation som visar hur denna typ av programmering kan gå till. Därför inleddes undersökningen med en omfattande litteratursökning. Den utgick ifrån begreppet klientbaserad programmering och ett antal olika begrepp som kan betraktas som intressanta detta sammanhang, t.ex. distribuerade system, klient/server, klientbaserad programmering, mobil kod, Java applets, ActiveX m.m. Information samlades in från böcker, artiklar och rapporter.

Undersökningen inleddes med en sammanställning av det material som behandlade bakgrunden till klientbaserad programmering, vad begreppet står för, användningsområde m.m. Resultatet av denna sammanställning återfinns i kapitel 3, Distribuerade system, samt kapitel 4, Klientbaserad programmering. När bakgrundmaterialet till klientbaserad programmering var klar koncentrerades undersökningen till en jämförelse mellan ActiveX och Java applets. För att samla in information om dessa tekniker sökte jag dels efter undersökningar, rapporter och artiklar som behandlade de aktuella teknikerna var för sig och dels efter undersökningar, rapporter och artiklar som jämförde de båda teknikerna. De aspekter jag tittade på var plattformsoberoende, säkerhet och funktionalitet.

Utvecklingen av den klientbaserade applikationen utfördes parallellt med och efter det att litteraturstudien, sammanställningen av bakgrundsmaterialet och jämförelser mellan ActiveX och Java applets var slutförd. Under utvecklingen av applikationen användes Internetbaserade java-manualer, s.k tutorials, om applets, nätverkskopplingar, filöverföringar m.m. För att lösa en del svårigheter deltog jag även i olika användarforum för Javautveckling på Internet. Denna del av undersökningen inleddes med en studie av hur Java applets fungerar, eventuella begränsningar med tekniken, säkerhetsmodell m.m. Utifrån den kravspecifikation Resco sammanställt för applikationen (se kapitel 5.2.1) koncentrerade jag mig sedan på att undersöka hur man i Java kan lösa de funktionella kraven. Den kod jag skrev testades först i vanliga skrivbordsapplikationer för att sedan utformas till en Java applet.

2.4 Källkritik och undersökningskvalitet

Risken för fel finns genom hela undersökningsprocessen. Eftersom olika felkällor kan leda till att felaktiga slutsatser dras, är det viktigt att i största möjliga mån försöka eliminera dem. Det är därför viktigt att vara medveten om de olika felkällorna som kan påverka uppsatsens kvalitet och inriktning[9].

[9] Patel, Davidsson, 1994

2.4.1 Källkritik

För att uppnå saklighet och objektivitet i förhållandet till sitt eget material och för att kunna göra en bedömning om dess fakta är trovärdiga måste man förehålla sig kritiskt till de dokument man använder sig av. Man måste ta reda på när och var dokumenten tillkommit, varför det har tillkommit, vilket syfte upphovsmannen hade och under vilka omständigheter det tillkom.

Ett annat problem är att bestämma hur mycket material i form av dokument som ska samlas in. Oavsett mängden material måste man se till att få en allsidig bild av problemområdet. Genom att bara välja vissa fakta kan man åstadkomma skevhet och undersökningen ger en falsk bild av verkligheten[10]. Ett material kan bl.a. bedömas utifrån följande tre kriterier:

- Samtidskrav, en källas värde minskar ju längre tid som gått sedan källan framställdes.
- Tendenskrav, ett övervägande för att värdera om källan framställs vinklad för att vinna gehör.
- Beroendekritik, en källa bedöms huruvida den påverkats av andra informationsgivare[11].

När man använder sig av dokument från Internet är det extra viktigt att kritiskt granska informationen innan den används. På Internet finns ingen kontroll av det som publiceras och det behöver inte heller finnas någon angivelse om vem som skrivit texten eller när den publicerats[12].

2.4.2 Undersökningskvalitet

Undersökningskvalitet kan bedömas utifrån de metoder och de källor man använt för att utföra undersökningen. För att uppsatsen resultat ska bedömas som vetenskapligt måste de metoder och tekniker som används uppfylla kraven för validitet samt reliabilitet. Validitet innebär att man undersöker det man avser att undersöka och reliabilitet att det görs på ett tillförlitligt sätt[13]. För att säkerställa validiteten en i undersökning är det viktigt att vara kritisk till den information man samlar in och de slutsatser man drar. Man bör kontinuerligt ifrågasätta om undersöknigen uppfyller uppsatsens syfte och frågeställningar[14]. Reliabiliteten i en undersökning säkerställs genom att man använder flera olika metoder på samma problem eller genom att samla in data från flera oberoende källor. I kvalitativa undersökningar är det svårt att säkerställa reliabiliteten. Resultatet bygger personliga tolkningar och det finns inga instrument som registrerar ”rätt” resultat att tillgå. Vid denna typ av undersökningar får man istället förlita sig på uppsatsskrivarens förmåga att ta fram ett tillförlitligt

[10] Patel, Davidsson, 1994
[11] Wiedersheim, Eriksson, 1997
[12] http://student.educ.umu.se/~dis99-19/kallkritik/internetkritik.html
[13] Patel, Davidsson, 1994
[14] Thurén, 1994

resultat. Undersökningskvalitet i kvalitativa studier beror därför på uppsatsskrivarens perspektivmedvetenhet, logik och förmåga att ta fram data med god kvalitet[15].

2.4.3 Diskussion angående uppsatsen källkritik och undersökningskvalitet

Uppsatsen bygger till stor del på litteratur och dokument från Internet, vilket ställer höga krav på källkritik. Materialet har därför samlats in med ovanstående kravkriterier i åtanke. För att säkerställa samtidskravet har jag försökt leta efter nyskrivna dokument angående klientbaserad programmering. De flesta dokument som behandlar detta område, speciellt jämförelser mellan ActiveX och Java applets, har dock visat sig vara 1997-1998 då tekniken var på uppgång och gavs mycket utrymme i diskussionsforum och tekniska rapporter. För att motsvara samtidskravet har jag därför försökt säkerställa insamlad fakta med aktuell produktinformation.

Eftersom jag använt mig av tekniska rapporter och manualer från olika företag, som utvecklar klientbaserade applikationer samt utvecklar tekniker för denna programmering, har det även varit viktigt att undersöka dokumenten utifrån tendenskrav och beroendekrav. Dokument från företag är i många fall vinklade för att framhäva en produkt eller en teknik i jämförelse med andra, speciellt när det gäller två konkurrerande tekniker som ActiveX och Java applets. En ansats har därför gjorts att försöka hitta oberoende artiklar och rapporter när det gäller jämförelser av dessa båda tekniker samt hitta en balans mellan antalet produktvinklade dokument från olika företag och utvecklare.

Undersökningens validitet säkerställs genom att jag, under arbetets gång, varit kritisk till det material som använts. Klientbaserad programmering är ett brett område och mycket information finns att tillgå. Ansatsen har dock varit att uppfylla uppsatsens frågeställning och syften och på så sätt säkerställa att resultatet behandlar det som avsetts att undersökas.

Angående undersökningens reliabilitet har jag försökt säkerställa denna genom att använda ett stort antal oberoende källor. Reliabiliteten säkerställs ofta genom att man använder flera olika metoder på samma problem och när det gäller denna undersökning använder jag mig av en litteraturstudie samt utförande av egna experiment. I efterhand anser jag att intervjuer med kunniga personer inom klientbaserad programmering, ActiveX och Java applets hade varit intressant vid jämförelsen mellan dessa tekniker samt ytterligare stärkt undersökningens reliabilitet

Uppsatsen grundas på en kvalitativ undersökning vilket innebär att den till stor del baseras på egna tolkningar av fakta samt idéer om hur en klientbaserad applikation kan utformas. Detta innebär att det inte går att dra några generella slutsatser utifrån resultatet. Det är med andra ord inte säkert att andra hade kommit fram till samma slutsatser och resultat även om de använt samma undersökningsansats med tillgång till samma information. En kvalitativ studie bygger dock på att förstå snarare än att skapa generell statistik och låg reliabilitet är därför något som är utmärkande för denna typ av studier.[16] Något som dock stärker reliabiliteten, i denna undersökning, är att jag har försökt vara medveten om olika perspektiv på problemområdet. Detta

[15] Wiedersheim, Eriksson, 1997
[16] Patel, Davidsson, 1994

har skett genom att använda ett stort antal källor samt att ta fram material av god kvalitet enligt kriterierna för källkritik.

2.5 Problemavgränsning

De senaste åren explosionsartade utveckling av webbaserade applikationer har inneburit att ett stort antal tekniker för att åstadkomma klientbaserad programmering har presenterats på marknaden. Avsikten med denna uppsats är ej att beskriva och kartlägga samtliga av dessa tekniker utan att beskriva vad klientbaserad programmering innebär och ge några exempel på tekniker som tagits fram för att åstadkomma denna typ av programmering. När det gäller jämförelsen mellan ActiveX och Java applets finns det flera olika aspekter som kan behandlas. Jag valt att endast titta på tre aspekter, plattformsoberoende, säkerhet och funktionalitet. Dessa aspekter är centrala i kravspecifikationen av den klientbaserade applikationen (se kapitel 5.2.1). Resco var intresserade av en plattformsoberoende applikation som på ett, för klienten, säkert sätt kunde skriva och läsa från den lokala hårddisken. Utöver detta var kravet att applikationen skulle bestå av så lite kod som möjligt vilket i sin tur ställer krav på verktygets inbyggda förutsättningar och funktioner.

3 Distribuerade system

Ett av syftena med denna uppsats är att svara på vad klientbaserad programmering är och hur det har växt fram. Ordet klientbaserad ger en vink om att den här typen av programmering härstammar ifrån distribuerade system och dess applikationsutveckling enligt klient/server-modellen. I detta kapitel har jag därför valt att inledningsvis beskriva distribuerade system och klient/server-modellen för att sedan gå in på den miljö där de klientbaserade verkar, d.v.s. Internet och WWW. I denna webbaserade miljö förekommer en rad begrepp, t.ex. URL, HTTP, statiska och dynamiska webbsidor, och jag har även valt att kortfattat beskriva dessa. Innehållet i detta kapitel kan upplevas som överflödigt för vana webbutvecklare, men avser att skapa en bild av hur och varför klientbaserad programmering har växt fram.

3.1 Introduktion

Under arbetet med denna uppsats har jag stött på många olika definitioner av distribuerade system. Den mest förenklade versionen beskriver ett samarbete mellan individuella komponenter för att uppnå ett allmänt mål. En annan vanlig aspekt var att ett bra uppbyggt distribuerat system kännetecknas av att användaren inte märker att det existerar. I ett distribuerat system skall datorerna dessutom samordna aktiviteterna och dela resurser som hårdvara, mjukvara och data. Coulouris, Dollimore och Kindberg [sid 5, 2001] definierar ett distribuerat system enligt följande:

”Ett distribuerat system består av komponenter, lokaliserade i datorer och sammankopplade i nätverk, som kommunicerar och koordinerar sitt agerande enbart genom att skicka meddelanden”.

Denna definition ger ett distribuerat system följande karaktärsdrag:

- Konkurrens. I ett nätverk är konkurrerande programexekvering ett viktigt element eftersom flera användare delar på resurser som webbsidor, filer och program under tiden de arbetar. För att detta ska fungera är det viktigt att koordinera de program som delar resurser.
- Saknaden av en global klocka. När program behöver samarbeta koordinerar de sina handlingar. Ett sådant samarbete kräver ofta ett gemensamt tidsbegrepp för att kunna bestämma när ett programs agerande ska inträffa. Datorer i distribuerade system kan inte synkronisera sina klockor eftersom det inte finns en allmän korrekt global klocka och detta är anledningen till att datorer i nätverk enbart kommunicerar genom att skicka meddelanden.
- Oberoende av felande komponenter. I ett distribuerat system slutar inte hela nätverket att fungera p.g.a. att en ansluten dator eller annan komponent krashar[17].

Resursdelning är en de främsta motiven till att konstruera distribuerade system. Resurser är en relativt abstrakt term och består av alla de saker som det anses användbart att dela på i ett nätverk. Resurser kan i detta sammanhang vara allt från hårdvarukomponenter, som skrivare och hårddiskar, till mjukvarukomponenter, som filer och databaser.

[17] Coulouris, Dollimore, Kindberg, 2001

En teknik för resursdelning är objektmodellen där varje efterfrågad tjänst i systemet ses som ett objekt. En skrivare ses t.ex. som ett skrivarobjekt och en hårddisk som ett lagringsobjekt. När en tjänst efterfrågas görs en förfrågan till hela systemet efter ett visst objekt. Finns tjänsten tillgänglig returneras objektet. Kan inte ett objekt utföra begärd tjänst kan objektet i sin tur fråga andra efter tjänsten. Fördelen med modellen är att en tjänst kan användas utan någon kunskap om vem som tillhandahåller den. En teknik som använder sig av modellen är CORBA, Common Object Request Broker Architechture[18].

En annan och vanligare teknik för resursdelning är client/server-modellen som bl.a. används då man hämtar och laddar ner webbsidor från en webbserver. Klientbaserad programmering bygger just på denna teknik och jag kommer därför att beskriva den mer ingående.

3.2 Client/server-modellen

Begreppet client/server härstammar ifrån 1970-talet. Datorsystemen bestod då av ”dumma” terminaler, utan egen datorkraft, som var kopplade till stordatorer, s.k. mainframes. All datorkraft och lagring av data var belägen i dessa centraliserade datorer. På 1980-talet blev utvecklingen av persondatorer och Local Area Network alltmer populära och detta bidrog till att persondatorerna snabbt integrerades i de existerande stordatorbaserade distribuerade systemen. Detta nya arrangemang ändrade radikalt den centraliserade infrastrukturen för databehandling och lagring genom att tillåta en uppdelning av programexekvering mellan stordatorer och persondatorer. På grund av stordatorernas tillförlitlighet och stora kapacitet blev dessa tillhandahållare av service, s.k. tjänster, för ett stort antal persondatorer och därav kommer uttrycket client/server[19].

Mer formellt definierat är ett client/server system en arkitektur baserad på klientsystem, system som efterfrågar tjänster, som kommunicerar över ett nätverk med serversystem, system som tillhandahåller tjänster. Termen server kan även definieras som en process eller ett program på en nätverksansluten dator som accepterar förfrågningar, s.k requests, från processer och program som körs på andra datorer i nätverket. Den dator som skickar en request kallas klient. En request skickas i form av ett meddelande till servern som i sin tur skickar ett svar, respons, tillbaka. En fullständig interaktion mellan en klient och en server där klienten skickar sin request tills den får sin respons kallas remote invocation (se figur 1)[20].

[18] http://palver.dtek.chalmers.se/groups/d1proj-2000/d1proj13/allmant.html
[19] Flores-Méndez, 1997
[20] Coulouris, Dollimore, Kindberg, 2001

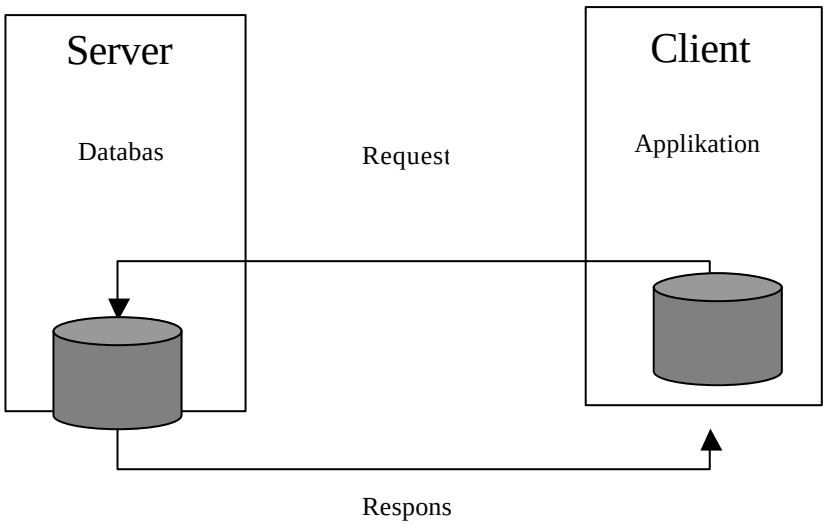

Fig.1 Client/server modellen, Remote invocation

3.2.1 Internet – ett client/server-system

Internet kan definieras som en stor sammankoppling av regionala nätverk som kommunicerar med varandra via ett gemensamt protokoll, TCP/IP. Internet är i huvudsak en kommunikationsstruktur bestående av fyra element:

- Nätverksdatorer, utgör den fysiska plattformen för bearbetning, lagring och överföring av information.
- Användare, tillhandahåller och efterfrågar information.
- Tjänster, protokoll och standards som används för att organisera, tillhandahålla och överföra information.
- Information, formaterad data med meningsfullt innehåll[21].

Internet var ett projekt som startades 1969 av det amerikanska försvarsdepartementet. Projektet gick ut på att fördela datorverksamheten på flera, geografiskt åtskilda datorer för att förhindra att en eventuell bomb skulle ödelägga hela systemet. Meddelanden som skickades mellan datorerna delades upp i paket med en adress till dess slutdestination. På detta sätt kunde paketen ta olika vägar genom nätverket för att sättas ihop till det ursprungliga meddelandet vid slutdestinationen. Mjukvaran som gjorde detta möjligt var protokollet TCP/IP, Transmission Control Protocol/Internet Protocol[22].

[21] Capron, 1999
[22] Ibid

Sedan 1969 har Internet växt från att sitt första år koppla samman 6 datorer till mer än 50 miljoner datorer 2000[23]. På grund av antalet användare och mängden information och tjänster som Internet erbjuder anses det vara det mest sofistikerade client/server-system som existerar. En av de faktorer som har bidragit till Internets snabba utveckling är implementeringen av mekanismer för att integrera olika operativsystem till en allmän kommunikationsmiljö[24]. En miljö som gjort det möjligt att överföra kunskap och information från ett stort antal källor rörande ett stort antal ämnesområde. World Wide Web är en Internettjänst som exemplifierar hur client/server-principen gör denna resursdelning möjlig.

3.2.2 World Wide Web

World Wide Web, WWW, är ett system som utvecklats för att underlätta publiceringen och tillgängligheten av resurser och tjänster på Internet. WWW utvecklades 1989, på det europeiska centret för atomforskning (CERN) i Schweiz, som ett verktyg för att utbyta dokument mellan olika forskarlag. Ett utmärkande drag för WWW är att den tillhandahåller en hypertextstruktur mellan de dokument den lagrar. Dokumenten innehåller länkar till andra dokument och denna struktur är tänkt att reflektera användarens syn på hur han/hon organiserar sin kunskap[25].

WWW är ett öppet system och kan utvidgas och implementeras på nya sätt utan att det påverkar dess existerande funktionalitet. WWW är även ett öppet system i den aspekten att olika typer av resurser kan publiceras och delas. En resurs på WWW kan idag vara allt från en enkel textsida eller bild till utförandet av en tjänst i form av elektronisk handel. Utförandet av tjänster har krävt en utveckling från statiska till dynamiska webbsidor och denna utveckling har kunnat ske utan att WWW:s basarkitektur har förändrats. WWW består av tre standardiserade tekniska komponenter:

- HTML (Hyper Text Markup Language), ett språk för att specificera innehållet och layouten på de sidor som visas i en webbläsare.
- URL (Uniform Resource Locator), identifierar dokument och andra lagrade resurser på WWW.
- En client/server-arkitektur med standardiserade regler för interaktion (HTTP, Hyper Text Transfer Protocol) genom vilka webbläsare och andra klienter hämtar resurser från webbservrar[26].

3.2.2.1 HTML

HTML är det språk som används för att utforma en webbsida. HTML är ett beskrivningsspråk som idag är starkt förknippat med WWW. HTML definierades första gången 1990 i World Wide Web-projektet på forskningscentrumet Cern i Schweiz. HTML presenterades 1993 som ett ”Internet draft”, d.v.s. ett utkast till en standard, och sedan dess har språket genomgått ett antal revideringar och förfiningar. Mycket har hänt med Internet och framförallt WWW sedan 1990 och varje

[23] Coulouris, Dollimore, Kindberg, 2001
[24] Flores-Mendéz, 1997
[25] Coulouris, Dollimore, Kindberg, 2001
[26] Ibid

version har inneburit att ett antal nya element och funktioner lagts till. Den snabba utvecklingen av WWW skulle dock i princip kräva nya HTML-versioner varje halvår och eftersom detta inte är praktiskt möjligt försöker man hitta andra tekniker och möjligheter[27]. Några exempel på nya tekniker som vuxit fram under de senaste åren är Cascading Style Sheets, CSS, och XML, EXtensive Markup Language.

3.2.2.2 URL

På Internet finns flera olika typer av adresser. Nodnamn och IP-nummer adresserar en dator, som i sin tur kan ha servrar för många olika tjänster eller protokoll. För att adressera en tjänst eller en del av en tjänst, t ex en applikation eller ett dokument, behövs en mer detaljerad adressering. URL är en typ av adressering och utvecklades för WWW, men gjordes så generell att den också går att använda till många andra tjänster på Internet.

URL står för Uniform Resource Locators, vilket kan översättas med "generella resurspekare". Standarden för hur en URL är uppbyggd betonar att det handlar om en generell pekare, inte bara en dokumentpekare.. URL är uppbyggd i flera led. Först preciseras vilket protokoll det rör sig om, därefter namnges servern (om det behövs för protokollet) och sist en söksträng till den resurs URL:en pekar på. Efter denna söksträngen kan man även ange argument[28].

Protokoll	Server	Fil (med fullständig sökväg)
http://	ww.resco.se	/katalog/index.html

Fig. 2 Exempel på en URL-adress

3.2.2.3 HTTP

HTTP protokollet en uppsättning regler som gör det möjligt att utväxla filer på WWW. I relation till TCP/IP protokollen, vilka är basen för informationsutbyte på Internet, är HTTP ett applikationsprotokoll. Det grundläggande konceptet bakom HTTP bygger på client/server-modellen med request- och responsanrop. En webbserver innehåller, förutom de webbsidor och andra filer den tillhandahåller, även en ”HTTP-daemon”. En daemon är ett program som är utformat för att invänta HTTP-requests och ta hand om dem när de anländer. En webbläsare är en HTTP-klient som sänder requests till servern. När en användare, via sin webbläsare, efterfrågar en fil genom att ange en URL-adress eller klicka på en hypertextlänk sätter webbläsaren samman en request och skickar den till rätt webbserver. Servens HTTP-daemon tar emot requesten och skickar tillbaka den efterfrågade filen[29]. Den senaste versionen av

[27] http://www.w3.org/MarkUp/
[28] http://wildesweb.com/glossary/url
[29] http://searchsystemsmanagement.techtarget.com/sDefinition/0,,sid20_gci214004,00.html

HTTP, HTTP 1.1, implementerar ihållande uppkopplingar och “pipelining” för att kunna använda en enda transportkanal för multipla request/respons interaktioner[30].

3.3 Från statiska till dynamiska webbsidor

HTML är i grunden statiskt, webbläsaren visar text, grafik m.m och väntar på att användaren ska klicka på en länk eller skicka tillbaka data till servern. Allt dynamiskt beteende är begränsat till klickning på länkar eller knappar i formulär för att efterfråga mer statisk HTML. Kravet på interaktion och utökat dynamiskt beteende har dock gått hand i hand med utvecklingen av WWW och nya tekniker och språk har utvecklats för att tillfredställa detta krav. Man kan uppnå en viss användarinteraktion genom att använda HTML-formulär och skicka data från dessa formulär till serverbaserade applikationer, CGI-program,. CGI-programmen tar hand om informationen från formuläret, utför eventuella beräkningar på informationen och skickar tillbaka en ny statisk HTML-sida. Denna teknik kallas ”server-side processing” och JSP, Java Servlets, ASP, Perl och PHP är exempel på språk eller tekniker som används för att utveckla denna typen av applikationer.

Utvecklingen av ”client-side processing” tog fart eftersom denna teknik tillför tjänster som får en webbläsare att uppföra sig mer som en vanlig skrivbordsapplikation. Klientbaserad programmering innebär också att webbläsaren kan uppträda som en klient i ett distribuerat system. Anledningarna till att man vill flytta applikationerna till webben liknar anledningarna till att införa en distribuerad arkitektur, d v s:

- Centralisera information och datatillgänglighet.
- Underlätta underhåll.
- Flyttbarhet och plattformsoberoende[31].

Skrivbordsapplikationer använder vanligtvis ett grafiskt användargränssnitt som svarar på musaktiviteter. De använder också nätverkkopplingar och lokal processorkraft. Webbläsare har under de senaste åren utvecklats för att kunna erbjuda dessa funktioner och på så sätt bemöta kravet på ökad interaktion. Tidigare webbteknik hade utvecklats utifrån principen att webbläsaren presenterar webbsidorna medan all databehandling sker på servern. Fördelarna med att flytta en del av datorkraften och behandlingen till klienten är att man underlättar utvecklingen av webbsidor som t.ex svarar på händelser, utför lokal validering under ifyllandet av formulär samt minskar nätverksbelastningen och förseningar på grund av att sidan behöver laddas om[32]. Det finns många olika tekniker för att åstadkomma klientbaserad programmering och i nästa kapitel kommer jag att beskriva dessa samt klientbaserad programmering mer ingående.

[30] http://wildesweb.com/glossary/http
[31] Barlotta, 1998
[32] Hirsch, 1997

4 Klientbaserad programmering

Klientbaserad programmering är en teknik som härstammar från distribuerade system och client/server-modellen (se kapitel 3). Följande kapitel har syftet att utförligt precisera vad begreppet klientbaserad programmering innebär, vilka sammanhang man som utvecklare kan använda sig av denna typ av programmering samt ge exempel på olika tekniker för att skapa klientbaserade applikationer.

4.1 Introduktion

Klientbaserad programmering är utformad för att utöka klientens dynamiska beteende genom att exekvera program i webbläsaren. Det finns flera anledningar till att man har utvecklat tekniker för klientbaserad programmering. En vanlig anledning kan vara behovet av att lätt verifiera att data skrivits in på rätt sätt, t.ex. att alla fält i ett HTML-formulär fyllts i eller att utöka en webbläsares kapacitet, t.ex. Shockwave för animeringar[33]. I tidigare webblösningar har serverbaserade program stått för all användarinteraktivitet. Servern producerar statiska webbsidor till webbläsaren som tolkar och presenterar dem. Denna teknik är helt serverbaserad, ”server-side processing”, och går till enligt följande: HTML innehåller enkla formulär för datainsamling, d.v.s. textrutor, radioknappar, lister m.m., vars innehåll skickas iväg till servern genom exempelvis en knapptryckning. Formulärets data passerar genom ett gränssnitt, Common Gateway Interface (CGI). Utifrån innehållet i datan bestämmer CGI vilka åtgärder som ska vidtas. Den vanligaste operationen är att starta ett program lokaliserat i serverns cgi-bin katalog. Ett CGI-program skrivs i ett programmeringsspråk som kan exekveras på serverns system. Exempel på språk är C, C++, PERL, TCL, Visual Basic, och UNIX shell script.

Microsofts alternativ till CGI är ISAPI (Internet Server Application Programming Interface) vilket är en öppen standard för att ge program tillgång till interna funktioner och egenskaper hos webbservern. ISAPI-teknologin är en del av Microsofts helhetslösning för att skapa dynamiska webbapplikationer. Microsofts Active Server Pages är en utvidgning av ISAPI och de applikationer man skapar i denna helhetslösning kallas ASP-applikationer[34].

Många webbsidor bygger idag helt på serverbaserade CGI- eller ASP-applikationer. Ett problem med serverbaserad programmering är dock responstiden. Responsen från ett serverbaserat program beror på hur mycket data som ska skickas samt på serverns och nätverkets belastning. Ett dynamiskt ritprogram är i den bemärkelsen i stort sett en omöjlighet eftersom en GIF-bild måste skapas och flyttas från servern till klienten för varje ny uppritad version. Ett annat exempel är verifiering av text i en textruta. All data i ett formulär skickas till servern som i sin tur startar ett CGI-program. CGI-programmet upptäcker ett fel, skapar en HTML-sida för att informera om felet och skickar tillbaka sidan till webbläsaren. Användaren rättar till felet och skickar på nytt innehållet i formuläret till servern. Denna teknik är långsam och belastar både servern och nätverket. Lösningen på detta problem är klientbaserad programmering [35].

[33] Wisman, 2001
[34] Fedorov, 1998
[35] Eckel, 2000

Idén bakom klientbaserad programmering är att den dator som användaren använder för att koppla upp sig mot Internet besitter mycket outnyttjad processorkraft. För att minska belastningen på servern är det fördelaktigt att dela upp processorbelastningen mellan server och klient. Mycket av de operationer CGI-programmen uträttar kan flyttas över till klienten och rent krasst behöver servern bara anropas när en webbapplikation t.ex. ska sända ett e-mail eller få tillgång till databaser och filer[36].

1

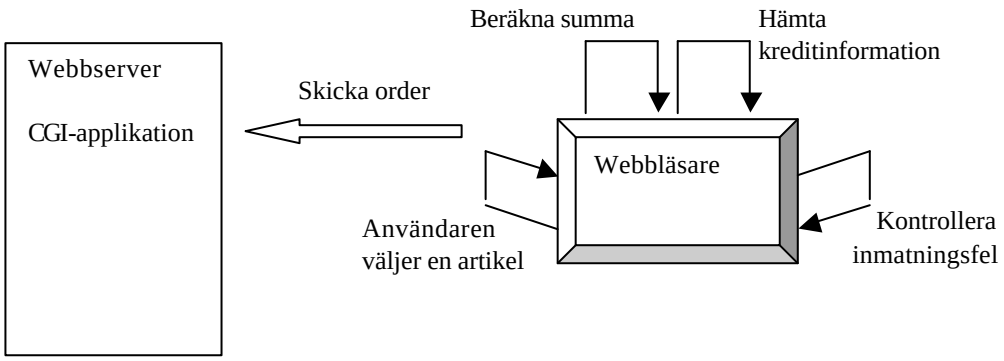

Fig. 3 Exempel på klientbaserad programmering vid e-handel.

4.2 Tekniker för klientbaserad programmering

Olika tekniker för klientbaserad programmering kan delas in i fack beroende på komplexitet, säkerhet, distribuerad datorkraft, svårighetsgrad vid installation och användning m.m. Jag har valt att använda mig av följande gruppering från Hirsch [1997]:

1 Handskas med ny typ av information
- Hjälpapplikationer
- Plugins

2 Tillhandahålla lokal datorbehandling
- HTML 3.2 Client-side features (client-side image maps, client-side stylesheets)
- Dynamisk HTML
- Script (JavaScript, VBScript)
- Java Applets

3 Integrering av objekt och applikationer
- Spyglass Software Development Interface
- ActiveX

4.2.1 Handskas med ny typ av information

[36] Sol, 2001

När en användare efterfrågar en webbsida, antingen genom att ange en URL-adress eller klicka på en länk, skickas en request till en webbserver. Webbservern tolkar requesten och returnerar sidans innehåll. Innan servern skickar tillbaka sidan bestämmer den innehållets typ och skickar denna information till HTTP Content-Type Header. Content-Type, d.v.s den typ av innehåll som webbsidan består av, definieras av MIME-typer (Multi-purpose Internet Mail Extensions) och dessa består av en typ och en subtyp. Typer delas in kategorier som text, bilder, ljud m.m och subtyper bestämmer själva formatet. Texttyper består exempelvis av text/html och text/plain, där text är själva typen och html och plain är subtypen. Webbläsare är utformade att tolka och presentera många olika typer men de kan inte klara av alla typer som existerar eller kommer att existera i framtiden. När en webbläsare får information, via HTTP Content-Type Header, om att innehållet i en efterfrågad webbsida består av en typ webbläsaren ej känner till kommer dess beteende att bero på de inställningar användaren preciserat. Om en hjälpapplikation har installerats och definierats för att ta hand om typen kommer sidans innehåll att sparas i en fil och hjälpapplikationen kommer att köras med denna fil som input. Om en plugin har definierats för typen kommer denna att anropas.

Hjälpapplikationer och plugins har många likheter. Båda är kompilerade för att köras på ett specifikt operativsystem på en viss plattform och de har full tillgänglighet till filsystem, nätverk och andra resurser. Både hjälpapplikationer och plugins måste dessutom installeras av användaren. Säkerhetsaspekten när det gäller dessa båda tekniker är minimal och kräver förtroende för säljaren av applikationen eftersom denna typ av applikationer ofta laddas ner från Internet. Vissa hjälpapplikationer och plugins kan dessutom köras utan något, för användaren, synligt fönster vilket också ökar kravet på förtroende för tillverkaren[37].

Ett exempel på en Content-Type som ej har stöd hos vissa webbläsare är Adobe PDF-formatet och för denna Content-Type finns både hjälpapplikationer och plugins definierade.

4.2.1.1 Hjälpapplikationer

En hjälpapplikation är, som nämnts ovan, byggd för att presentera och behandla en viss Content-Type. När den kompilerade hjälpapplikationen är installerad på datorn måste webbläsarens inställningar vara satta så att hjälpapplikationen anropas när den korrekta MIME-typen ska behandlas[38].

4.2.1.1 Plugins

En plugin är ett dynamiskt bibliotek, s.k API, som är installerat i webbläsarens plugin-katalog. Om en plugin är installerad i rätt katalog och ingen hjälpapplikation är definierad för den aktuella MIME-typen kommer den att anropas. En plugin är inte en självständig applikation utan är utformad för att anpassas till en specifik API. Denna

[37] Hirsch, 1997
[38] Ibid

API tillåter webbläsaren att kalla på metoder i en aktuell plugin och gör det möjligt för en plugin att få tillgång till information i webbläsaren[39].

Plugin-tekniken var ett stort steg framåt i utvecklingen av klientbaserade program. Plugins innebär en möjlighet för programmeraren att utöka webbläsarens funktionalitet genom att ladda ner en bit kod som sedan läggs in på ”rätt plats” i webbläsaren. Koden talar om för webbläsaren att den kan utföra en ny aktivitet, d.v.s ta hand om en ny Content-Type[40].

Att skriva och utveckla plugins är ingen lätt uppgift och därför är det inte något en webbutvecklare vanligtvis gör i utvecklingsprocessen av en webbsida. Betydelsen av plugins i klientbaserad programmering är dock att en programmerare kan utveckla ett nytt språk och utöka webbläsaren till att ta hand om detta språk utan tillåtelse från webbläsarens tillverkare. På detta sätt innebar plugins en bakväg till utvecklingen av nya språk och tekniker för klientbaserad programmering.

4.2.2 Tillhandahålla lokal databehandling

Lokal databehandling innebär att kod laddas ner till klienten för att exekveras på det lokala systemet.

4.2.2.1 HTML 3.2 Client-side features

I HTML 3.2 introducerade nya tekniker som flyttar mer av databehandlingen till klienten för att minska återladdningstiden av webbsidan. Exempel på dessa tekniker är Client-Side Image Map och Cascading Style Sheet.

En image map används för att associera specifika platser på en bild, s.k. hot spots, till URL-adresser. När användaren klickar på en hot spot på en bild hämtas motsvarande webbsida. Detta möjliggörs genom att de aktuella koordinaterna skickas till servern som kartlägger vilken URL det rör sig om. En Client-Side Image Map är effektivare eftersom ”mappningen” till rätt URL sker på klienten och endast den efterfrågade URL-adressen skickas till servern. Client-Side Image Maps är även enklare att skapa och underhålla eftersom både bilden och informationen om hur mappningen ska gå till sparas på klienten, inga CGI-program behöver skrivas för detta ändamål.

Cascading Style Sheet är en standard från W3C som gör det möjligt att definiera hur presentationen av en HTML-sida ska se ut. Dessa uppgifter sparas i en fil, ett s.k style sheet, för att sedan användas av de dokument som specificerar just den presentationsstilen. Cascading Style Sheets ger utvecklaren kontroll över presentationen och möjligheten att skapa en presentationsstil som sedan används av flera HTML-sidor. Tekniken ger även webbläsaren möjligheten att ladda ner ett style sheet från servern en gång och sedan spara detta i ett cash-register, vilket ökar laddningshastigheten på webbsidor med samma style sheet[41].

[39] Ibid
[40] Eckel, 2000
[41] von Pepel, 1997

4.2.2.2 Dynamisk HTML

HTML är, som jag tidigare nämnt, inte tillräckligt kraftfullt för att skapa interaktiva eller dynamiska webbsidor. Standard HTML erbjuder dessutom endast begränsad kontroll över layouten på webbsidan. För att lösa dessa problem finns i de nya webbläsarna, Navigator 4.0 från Netscape och Internet Explorer 4.0 från Microsoft, stöd för några tekniker som tillsammans möjliggör dynamisk HTML.

I de nya webbläsarna finns utökade objektmodeller, utökad händelsehantering samt en förbättrad Cascading Style Sheet-teknik med möjligheter till absolut positionering av objekt. Det finns ingen standard för händelsehantering eller objektmodeller och detta har lett till att de är implementerade på olika sätt i Netscape Navigator och Internet Explorer. Detta leder till svårigheter att utforma dynamiska HTML-sidor som accepteras av båda webbläsare.

DHTML möjliggörs av Cascading Style Sheets, en utökad objektmodell samt JavaScript. Resultatet av DHTML innebär att webbsidor kan bli mer interaktiva och innehålla multimedialiknande effekter. Med DHTML kan man åstadkomma:

- Dynamiska stilar, d.v.s. ändra färg, storlek och format på texten på en sida utan att behöva vänta på att skärmen ska uppdateras.
- Dynamiskt innehåll, d.v.s. ord, bilder och multimedia kan ändras automatiskt eller döljas/visas på en sida utan att sidan behöver laddas om.
- Dynamisk positionering. Flytta bilder, text och objekt på sidan[42].

4.2.2.3 Script

Plugin-tekniken resulterade i en explosionsartad utveckling av scriptspråk som t.ex. JavaScript. Webbläsare som stödjer klientbaserade script tillåter scriptens källkod att bäddas in i en HTML-sida och den plugin som ska tolka språket aktiveras automatiskt under tiden HTML-sidan visas. Eftersom scriptkoden är vanlig text, inbäddad i en HTML-sida, laddas de snabbt ner till webbläsaren när webbsidan efterfrågas men detta innebär även att vem som helst kan se koden till scripten. Script är dock utvecklade för att lösa specifika problem, exempelvis att skapa mer sofistikerade användargränssnitt eller utöka interaktiviteten, och innehåller inte kod som behöver skyddas av säkerhetsskäl[43]. Script är även begränsade i det avseendet att de endast kan operera inuti webbläsaren och har på så sätt inte tillgång till filsystem, nätverk mm.

Användningen av klientbaserade script gör webbsidan mindre portabel eftersom webbläsaren måste stödja språket och användaren måste möjliggöra för exekvering av script i webbläsarens inställningar. En orsak till att en användare kan välja att inte tillåta script är säkerhetsaspekten, både på grund av buggar i webbläsarens script-implementering och för att scripts kan orsaka säkerhetsproblem[44].

[42] http://www-und.ida.liu.se/~petda482/it/vad.html
[43] Eckel, 2000
[44] Hirsch, 1997

Säkerhetsaspekten när det gäller klientbaserad programmering kommer att beskrivas mer ingående i nästa avsnitt.

4.2.2.4 Java applets

Java applets var den första typen av exekverbara program som kunde laddas ner och köras i en webbläsare[45]. Den stora fördelen med program utvecklade i Java är att de är plattformsoberoende och kan exekveras på alla operativsystem som har en Java Virtual Machine. Det finns två typer av Javaprogram, vanliga skrivbordsapplikationer och webbaserade applets. Java applets är små program som bäddas in i HTML-sidor och exekveras då sidan laddas ner till i en webbläsare. En applet används för att ge webbläsaren nya funktioner t.ex. att spela spel, fungera som radio, kalkylark, grafikprogram, chatprogram mm. Applets använder, precis som plugins, en del av webbläsarens fönster.

En Java applet är en instans av en klass i Java som kallas applet. Applets exekveras i en webbläsare och har därför vissa säkerhetsbegränsningar som inte vanliga Java program har. Java applets kan inte utföra operativsystems kommandon, har inte tillgång till det lokala filsystemet och kan bara öppna nätverkskopplingar till den dator de laddats ned ifrån. Applets erbjuder mycket av kraften i en vanlig skrivbordapplikation utan något krav på installation eller säkerhetsrisk på användarsidan. Applets, inbäddade i samma webbsida, kan kommunicera med varandra men ej med andra program på den lokala maskinen[46].

Applets är en kraftfull teknik för att åstadkomma klientbaserad programmering och även om applets, av säkerhetsskäl, har omfattande funktionsbegränsningar i jämförelse med vanliga program finns det möjligheter att gå runt säkerhetsspärrarna genom bl.a. digitala signaturer. En del av syftet med denna uppsats är just att undersöka möjligheterna att få ett klientbaserat program att utföra operationer på den lokala maskinen och Java applets kommer därför att beskrivas grundligare längre fram i uppsatsen.

4.2.3 Integrering av objekt eller applikationer

Möjligheten att tillföra lokal databehandling i WWW-sammanhang genom att utöka HTML med scripts och applets är kraftfullt och möjligheten att integrera en webbklient med existerande applikationer på den lokala maskinen är i många fall något att eftersträva. Ett sätt att åstadkomma detta är att definiera en API, som vid plugins, och låta applikationer kommunicera med den. Ett annat sätt är att utöka webbläsaren till att delta i en objektmodell och komponentarkitektur.

4.2.3.1 Spyglass Software Development Interface

Ett av de tidigaste sätten att utöka en webbläsares kapacitet var att skriva applikationer bundna till den lokala maskinen webbläsaren kördes på. Dessa

[45] Lewis, Loftus, 1998
[46] Hirsch, 1997

applikationer använde sig sedan av interprocess communication (IPC) för att kontrollera och ta emot information från webbläsaren. Svårighet med denna teknik var att implementeringen av applikationerna skilde sig åt beroende på webbläsare och plattform.

Spyglass Software Development Interface (SSDI) är en specifikation som definierar hur en applikation kan koppla upp sig mot en webbläsare för att få meddelande om events och i sin tur skicka events till webbläsaren för att få den att utföra olika operationer. Ett exempel på ett SSDI-gränssnitt är när en applikation registrerar sig hos en webbläsare för att få meddelande om varje URL som webbläsaren anropar samt en kopia av alla de webbsidor webbläsaren laddar ner. Nackdelarna med SSDI är, precis som vid hjälpapplikationer, att tekniken kräver att användaren har tillit att installera en maskinberoende applikation på den lokala maskinen[47].

4.2.3.2 ActiveX

ActiveX är namnet på en uppsättning teknologier från Microsoft som inkluderar en sammansatt dokument-, komponent- och objektmodell. Komponenter kan skicka meddelanden till varandra och utföra ”remote method invocation”. ActiveX inkluderar all Internetrelaterad teknik från Microsoft och är ingen egentlig produkt. ActiveX tekniken täcker all teknologi som programmerare kan använda för att bygga webbapplikationer och de komponenter, byggstenar, man använder för att skapa detta kallas ActiveX-kontroller[48].
ActiveX är en vidareutveckling av den tidigare OLE-tekniken, Object Linking and Embedding. OLE är Microsofts standard för att få olika program att samverka, dokumentinriktad integration av tillämpningar[49].

ActiveX kontroller beskrivs ofta som Microsofts svar på Java applets och i båda teknikerna kan man, på olika sätt, få tillgång till resurser på klientens lokala dator. Nästa kapitel kommer att beskriva både ActiveX och Java applets mer ingående samt göra en jämförelse mellan teknikerna utifrån aspekterna plattformsoberoende, säkerhet och funktionalitet.

[47] Ibid
[48] http://www.active-x.com/articles/whatis.htm
[49] http://www.microsoft.com/com/tech/ActiveX.asp

5 Resultat

Kapitel 3 och 4 har haft syftet att ge en bakgrundbild till klientbaserad programmering samt beskriva några användningsområden och tekniker för denna typ av programmering. Java applets och ActiveX är två tekniker som använder sig av olika angreppssätt för att åstadkomma klientbaserade applikationer. Jag har valt att redovisa jämförelsen mellan dessa tekniker som en del av undersökningens resultat. Undersökningens resultat består således av två delar, där första delen redovisar denna jämförelse och andra delen beskriver tillvägagångssätt och resultat av den praktiska utvecklingen av den klientbaserade applikationen FileApplet.

5.1 Jämförelse mellan Java applets och ActiveX

Jämförelsen av Java applet och ActiveX har gjorts utifrån aspekterna plattformsoberoende, säkerhet och funktionalitet. Inledningsvis beskrivs dessa teknikerna mer ingående för att sedan fokuseras på ovanstående aspekter. Först beskrivs hur respektive teknik fungerar och sedan redovisas åsikter från olika artiklar och rapporter. Valet av artiklar och rapporter har gjorts med en ansats att försöka skapa en balans mellan förespråkare från båda läger samt dokument som, enligt mig, gör en objektiv bedömning och jämförelse av de båda teknikerna.. Resultatet av denna jämförelse kommer sedan att utvärderas i en avslutande diskussion.

5.1.1 ActiveX

ActiveX är namnet på en teknologi som är utformad för att öka interaktiviteten och flexibiliteten på webbsidor. Teknologin består av tre typer av komponenter, ActiveX-kontroller, ActiveX-dokument och ActiveX-scripts. ActiveX-kontroller är små kodmoduler som kan köras inuti webbsidor för att öka funktionaliteten. Från användarsynpunkt fungerar ActiveX-kontroller på samma sätt som plugins men skillnaden är att de laddas dynamiskt. ActiveX-kontroller kan användas till att skapa interaktiva menyer, klockor, kalendrar, kalkyler m.m.[50]. ActiveX-dokument gör det möjligt för användare att läsa dokument som ej är skrivna i HTML, t.ex. Word- och Excel-dokument, i en webbläsare. Genom att använda ett programmeringsspråk, t ex Visual Basic, kan man även skapa hela applikationer som utåt sett liknar och uppför sig som ett vanligt dokument. ActiveX-script är de språk som kan användas för att skapa ActiveX-kontroller och dokument, t ex Visual Basic, C++ och Java. Det enda kravet på dessa språk är att de stödjer OLE-tekniken[51].

ActiveX-teknologin härstammar ifrån slutet av 1980-talet och har fått genom gå flera namnbyten sedan dess. ActiveX baseras på två andra Microsoft teknologier, COM (Component Object Model) och OLE (Object Linking and Embedding). Com och OLE används för att förenkla skapandet av skrivbordapplikationer och öka dessa applikationers funktionalitet Microsoft hade en vision att skapa dokumentfokuserad datorbehandling vilket skulle ge användaren möjlighet att manipulera och presentera olika typer av data. För att realisera denna vision skapade Microsoft OLE-teknologin,

[50] www.ncompasslabs.com/scriptactive
[51] www.microsoft.com/activex/index.html

vilket förenklade skapandet av sammansatta dokument. Sammansatta dokument är datafiler som är skapade antingen genom att sammanlänka flera olika datafiler (förändras data i de länkade filerna förändras data automatiskt i det sammansatta dokumentet) eller genom att kopiera in innehållet från en datafil till en annan fil (förändringar av data i den inbäddade filen påverkar ej data i det sammansatta dokumentet). När en användare tittar på ett sammansatt dokument uppfattar han/hon den som en helhet. Ett exempel på ett sammansatt dokument är ett dokument som innehåller en kalkylfil, skapad i Excel, inbäddad i ett textdokument, skapad i Word.

Den första versionen av OLE tillät enbart presentation av sammansatta dokument som en helhet. För att modifiera länkade eller inbäddade filer var man tvungen att avsluta den aktuella applikationen och öppna den applikation som filen var skapad i. Detta var besvärligt att hantera och man insåg att det var bättre att klicka på den del av dokumentet som innehöll den länkade eller inbäddade filen. Microsoft kallade denna funktion ”in-place activation” och presenterades i den andra versionen av OLE, vilken använde COM.

COM skapades ursprungligen för att göra ”in-place activation” möjlig och är ett objektorienterat protokoll för binär körbar kod. Den är ett ramverk för att skapa mjukvarukomponenter som kan anropas och köras. Protokollet specificerar en standard för data och metoder, i instansierade COM-objekt, att bli tillgängliga under programkörning. Protokollet är tillräckligt abstrakt för att tillåta komponenter (objekt) i ett sammansatt dokument att skapas oberoende av varandra och få information om varandra under programkörning. COM kan ses som en robust mekanism för återanvändning av kod som ej är knuten till ett specifikt programspråk[52].

Den andra versionen av OLE använde i huvudsak COM för att hantera och kommunicera med objekt i ett sammansatt dokument. Ett sammansatt dokument i OLE lagrade inte bara data utan även en notering om vilken körbar kod som kunde manipulera den. Denna information lagrades i ett standard formulär vilket gjorde det möjligt för andra COM-objekt att läsa, tolka och anropa den. COM är även användbart i ett mer generellt sammanhang än vid sammansatta dokument. COM tillåter Windows-applikationer att kommunicera med andra applikationer och objekt på samma maskin, t.ex. kan funktionerna i ett COM-baserat Excel-program användas av andra program. COM kan därför ses som en ”inter-process” kommunikation mellan Windows-objekt[53].

I Windows 95 introducerades en ny standard, OCX, vilket står för OLE Custom eXtension. OCX är en bit COM-baserad kod, vanligtvis mindre än en hel applikation, som även innehållet ett eget användargränssnitt. Idén bakom OCX är att alla, oavsett programmeringskunskaper, ska kunna kombinera flera OCX-kontroller och på så sätt skapa skräddarsydda applikationer.

Microsofts Internetstrategi tillkännagavs 1996 och gick ut på att teknologierna, COM och OLE, även skulle användas för nätverkskopplingar och Internet, och detta är grunden till ActiveX. ActiveX är Microsofts namn för den teknologi som tillför exekverbar kod till en webbsida. Den tillåter sammansatta dokument att själva anropa mjukvara för att presentera data och tillför en standardmekanism för

[52] Stanek
[53] http://www.microsoft.com/com/presentations/default.asp

nedladdning och körning av godtyckliga filer oavsett om de är belägna på den lokala datorn eller ej. ActiveX förhållande till COM och OLE är att COM-komponenter är det format som vanligtvis laddas ned över Internet och nästan alla dessa komponenter stödjer OLE-gränssnitten. Grundelementen i ActiveX-teknologin är ActiveX-kontroller som är en förenklad version av en OCX-kontroll.[54]

Eftersom COM enbart var en skrivbordslösning var Microsoft tvungen att utöka denna teknologi för att möjliggöra för nätverkssupport. Microsoft skapade därför Distributed COM, DCOM, vilket har stöd för att både lokala och avlägsna objekt och applikationer ska bli tillgängliga på samma sätt som i COM. DCOM använder ”remote procedure” anrop samt systemregister på den lokala maskinen, innehållande information om distribuerade objekt, för att skapa denna illusion.

Microsofts ovan nämnda Internetstrategi består av två viktiga element:

- Webbapplikationer, med en funktionalitet som påminner om Java applets applikationer, kan skapas på ett sätt som liknar det sätt som vanliga Windows-applikationer utformas på. Detta gör det lättare för utvecklare, som redan är vana vid miljön i Windows-applikationer, att skapa webbapplikationer.
- Windows-applikationer som är baserade på COM och OLE kan lätt modifieras till webbapplikationer med hjälp av ActiveX. En version av Microsoft Word som kan öppna och spara filer, lokaliserade på flera olika maskiner i ett nätverk, kan exempelvis lätt utvecklas genom små modifikationer av den skrivbordsbundna versionen.

Microsofts vision består i att vilken applikation som helst ska, på ett enkelt sätt, kunna komma åt och dra fördel av nätverket och Internet på ett standardiserat sätt. Utöver detta ska skapandet av en webbapplikation kunna utföras på ett snabbare sätt på grund av de likheter som finns mellan COM/OLE och ActiveX samt den återanvändning av kod som COM erbjuder[55].

5.1.2 Java applets

Java är ett plattformsoberoende objektorienterat programmeringsspråk utvecklat av Sun Microsystems. Sun Microsystem har sedan starten arbetat utifrån övertygelsen att datorer utnyttjas mest effektivt om de är förbundna med varandra i nätverk[56]. Datorer är inte de enda apparater som kan kopplas ihop i nätverk och kommunicera med varandra. Sun började under 1980-talet skissa på sätt att programvarumässigt styra hemelektronik t ex videobandspelare och brödrostar. Det visade sig snart att de redan existerande programmeringsspråken inte lämpade sig för program som skulle fungera över nätverk och inte bara lokalt på stationära datorer. Sun började därför, 1990, skräddarsy ett nytt språk för detta ändamål. Det nya programmeringsspråket fick namnet Java och utvecklades med syftet att lösa problemet med plattformsberoende. Man ville skapa portabla program som kunde köras på vilket processor och i vilken miljö som helst[57]. Under 1994/95 ökade intresset för Internet explosionsartat och kravet på att olika datorer skulle kommunicera med varandra

[54] Stanek
[55] Smith, 1997
[56] http://java.sun.com/features/1998/05/birthday.html
[57] http://www.internet.physto.se/fil/java.html

över ett nätverk uppstod. Java visade sig vara ett mycket lämpligt språk för att lösa denna uppgift.

Sun fick viktig draghjälp av webbläsartillverkaren Netscape Communications, som var ett av de första företagen att licensiera Java från Sun. Fram till dess hade det endast funnits en webbläsare med Java-kapacitet, Suns egna testprogram HotJava. Netscape lanserade Netscape Navigator 2.0 som den första riktiga Java-kapabla webbläsaren.

Javas systerspråk C++ är ett kompilerat programmeringsspråk, d.v.s. en programmerare skriver programkod som översätts till maskinkod av en kompilator. Det behövs två versioner av samma program eftersom programmeraren bara förstår C++-kod och datorn bara maskinkod, och därför behövs översättningen mellan de två versionerna. Olika datorer förstår olika sorters maskinkod: ett C++program kompilerat för Macintosh PPC-processor kan inte köras på en PC med Intel-processor. Ett program måste alltså skrivas och kompileras om för varje plattform det är tänkt att köras på.

Java fungerar däremot annorlunda. En Java-kompilator översätter inte programkoden till olika sorters processorbundna maskinkoder utan till något som kallas bytekod. Bytekod ser ut som maskinkod, men kan inte tolkas av en processor. Vid exekveringen av ett Java-program behövs en Java Virtual Machine (JVM). JVM är ett program som omtolkar bytekoden till den maskinkod datorns processor förstår. Man kan säga att Java både är kompilerat (från programkod till bytekod; på programmerarens dator) och interpreterat (från bytekod till maskinkod, på användarens dator)[58].

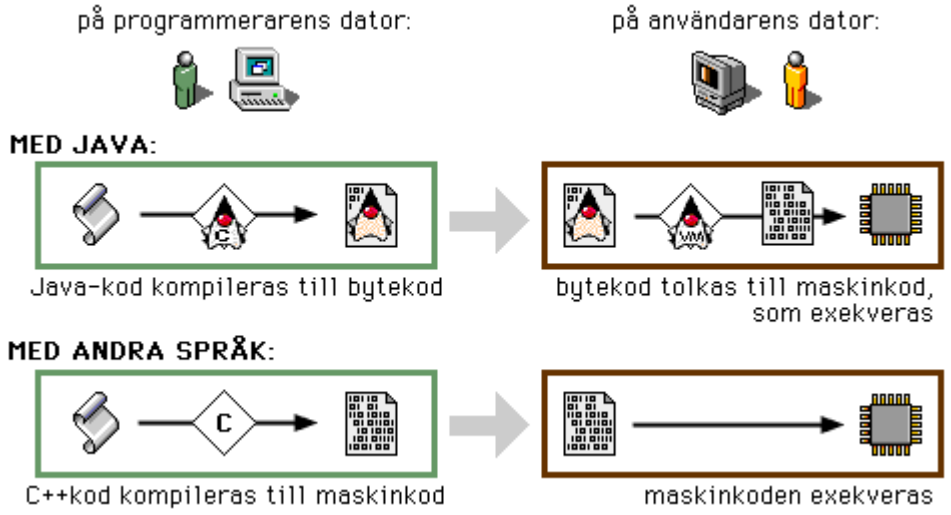

Fig. 4 Java Virtual Machine (från http://internet.physto.se/fil/java.html)

JVM kan vara fristående program (t.ex. Suns Java Applet Viewer), vara inbyggda i webbläsare (t.ex. Netscape Navigator, Internet Explorer) eller vara inbyggda i datorns

[58] Lewis, Loftus, 1998

operativsystem. De kan också ersätta datorns operativsystemet helt och hållet. Ett exempel på detta är JavaOS i de s.k. NC, nätverksdatorerna, vilka saknar möjlighet att lagra data. Dessa datorer hämtar program och data från servrar via nätverket allt eftersom de behövs[59].

Det finns olika typer av Java-program, t.ex. applets, servlets och skrivbordsapplikationer. Java servlets körs på servern och ger en plattformsoberoende miljö för serverapplikationer, oftast i samband med en Java-baserad webbserver. Vanliga skrivbordapplikationer är applikationer som körs på en lokal dator och kommunicerar med operativsystemet och andra program. Applets är de program som körs inuti en webbläsare och är därför Javas teknik för att åstadkomma klientbaserad programmering. Applet-klassen lagras på en webbserver för att sedan laddas ned till användaren när en webbsidan efterfrågas. Webbläsaren startar sin inbyggda JVM som interpreterar bytekoden och visar resultatet i webbläsarens fönster. Applets kan, som ActiveX-kontroller, användas för att skapa klockor, menyer, animeringar m.m. Applets användes i början just för att utöka webbsidan med olika finesser men tekniken har utvecklats och kan idag användas för att skapa komplexa distribuerade system.

Java har som programmeringsspråk utvecklats mycket under de senaste åren. Man har utökat de grundläggande klasserna i Java för att förenkla distribuerad och klientbaserad programmering. Nedan följer några av de tekniker för distribuerad programmering som det finns stöd för i Java:

- Hämta och skicka information mellan två datorer i ett nätverk: Javas net-klass.
- Databaskopplingar över ett nätverk: Java DataBase Connectivity (JDBC).
- Utföra tjänster via en webbserver: Java Servlets och Java Server Pages (JSP).
- Anropa och exekvera metoder i Java-objekt som är lokaliserade på andra maskiner: Remote Method Invocation (RMI).
- Använda kod som är skriven i andra programmeringsspråk och som körs i andra arkitekturer: Javas inbyggda stöd för Common Object Request Broker Architecture (CORBA).
- Isolera affärslogik från anslutnings- och uppkopplingsdetaljer, som transaktioner och säkerhetsrestriktioner vid databasuppkopplingar: Enterprise JavaBeans (EJB).
- Dynamiskt lägga till och ta bort nätverksansluten utrustning: Javas Jini[60].

5.1.3 Aspekter för utveckling av klientbaserad applikationer

ActiveX och Java applets använder sig av två skilda angreppssätt för att åstadkomma klientbaserade applikationer. Enligt Kotz & Gray [1999] finns det tre tekniska hinder som måste övervinnas för att klientbaserade applikationer ska kunna användas på Internet. Det är därför dessa tre kriterier som jag har valt att fokusera på i min jämförelse mellan de båda teknikerna:

[59] http://www.internet.physto.se/fil/java.html
[60] Eckel, 2000

- Plattformsoberoende: Den nedladdningsbara applikationen bör kunna exekveras på flera olika operativsystem och hårdvaruplattformar eftersom koden överförs via ett nätverk till ett stort antal användare.
- Säkerhet: Systemet bör garantera kodens integritet under överföringen, utföra en verifiering av både klient och server samt tillhandahålla metoder för att förhindra att virus överförs via den klientbaserade koden.
- Funktionalitet: Nedladdningsbara applikationer bör vara relativt små och innehålla återanvändbar kod eftersom de, av hänsyn till plattformsoberoende och säkerhet, ofta är skrivna i ett relativt långsamt interpreterat språk[61].

5.1.4 Plattformsoberoende

I en jämförelse mellan ActiveX och Java applets är plattformsoberoende en central aspekt. ActiveX är en Microsoftprodukt och därför hårt bunden till Windows operativsystem. ActiveX-kontroller kan idag utvecklas i vilket programmeringsspråk som helst, så länge språket stödjer OLE. Dessa språk kräver dock icke-portabla utbyggnader som begränsar deras användning till Windows operativsystem. För att övervinna dessa plattformsbegränsningar har man implementerat OLE/COM på Macintosh och Unix plattformar för att på så sätt göra ActiveX-kontroller tillgängliga. Överföringen av OLE/COM till olika operativsystem garanterar dock inte automatisk portabilitet när det gäller ActiveX-kontroller eftersom de fortfarande kommer att vara bundna till den plattform de utvecklas på. Om man har utvecklat en kontroll på en Unix-dator är den bunden till denna plattform. Ett sätt att lösa detta problem är att ha olika versioner av kontrollerna för varje operativsystem. Webbläsare behöver då identifiera dessa versioner och ladda ned den version som passar klientens plattform. Nackdelen med denna lösning är det tidsödande arbetet med att utveckla olika versioner för varje kontroll[62].

Microsoft hade, vid utvecklingen av ActiveX, målet att införa en ny Internetstandard och koncentrerade inte utvecklingen på plattformsoberoende. Idag använder man Java för att utveckla portabla kontroller i ActiveX. ActiveX-kontroller kräver dock plattformsspecifika anrop som är unika i olika miljöer, även om de är skrivna i Java[63].

Till skillnad från Microsoft utgick Sun Microsystems från plattformsoberoende när de tog fram Java och en stor fördel med att använda Java applets är just portabiliteten. Det finns två olika typer av portabilitet, källkods portabilitet och binärkods portabilitet.

- Källkods portabilitet innebär att koden kompileras och översätts till någon form av plattformsoberoende representation, t.ex. Java bytekod, för att efter överföringen kompileras till maskinkod eller exekveras inuti en interpreter.
- Binärkods portabilitet innebär möjligheten att flytta en körbar kopia av ett program från en maskin till en annan[64].

Java erbjuder källkods portabilitet eftersom koden kompileras och översätts till någon form av plattformsoberoende representation. Normalt när man skriver och kompilerar

[61] Kotz, Gray, 1999
[62] Flores-Mendez, 1997
[63] http://sunsite.ics.forth.gr/javabeans/faq/faq.interop.html
[64] Flores-Mendez, 1997

applikationer konverteras källkoden till maskinkod som bara kan tolkas av samma operativsystem som det kompilerades på. Java-applikationer kan dock köras på flera operativsystem eftersom de konverteras till bytekod som sedan kan tolkas av en JVM. De största webbläsarna Internet Explorer och Netscape Navigator innehåller idag båda en JVM[65].

Java är alltså utformat för att klara av plattformsoberoende och applikationer, utvecklade i Java, kan köras på ett stort antal processorer och operativsystem utan att modifieras. JVM är den applikation som möjliggör exekveringen av applikationer på ett plattformsoberoende sätt. De funktioner som gör detta möjligt finns i ett standardbibliotek och detta bibliotek måste underhållas på varje plattform för att portabiliteten ska fungera[66].

De artiklar och rapporter som jag valt ut gör alla en jämförelse eller utvärdering av teknikerna och genomgående i dessa dokument är ActiveXs begränsningar när det gäller plattformsoberoende. ActiveX utvecklades av Microsoft för att enkelt kunna integreras i Windows-applikationer och är därför främst utformad för Windows och webbläsaren Internet Explorer. ActiveX har inte stöd i webbläsaren Netscape och Pollán [1998] beskriver ActiveXs begränsningar enligt följande:

"*Due to market wars. Netscape browsers do not run ActiveX controls ../.. they are designed for the Windows world, which makes their portability quite limited*."

Försök att implementera ActiveX-tekniken på andra plattformar, t.ex. Unix och Macintosh, har gjorts med enligt många av författarna är det svårt att få dessa implementeringar att fungera på ett tillfredsställande sätt. Richard Hoffman [1999] skriver såhär:

"../..Even if ActiveX controls are available on some non-Windows platforms, such as Apple Computer´s Macintosh, Java has much wider client and OS support ../.. ActiveX is not supported on Netscape browsers without the use of plugins."

Java är däremot ett plattformsoberoende programmeringsspråk och Java applets kan köras på en rad olika operativsystem. Java har dessutom stöd i båda webbläsarna Nescape och InternetExplorer. Wilders [2000] drar därför slutsatsen att Java är den enda valmöjligheten när det gäller att utveckla program för Internet:

"*If you have an heterogenus environment or are out on the Internet serving many platforms, I don´t think you have any choise but Java*."

Trots Microsofts försök att göra ActiveX plattformsoberoende är tekniken utformad och anpassad till Windowsmiljön. Om man vill använda sig av ActiveX för att skapa Internet-applikationer måste man vara säker på att alla klienter som laddar ner kontrollen använder sig av Internet Explorer och sitter i en Windowsmiljö. Detta gör ActiveX, i jämförelse med Java applets, till ett mindre fördelaktigt val. Om de klienter man vänder sig till alla är placerade i ett intranät, t.ex. inom ett företag, och man kan säkerställa att alla arbetar i Windows och Internet Explorer är inte ActiveX en sämre teknik. Microsofts Product Manager Greg Leake säger i Wilders [2000]:

[65] http://www.abdn.ac.uk/diss/webpack/chap29.hti
[66] http://www.sims.berkeley.edu/courses/is206/f97/GroupC/java.html

"*It´s not a choise you have to make. ActiveX is better for some things and Java for others. ../.. For Intranets and Microsoft environment, ActiveX is not a bad business decision.*"

Flera författare håller med om detta påstående och även i den tekniska rapporten "*White paper on ActiveX*" från Microsoft [2001] dras denna slutsats :

"*ActiveX is clearly a Microsoft-dominated standard that does not enjoy Java´s wide availability…/.. ActiveX´s availability to integrate the multiple host connectivity is more straightforward than Java../.. This is especially true in the case of Intranets where there are more formal and documented expectations for web-to-host integration*."

Författarna är i stort sett eniga om att ActiveX, i jämförelse med Java och Java applets, passar bättre vid utvecklingen av intranätbaserade webbapplikationer till följd av teknikens brister i plattformsoberoende.

5.1.5 Säkerhet

Klientbaserad programmering är en viktig säkerhetsfråga eftersom tekniken innebär att exekverbar kod laddar ner till den lokala datorn. Klientbaserad kod används, som jag tidigare nämnt, för att tillföra funktioner till en webbsida och på så sätt göra den mer dynamisk. Denna kod är placerad inuti HTML-kod och exekveringen av koden sker ofta utan att användaren är medveten om det. Dåliga säkerhetsresurser i persondatorernas operativsystem förstorar problemen eftersom ett program som körs på ett sådant operativsystem i stort sett har obegränsad tillgång till datorns resurser[67]. Användningen av klientbaserad kod möjliggör för ökad funktionalitet och användarinteraktion men den skapar även problem. Tilliten till klientbaserad kod skulle försvinna helt om inte säkerhetsmekanismer hade utvecklats för att isolera och kontrollera effekterna av koden. Dessa säkerhetsmekanismer kan antingen definieras till komponenter som vid nedladdningen exakt tar reda på varifrån den mobila koden kommer och på så sätt utför verifiering av koden. Mekanismerna kan även knytas till den miljö där den mobila koden ska exekveras[68]. ActiveX och Java löser säkerhetsproblemen på olika sätt.

Microsoft har baserat ActiveX:s säkerhetspolicy på digitala signaturer och kryptering. Signering innebär en form av krypteringsteknik i vilken en godtycklig binär fil kan "signeras" av en utvecklare. En digital signatur har vissa matematiska egenskaper som gör att är de svåra att forcera och förfalska. ActiveX-kontroller är binära komponenter som exekveras drekt på maskinen. När en komponent körs har den, med andra ord, tillgång till hela klientens datorsystem utan några restriktioner. För att minska den säkerhetsrisk som detta innebär har Microsoft utvecklat Windows Trust Verification Service. Denna tjänst låter användaren indikera om den har förtroende för komponenten eller ej, d.v.s. om källan till koden är betrodd[69]. Windows Trust Verification Service verifierar signaturen och talar om för användaren var koden kommer ifrån och frågar om koden ska exekveras eller ej.

[67] Committee in Information Systems Trustworthiness, 1999
[68] Ibid
[69] Kivi, Wong, 2000

Microsoft teknik för att verifiera digitalt signerad kod förlitar sig på att användaren kan bestämma om koden ska exekveras utifrån kunskapen om var koden kommer ifrån. Detta antagande kan ifrågasättas eftersom det har visat sig att många användare inte bryr sig om att titta på signaturen eller inte har den kunskap som krävs för att fatta rätt beslut. En annan svårighet med denna teknik rör själva härstamningen, d.v.s. att kontrollen verkligen är signerad av den källa som påstås[70].

Säkerhetsaspekterna när det gäller ActiveX kan delas in i två områden:
1, aspekter relaterade till nedladdning och installation av kontrollerna.
2, aspekter relaterade till exekvering av kontrollerna.

Beslut att ladda ner och installera en mjukvara baseras på mjukvarans duglighet. När det gäller ActiveX-kontroller finns det inget bra sätt att utföra en sådan kontroll utan beslutet måste, som tidigare beskrivits, baseras på antagande om kodens källa. Det största problemet med signering är att säkra kontroller kan komma ifrån otillförlitliga källor och osäkra eller fientliga kontroller kan komma från tillförlitliga källor. Ett annat problem är att en kontroll bara behöver registreras en gång per maskin i den nuvarande Windows arkitekturen. Om en användare laddar ner kontrollen är den tillgänglig för alla användare som utnyttjar maskinen. Övriga användare får alltså inte någon förfrågan om de önskar ladda ner och exekvera kontrollen. Detta innebär en säkerhetsrisk om kontrollen är av fientlig karaktär, t.ex. är utformad för att radera filer på den lokala hårddisken.

När det gäller exekvering har ActiveX-kontroller större kapacitet än komponenter som enbart körs i en s.k. ”sandlåda”. Java använder sig av sandlådeprincipen för att öka säkerheten vid exekveringen av applets (se nedan). Eftersom ActiveX–kontroller är skrivna i maskinkod kan de köras direkt på den fysiska maskinen och kontrollerna får på så sätt tillgång till tjänster och resurser som ej är tillgänglig för kod som körs i en begränsad miljö[71].

Sun har byggt upp en gedigen säkerhetsbarriär för Java applets. För att förhindra att applets får otillbörlig tillgång till en klients lokala system har man isolerat dem till att exekveras och verka i en sandlåda. Idén med Javas sandlåda är att man ska kunna exekvera otillförlitlig kod på den lokala maskinen utan att oroa sig över vad den kan åstadkomma. Javas sandlåda är uppbyggd av tre olika delar; bytekodverifieraren, klassladdaren och säkerhetshanteraren. Dessa tre komponenter utför laddnings- och exekveringskontroller för att begränsa tillgängligheten till filsystem, nätverk m.m.

Bytekodverifieraren är den första komponenten i Javas säkerhetsmodell. När ett javaprogram kompileras får man plattformsoberoende bytekod, denna bytekod körs sedan på den lokala maskinen. Eftersom det finns källkod för att skapa egna javakompilatorer tillgänglig så kan man inte förlita sig på att den bytekod som skapas man är tillförlitlig. För att komma till rätta med detta problem utförs en kodsverifiering. Alla programobjekt i Java tillhör klasser och en klassladdare används för kontrollera laddningen av dessa klasser under exekveringen av ett program. Appletklassladdaren bestämmer t.ex. när och hur en applet kan lägga till eller anropa klasser under exekvering. Säkerhetshanterarens kontrollerar, under exekveringen, att

[70] Committee in Information Systems Trustworthiness, 1999
[71] Cert, 2000

inga "farliga" operationer utförs, d.v.s. operationer som kräver filhantering, nätverkstillgänglighet eller operationer som försöker skapa nya klassladdare. Säkerhetshanterarens uppgift är, med andra ord, att vakta gränserna på sandlådan[72].

I denna uppsats är ett av målen att undersöka och utforma en applet som skriver ner till en klient lokala filsystem, vilket kräver att appleten opererar utanför sandlådan. Detta möjliggörs genom att appleten signeras på ett sätt som liknar Microsofts signering av ActiveX-kontroller. Till skillnad från ActiveX:s signering, där en kontroll antingen är betrodd eller ej, har Sun utvecklat en princip med säkerhetspolicys. Säkerhetspolicyn innebär att man kan specificera exakt vad en applikation ska få tillgång till. En systemadministratör kan t.ex. signera en applikation och ge den tillgång till ett antal godtyckliga filer i ett system dock ej hela systemet. När kodsignering kombineras med säkerhetspolicys kan en applets gradvis ta sig ut ur sandlådan (se figur 4).

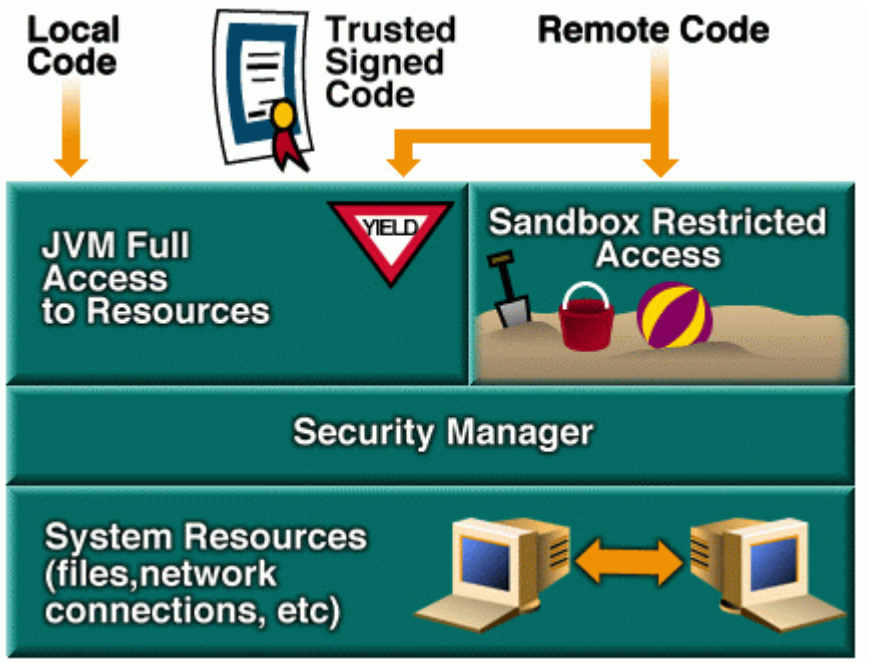

fig.4 Sandlådemodellen (från java.sun.com)

ActiveX och Java använder sig, som beskrivits ovan, av olika tekniker för att upprätthålla säkerheten för nedladdningsbar kod. ActiveX-kontroller är skrivna i maskinspecifik kod som exekveras direkt på klientens operativsystem. Detta leder till att kontrollerna får tillgång till klientens lokala resurser. Java laddas ned till klienten i form av bytekod och exekveras därefter i webbläsarens inbyggda JVM. Detta gör det möjligt att isolera applikationens tillgänglighet till det övriga systemet, vilket Hodson [1998] beskriver enligt följande:

”*Java runs in a ”sandbox” which gives the applet restricted access to resources on the machine it´s running on*.”

[72] McGraw, Felten, 1999

ActiveX-kontroller kan, i stort sett, utföra vilka operationer som helst på klientens dator vilket innebär att de kan radera filer, hårddiskar m.m. Signeringen av kontrollerna förhindrar inte dem från att utföra attacker på det lokala systemet, vilket Murawski [1997] belyser:

”*A signature does not guarantee the code is safe to run. It only identifies the entity who registered the control and assures that the original code is untampered*.”

Java använder sig av sandlådemodellen när det gäller osignerade applets och signering samt säkerhetspolicys för att tillåta appleten att ta sig ur sandlådan. Denna metod anses av många författare som säkrare vid utvecklingen av webbappliaktioner. Ett av syftena med denna uppsats är att låta en klientbaserad applet ta sig ut sandlådan för att utföra filöverföringar mellan klient och server. ActiveX-kontroller har redan tillgång till dessa resurser, dock på ett för klienten riskfyllt sätt. I en intranätmiljö är kraven på säkerhet inte lika omfattande och det anses, av många författare, vara ytterligare en anledning till att ActiveX passar för att användas i denna miljö. Murawski [1997]skriver så här:

”*ActiveX is a technology that makes sense only within a trusted environment. As a result, it’s a good choice for an intranet where all web pages are written by known sources within an enterprise*.”

Shoffner [1999] förstärker denna syn genom följande påstående:

”*The ActiveX security model has received much criticism with respect to its fitness for Internet use, but these concerns do not apply to the same degree on the intranet. As a result of these factors, Microsoft is pushing ActiveX in the intranet market*.”

5.1.6 Funktionalitet

Funktionalitet när det gäller olika programmeringsspråk och tekniker är kopplat till kapaciteten att producera kod som uppfyller användarens förväntningar. När man utvärderar en tekniks funktionalitet kan man undersöka följande områden:

- **Dataprocessing**: innefattar egenskaper som objektmodell, datatyper, stöd för trådning, feltolerans och minneshantering.
- **Användargränssnitt**: omfattar funktioners möjligheter att skapa och manipulera vanliga användargränssnittskomponenter.
- **Filsystem**: utvärderar möjligheten att komma åt filstrukturer och deras innehåll.
- **Nätverkskopplingar**: analyserar möjligheten för funktioner att bygga kommunikationsstrukturer mellan lokala och avlägsna datorer. Vanliga protokoll och tekniker att stödja är sockets, datagram och HTTP kopplingar[73].

Eftersom ActiveX inte är ett programmeringsspråk utan en teknologi för att skapa exekverbara objekt är det svårt att mäta funktionaliteten på denna teknik. Funktionaliteten hos ActiveX-kontroller beror till största delen på funktionaliteten i de programmeringsspråk som den specifika kontrollen är skriven i, t ex Visual Basic,

[73] Flores-Mendez, 1997

C++, Java m.m. När en ActiveX-kontroll är signerad och verifierad som tillförlitlig har den obegränsad tillgång till det system de exekveras på och inga funktionsbegränsningar uppstår för det valda programmeringsspråket.

Java är, till skillnad från ActiveX, ett programmeringsspråk och är uppbyggt kring fördefinierade klasser, Javas Application Programming Interface, från vilka instansieringar, arv m.m genomförs. Ett Application Programming Interface (API) är en specifikation för hur ett programobjekt skall kommunicera med ett annat. För att utvärdera Javas och i synnerhet applets funktionalitet utifrån ovanstående områden måste man dels titta på de klasser som finns tillgängliga i API:n och de operationer de klara av att utföra. Det finns i Java stöd för alla ovan nämnda områden.

En funktionsrelaterad aspekt som många författare tar upp, i jämförelser mellan ActiveX och Java applets, är applikationernas snabbhet. ActiveX är snabbare än Java när det gäller nedladdning och exekvering av kod. Java applets exekveras långsammare eftersom det är ett tolkat, interpreterat, språk. Koden är inte kompilerad och färdig att exekveras utan måste först tolkas av JVM. ActiveX-kontroller är däremot kompilerade komponenter som är redo är exekveras direkt vid nedladdningen. Applets laddas dessutom ned varje gång klienten anropar webbsidan medan en ActiveX-kontroll endast laddas ned en gång vilket gör ActiveX snabbare. Murawski [1997] beskriver detta så här:

”*Java byte code is more compact than ActiveX so its download faster. But, ActiveX controls only have to be downloaded once. ActiveX wins all speed contests and Java wins on of the security issues. If all you care about is performance, ActiveX is the clera winner. If security matters you must accept Java´s lack of speed as the penalty to pay*.”

5.1.7 Diskussion

Inledningsvis vill jag uppmärksamma de svårigheter som det innebär att jämföra de båda teknikerna Java applets och ActiveX. Svårigheterna uppstår p.g.a att Java är ett programmeringsspråk som används för att skriva applikationer eller applets medan ActiveX är en teknologi för att skapa exekverbara objekt. Tekniska rapporter från Microsoft vill också betona att ActiveX inte konkurerar med Java och menar att de båda teknikerna går att kombinera för att skapa webbapplikationer som drar fördel av båda tekniker[74].

Sun utvecklade under slutet av 1990-talet JavaBeans, den javabaserade teknik som påminner mest om ActiveX-kontroller. JavaBeans och ActiveX utför samma typ av funktioner. Sun har dessutom utvecklat en ActiveX-brygga för JavaBeans som ger ActiveX-utvecklare möjligheten att använda och bädda in portabla JavaBean-komponenter i exempelvis Word eller Visual Basic-applikationer. Detta sker på samma sätt som vid inbäddningen av de plattformsberoende ActiveX-kontrollerna.

När jag inledde arbetet med denna uppsats hade varken jag, eller min handledare på Resco, någon kunskap om kombinationer mellan ActiveX och Java applets eller JavaBeans och undersökningens frågeställning och syfte fokuserades därför på Java

[74] Microsoft, 2001

applets i jämförelse med ActiveX. Om jag haft den kunskap jag har i dag rörande Java och ActiveX anser jag att en studie hur ActiveX kan kombineras med Javabaserade tekniker eller en jämförelse mellan JavaBeans och ActiveX hade varit mer intressant. Även utvecklingen av den klientbaserade applikationen kunde, med denna förkunskap, gjorts annorlunda. Denna undersökning syftar dock till att undersöka klientbaserad programmering och dess tekniker samt att jämföra Java applets och ActiveX eftersom det var detta var den kunskap Resco var i behov av när arbetet inleddes.

De artiklar och rapporter jag analyserat är alla i stort sett eniga när det gäller fördelar och nackdelar med respektive teknik. Jag har i nedanstående diagram gjort en sammanfattning av hur jag, efter min jämförelse, ser på ActiveX och Java applets.

ActiveX	**Java applets**
Fördelar: Snabb exekvering p.g.a. kompilerad kod. En ActiveX-kontroll laddas bara ned en gång och är sedan tillgänglig för systemet. En ActiveX-kontroll kan skrivas i många olika programmeringsspråk..	**Fördelar:** Plattformsoberoende p.g.a. interpreterad kod och JVM. Säkert för användare p.g.a. JVM och sandlådeprincipen.
Nackdelar: Utvecklat för Windowsmiljön och därför brister i plattformsoberoende. Låg säkerhetsnivå, ActiveX-kontroller körs på det lokala operativsystemet och har obegränsad tillgång till systemet.	**Nackdelar**: Exekverars långsamt p.g.a. att koden måste interpreteras av JVM. Måste laddas ned till användaren varje gång webbsidan efterfrågas.

För att, utifrån denna jämförelse, svara på om och varför Java applets är en bra teknik, för den klientbaserade applikation Resco är i behov av, är min slutsats att det beror på den miljö applikationen ska användas i. I Rescos fall beror valet av teknik på de krav kunden ställer för den efterfrågade applikationen. Valet kan även grundas på den kunskap och vilja webbutvecklarna har när det gäller en viss teknik. ActiveX går t.ex. att göra plattformsoberoende med hjälp av plugins om man föredrar att arbeta med denna teknik.

När det gäller den Java applet som uppsatsen syftar till att genomföra är kravet på plattformsoberoende stort. Applikationen ska användas på Internet av klienter med olika plattformar och webbläsare. Dessa förhållanden talar för att en applet är rätt teknik att använda sig av. Applikationen ska dock utföra operationer som kräver att appleten tar sig ur sandlådan. Dessa operationer är lättare att utföra med en ActiveX-kontroll eftersom de redan har tillgång till de efterfrågade resurserna. Kravet på säkerhet har dock avgörande betydelse och genom att signera appleten och samtidigt sätta upp en säkerhetspolicy, både på klient och server, för att begränsa appletens tillgång till systemet kan man tillgodose detta krav på ett tillfredsställande sätt.

Sammanfattningsvis ser jag beslutet att välja en teknik framför den andra på följande sätt:
Antingen lägger man ner mycket tid på att få ActiveX-kontrollen att fungera i de miljöer som krävs eller lägger man ner mycket tid på att på att få en Java applet att utföra de riskfyllda operationerna och ta sig ur sandlådan. Efter att analyserat artiklar och rapporter om de båda teknikerna anser jag att, i Rescos aktuella kundprojekt, är det senare att föredra.

5.2 Utvecklingen av FileApplet

I föregående avsnitt redovisades resultatet av den teoretiska undersökningen, jämförelsen mellan ActiveX och Java applets. Detta avsnitt syftar till att redovisa resultatet av den praktiska undersökningen, utvecklingen av FileApplet. Inledningsvis kommer kravspecifikationen för applikationen att beskrivas och sedan det praktiska tillvägagångssättet för utvecklingen av denna applikation. Eftersom den praktiska undersökningen har syftet att svara på frågan hur utvecklingen av en klientbaserad applikation kan utformas har jag valt att ge resultatet av utvecklingen en beskrivande karaktär. Jag kommer först att beskriva hur man skapar vissa, för applikationen, betydande funktioner för sedan precisera hur jag valt att gå till väga genom beskrivande kodrader. Koden i sin helhet återfinns i bilagorna 1-3. Kapitlet avslutas med en redovisning av de svårigheter och begränsningar jag stött på under utvecklingen samt en övergripande diskussion.

5.2.1 Kravspecifikation

Under de inledande diskussionerna med min handledare på Resco utformades en enkel kravspecifikation för FileApplet. Kravspecifikation innehåller en bakgrund till problemet och behovet, d.v.s. en beskrivning av:

- det sammanhang som applikationen skall ingå i.
- behovet av applikationen.
- det problem applikationen ska avhjälpa.

En kravspecifikation bör även innehålla en översiktlig beskrivning av själva applikationen samt funktionella respektive icke-funktionella krav, d.v.s. vad man ska kunna utföra med applikationen respektive vilka villkor och restriktioner som finns[75].

5.2.1.1 Bakgrund

Applikationen FileApplet är tänkt att användas som en klientbaserad filuppladdning/nedladdnings komponent. Rescos behov av denna applikation uppstod under utvecklingen av en webbsida i ett kundprojekt. Den specifika sidan är utformad för att distribuera filer mellan olika deltagare i ett projekt. Användaren loggar in på webbsidan och de projekt användaren deltar i presenteras. Användaren väljer därefter ett projekt och alla de filer som används i det specifika projektet presenteras. En fil kan därefter markeras för uppladdning eller nedladdning, ”check in” respektive ”check out”. Tanken är att endast en projektdeltagare i taget ska kunna arbeta med filen för att inga felaktiga överskrivningar av information ska inträffa. Om en fil är ”incheckad” kan en projektdeltagare checka ut filen, arbeta med den och sedan checka in den igen. Endast den deltagare som har checkat ut filen är berättigad till att checka in den och under tiden deltagaren arbetar med filen är den ej tillgänglig för övriga deltagare. Figur 6 är en bild på den sida webbsida som ska anropa FileApplet.

[75] http://www.du.se/iot/gtbm/byggteknik/introkurs/forelasningar/intro/tsld025.htm

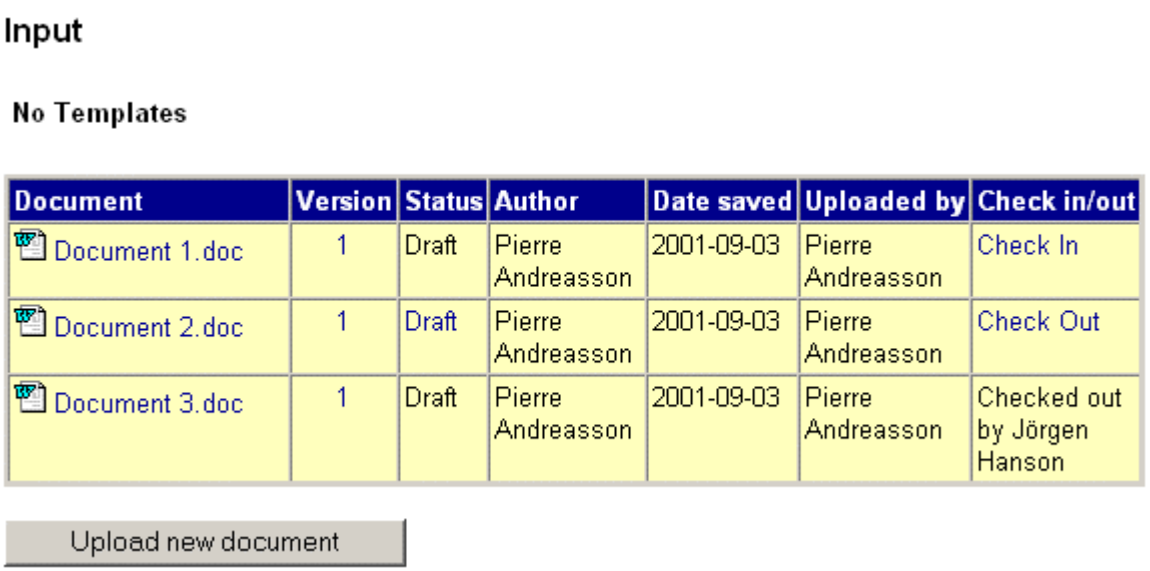

Fig. 6 Skärmdump på den webbsida från vilken FileApplet ska anropas.

Applikationen FileApplet anropas och exekveras när användaren klickar på texten check in respektive check ou.t. Applikationens uppgift är att ta emot information från webbsidan angående vilken fil som ska laddas upp respektive ned, d.v.s var filen är lokaliserad och vilken slutdestination filen ska ha. Dessa sökvägar är redan definierade när appleten anropas och det är inte FileApplets uppgift att ta fram denna information. På servern finns en bestämd katalog för filerna i de olika projekten och det är även tänkt att en liknande katalogstruktur ska finnas på klienten. FileApplet har därför även i uppgift att kontrollera om en katalog med projektets namn existerar på klienten och om så ej är fallet skapa denna katalog innan nedladdningen av filen påbörjas.

Det problem applikationen ska avhjälpa är att genom klientbaserad programmering åstadkomma nedladdning av filer från server till klient respektive uppladdning av filer från klient till server. Applikationen ska kunna få tillgång till klientens filsystem och på så sätt kringgå de säkerhetsbarriärer som annars är liktydiga med klientbaserade applikationer.

5.2.1.2 Funktionella krav

Funktionella krav beskriver de funktioner som är väsentliga och centrala för applikationen. När det gäller FileApplet är de funktionella kraven enligt följande:

- **Filuppladdning**: Applikationen ska ta emot två sträng-parametrar från en webbsida, en sträng med sökvägen till den fil som ska laddas upp från det lokala filsystemet samt en sträng med sökvägen till den plats på serverns filsystem där filen ska sparas. Applikationen ska utifrån dessa parametrar utföra själva uppladdningen av filen till servern.
- **Filnedladdning**: Applikationen ska ta emot två sträng-parametrar från en webbsida, en sträng med sökvägen till den fil som ska laddas ned från serverns filsystem samt en sträng med sökvägen till den plats på klienten

lokala filsystem där filen ska sparas. Applikationen ska utifrån dessa parametrar utföra själva nedladdningen av filen till klienten.

- **Skapa katalog**: När en fil tillhörande ett visst projekt ska laddas ned på klientens filsystem ska applikationen utifrån katalognamnet i den erhållna sökvägen ta reda på om en katalog med rätt namn existerar. Om inte katalogen existerar ska en katalog med det erhållna katalognamnet skapas på klienten innan nedladdningen påbörjas.

5.2.1.3 Icke-funktionella krav

Icke-funktionella krav är de krav som ställs på applikationens egenskaper. FileApplet har följande icke-funktionella krav:

- Filuppladdning och nedladdning ska utföras med hjälp av HTTP-protokollet, s.k. HTTP-tunneling.
- Applikationen ska utvecklas som en Java applet.
- Applikationen ska kunna köras i Internet Explorer 4.0 och Netscape Navigator 4.0.
- Applikationsstorleken ska vara så liten som möjligt eftersom applikationen är webbaserad och användarnas uppkopplingshastighet, bandbredd m.m. varierar.

5.2.2 Teknisk utrustning

Eftersom utvecklingen av applikationen skedde på Rescos kontor och applikationen är tänkt att implementeras i ett av Rescos projekt blev det naturligt att använda den tekniska utrustning som fanns att tillgå på kontoret. Utvecklingen skedde således på plattformen Windows NT 2000. Som webbserver användes Microsofts Internet Information Server. All kod skrevs och kompilerades i Java-editorn Kawa 6.0 med Java 2 Softare Development Toolkit.

5.2.3 FileApplet

Applikationen FileApplet utvecklades med målet att uppfylla de krav som specificerades i kravspecifikationen. Tre klasser användes för att bygga upp själva applikationen, TheApplet, UploadFile, DownloadFile (se bilagor 1-3). Jag kommer här att beskriva uppbyggnaden av de olika klasserna och ge kodexempel på hur man, i Java, kan utforma en uppladdnings/nedladdnings-applikation.

5.2.3.1 TheApplet

Klasserna som bygger upp programmeringsspråket Java är organiserade i ett bibliotek, Java API. Biblioteken grupperas vanligtvis efter funktionalitet. Andra bibliotek är dock utformade som ”byggdelar” för att skapa egendefinierade klasser. En kategori av dessa är Javas application framework, vars syfte är att hjälpa utvecklaren att bygga applikationer genom att tillhandahålla en klass eller en grupp klasser som förser en applikation med grundläggande funktioner. Applets byggs upp

genom att utnyttja ett application framework. När man skapar en applet ärver man beteende och funktioner från en fördefinierad klass, Applet, som tillhör biblioteket java.applet i Javas API. Följande kodrad visar hur jag skapat TheApplet genom att importera och ärva från klassen Applet.

import java.applet.;*

public class TheApplet extends Applet{

Följande metoder använd för att initiera och exekvera en applet:

Metod	**Operation**
init( )	Anropas automatiskt varje gång appleten initialiseras.
start( )	Anropas varje gång appleten blir synlig i webbläsaren, börjar exekveras, för att starta appletens normala operationer.
stop( )	Anropas varje gång exekveringen av appleten ska avslutas, t ex när användaren surfar bort från den aktuella webbsidan eller stänger ner webbläsaren.
destroy( )	Anropas innan appleten ska tas bort från webbsidan för att utföra frigöra de resurser appleten utnyttjat[76].

Den klientbaserade appleten TheApplet initialiseras när användaren klickar på check in eller check out på webbsidan. För att starta en applet från en webbsida används HTML-taggen <APPLET>:

<APPLET CODE=*TheApplet.class*></APPLET>

När webbläsaren stöter på ovanstående kod händer följande:

1. Webbläsaren letar upp klassfilen TheApplet. Var klassfilen, bestående av bytekod, är lokaliserad framgår av attributet CODE i applet-taggen.
2. Webbläsaren hämtar bytekoden från webbservern och laddar ner koden på användarens dator.
3. Webbläsaren skapar en instans av Javaklassen Applet, d.v.s klassen TheApplet.
4. Webbläsaren anropar TheApplet:s init-metod och appleten initialiseras.
5. Webbläsaren anropar appletens start-metod som sätter igång de operationer som appleten ska utföra[77].

I applikationen FileApplet ska appleten ta emot två parametrar, sökvägen till filen som ska laddas upp eller ned samt filens slutdestination. Både när det gäller uppladdning och nedladdning ska två sökvägar till fil respektive destination skickas med och för att bestämma vilken operation appleten ska sätta igång har jag valt att

[76] http://java.sun.com/docs/books/tutorial/applet/index.html
[77] Ibid

skicka med en ytterligare parameter som talar om vilken text användaren har tryckt på, check in eller check out. Parametrarna skickas till appleten på följande sätt:

<APPLET CODE=*TheApplet.class*>
<PARAM NAME= *currentFile* VALUE= *Sökvägen till den lokal fil som ska laddas upp*
>
<PARAM NAME= *destinationFile* VALUE= *Sökvägen till den plats filen ska sparas på servern*>
<PARAM NAME= *check* VALUE= *Texten som valts*>
</APPLET>

TheApplet tar emot dessa parametrar på följande sätt:

```
String currentFile =  getParameter(“currentFile”);
String destinationFile = getParameter(”destinationFile”);
String check = getParameter(”check”);
```

När TheApplet fått tillgång till de parametrar den behöver för att starta nedladdning eller uppladdning skapas en tråd, uploader eller downloader. Trådar används för att starta och exekvera flera processer samtidigt i en och samma applikation eller för att underlätta för en applet att utföra ”förbjudna” åtgärder, vilket jag kommer att beskriva mer ingående längre fram. TheApplet bestämmer vilken tråd som ska skapas utifrån parametern check:

```
If check = = “check out” {
  Thread downloader = new Thread() {
     public void run() {
        DownloadFile client = new DownloadFile();
           try{
             client.download(destinationFile, currentFile);
           }catch(Exception ex){}

     }
  };
  downloader.start();
}
```

Ovanstående kod skapar en tråd, *downloader*, och metoden *public void run()* preciserar vad tråden ska utföra. Tråden startas längre ned med koden *downloader.start()*. Tråden skapar en instans av klassen *DownloadFile* som får namnet *client*. Via metodanropet *client.download(destinationFile, currentlFile)* skickas parametrarna vidare till klassen *DownloadFile* som tar hand om själva nedladdningen. Koden som tar hand om uppladdning av filer är utformad på liknande sätt men tråden kallas *uploader* och en instans av klassen *UploadFile* skapas istället för *DownloadFile*.

```
If check = = “check in” {
  Thread uploader = new Thread() {
     public void run() {
        UploadFile client = new UploadFile();
```

```
                try{
                  client.upload(destinationFile, currentFile);
                }catch(Exception ex){}

        }
    };
    uploader.start();
}
```

5.2.3.2 DownloadFile

För att överföra filer från ett filsystem till ett annat över Internet måste man skapa kopplingar, strömmar, till de platser där filerna är lokaliserade respektive de platser där filerna ska sparas. Ett av kraven i kravspecifikationen var att all filöverföring skulle utföras med HTTP-protokollet och jag har därför använt mig av Javaklassen *HttpURLConnection* för att skapa strömmar till de filer som antingen ska laddas upp eller ned från servern. Klassen *HttpURLConnection* tillhör biblioteket java.net i Javas API, ett bibliotek som innehåller klasser för att skapa nätverksapplikationer i Java. URL står för Uniform Resource Locator, och är en ”pekare” till en resurs på WWW, i detta fall en fil[78]. I kapitel 3, Distribuerade system, beskrivs hur en URL är uppbyggd. Följande exempel visar hur en URL till en fil i det specifika projektet kan se ut:

http://www.resco.se/ipcp/projekt1/index.txt

När TheApplet tar emot parametrar från webbsidan består en av dessa parametrar av en URL till en fil på servern. Genom att skapa en ström till denna URL kan man antingen läsa filen, vilket sker vid nedladdning, eller skriva till filen, vilket sker vid uppladdning. Den andra parametern består av en sökväg till en fil eller en plats på klientens lokala filsystem från vilken antingen uppladdning eller nedladdning ska ske. För att läsa och skriva till denna plats måste man också skapa strömmar. Läsningen eller skrivningen sker här lokalt, ingen data ska skickas över WWW, och därför används vanliga filströmmar. Jag har valt att använda FileOutputStream för att skriva till en fil eller en plats och FileInputStream för att läsa en fil. FileInputStrem och FileOutputStream tillhör biblioteket java.io i Javas API. Följande exempel visar hur en sökväg till en fil/plats på det lokala filsystemet kan se ut:

H:\ipcp\projekt1\index.hml

Klassen DownloadFile instansieras i TheApplet genom att objektet client skapas. Genom detta objekt anropas sedan metoden download i klassen DownloadFile och i detta metodanrop skickas parametrarna currentFile och destinationFile med.

```
import java.io.*;
import java.net.*;

public class DownloadFile{

                protected static void download(String currentFile, String destinationFile){
```

[78] http://java.sun.com/docs/books/tutorial/applet/index.html

Eftersom DownloadFile har till uppgift att ladda ner en fil från servern till klienten innehåller currentFile sökvägen till filen på servern, d.v.s. en URL, och destinationFile den plats där filen ska sparas ned på klienten. För att skapa en URL skriver man enligt följande:

URL url = new URL(currentFile);

För att skapa en plats där filen ska sparas skriver man enligt följande:

File fil = new File(destinationFile);

När en URL respektive File har skapats återstår att skapa strömmar för att läsa filen från URL och skriva filen till File. För att skapa en ström från en URL måste man först göra en uppkoppling till denna URL genom att använda klassen URLConnection. URLConnection är en klass som skapar kommunikationslänkar mellan en applikation och en URL och instanser av denna klass används både för att läsa och skriva till den resurs som URL:en pekar på. HttpURLConnection är en underklass till URLConnection och jag har valt att använda denna klass eftersom den, enligt Sun, specifikt stödjer HTTP-protokollet. Man öppnar en HttpURLConnection på följande sätt:

HttpURLConnection urlCon = url.openConnection();
urlCon.setDoInput(true);

Den andra raden sätter uppkopplingens inställningar till att tillåta avläsning. När man har öppnat en *HttpURLConnection* kan man skapa en ström för att, i detta fall, läsa innehållet i en fil. Jag använder här en *DataInputStream* för att skapa denna ström.

DataInputStream dis = new DataInputStream(new BufferedInputStream
(urlCon.getInputStream()));

Nästa steg är att skapa en ström till mitt File-objekt för att skriva det filinnehåll jag läser från ovanstående ström.

FileOutputStream fos = new FileOutputStream(fil);

När strömmarna är klara påbörjas nedladdningen och de båda strömmarna stängs.

```
while (nByte < size){
   fos.writeByte(dis.readByte());
  nByte++;
}
dis.close();
dos.close();
```

När en fil, tillhörande ett visst projekt, ska laddas ned är det inte säkert att en katalog med rätt namn existerar. Ett av kraven i kravspecifikationen var att applikationen skulle utföra denna kontroll och om katalogen ej existerar skapa en ny katalog med

projektets namn. För att ta reda på om katalogen inte existerar och därefter skapa katalogen har jag skrivit enligt följande:

```
katalog = Fil.getPath();
if (!katalog.isDirectory()){
   katalog.mkDirs();
}
```

5.2.3.3 UploadFile

Klassen UploadFile är uppbyggd på samma sätt som DownloadFile med den undantaget att strömmarna som ska läsa och skriva filerna är omvända. I DownloadFile skapades en ”inström” från en URL till en fil på servern och en ”utström” till en plats på klientens hårddisk. Vid uppladdning måste man göra tvärtom, en inström till servern och en utström till klienten.

```
HttpURLConnection urlCon = url.openConnection();
urlCon.setDoOutput(true);

DataOutputStream dos = new DataOutputStream(new BufferedOutputStream
(urlCon.getOutputStream()));

FileInputStream fis = new FileInputStream(fil);
```

Därefter sker läsning från klientens fil till serverns URL på samma sätt som vid DownloadFile.

5.3.3 Signering av applets

I kapitel 4, Klientbaserad programmering, behandlades säkerhetsaspekten när det gäller klientbaserad kod. På grund av den säkerhetsrisk som det innebär att exekvera nedladdningsbar kod på klientens dator har Sun Microsoft utvecklat en s k sandlåda i vilken applets exekveras. Sandlådan innebär att appleten inte kan göra något som skulle kunna innebära ett hot mot klientens maskin, t ex läsa, skriva, ta bort filer, kommunicera med andra maskiner. I tidiga webbläsare fanns inga mekanismer för att tillåta applets att verka utanför sandlådan. Moderna webbläsare som Internet Explorer 4.0 och Netscape Navigator 4.0 eller senare har dock utrustats med mekanismer för att ge ”tillförlitliga” applets möjligheten att ta sig ur sandlådan. Detta görs genom att digitalt signera applets med en identifikation. Klienten som appleten laddas ned till måste fastställa att han/hon har tillit till appleten som är signerade med denna signatur[79]. Från och med utgivandet av Suns Java Development Kit (JDK) version 1.2 finns det inbyggda säkerhetsredskap så att användare och utvecklare kan signera applets och applikationer samt definiera sin lokala säkerhetspolicy. En säkerhetspolicy specificerar hur stor del av det lokala systemet som signerade applets och applikationer ska få tillgång till. För att en applet till exempel ska få tillgång till lokala resurser på en klients maskin måste den beviljas särskild tillgång till dessa resurser. En applet beviljas tillgång till specifika resurser genom att man

[79] Griscom, 1999

utformar en säkerhetspolicyfil, innehållandes URL:er till de resurser som ska vara tillgängliga för appleten[80].

Digitala identifikationer är helt enkelt påståenden, signerade av en certifikatmyndighet, Certificate Authority, som talar om att den här personen är den han säger sig vara och den här Certificate Authority går i god för honom. Ett digitalt ID består av 2 delar, ett publikt certifikat och en privat nyckel. Den privata nyckeln är den krypteringskod du använder för att signera dina applets och applikationer och det publika certifikatet använder andra personer för att verifiera att dina applets och applikationer är signerade med din privata nyckel.

En Certificate Authority (CA) är ett tillförlitligt företag som säljer digitala ID. En CA är tillförlitlig om dess certifikat är installerad i klientens webbläsare och är redo att godkänna de digitala ID den har distribuerat. För att webbläsaren ska kunna tolka en digitalt signerad applet måste det alltså finnas två digitala ID inblandade, en som används för att signera appleten och en som installeras i webbläsaren för att verifiera validiteten av den förra. Resco har köpt sina digitala ID från en CA som heter VeriSign, vars certfikat är förinstallerade i de nyaste versionerna av Internet Explorer och Netscape Navigator[81]. Nedan följer de steg man måste gå igenom för att signera FileApplet och specificera klientens lokala säkerhetspolicy för hur stor tillgång till systemet FileApplet ska få.

5.3.4.1 Signering av FileApplet med VeriSign

För att signera FileApplet måste man genomgå följande sex steg:

1, Ladda ner Java 2 Software Development Kit, JSDK:
Vid utvecklingen av FileApplet användes denna utvecklingsmiljö. JSDK fanns därför tillgängligt på datorn innan signeringen påbörjades. De verktyg i JSDK som används för att skapa ett digitalt ID och signeringa koden är keytool, jar och jarsigner.

2. Generera publika och privata nycklar:
VeriSign använder sig av en databas, s.k. keystore, för att lagra de publika och privata nycklarna. För att skapa FileApplets nycklar skapas ett alias, MyCert, för keystore genom följande kommando:

C:\>C:\jdk1.3\bin\keytool -genkey -keyalg rsa -alias MyCert

Keytool svarar med att visa en prompt för inskrivning av lösenord för databasen, utvecklarens namn, organisation samt adressinformation. Uppsättningen av den publika respektive privata nyckeln genereras av automatiskt av keytool och sparas i MyCert. Dessa nycklar används sedan för att signera applikationen.

3. Generera en Code Signing Digital ID Signing Request (CSR):
För att skapa en förfrågan om ett digitalt ID (CSR) för nycklarna skickas följande komando till keytool:

[80] http://developer.java.sun.com/ developer/technicalArticles/Security/Signed/index.html
[81] http://www.verisign.com/rsc/wp/javaSigning/index.html

C:\>C:\jdk1.3\bin\keytool -certreq -alias MyCert

Keytool svara med en prompt där lösenordet för MyCert måste skrivas, därefter skapas en CSR. Följande visar hur en CSR kan se ut:

```
-----BEGIN NEW CODE SIGNING ID REQUEST-----
MIIBtjCCAR8CAQAwdjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNBMRIwE
AYDVQQHEwlDdXBlcnRpbm8xGTAXBgNVBAoTEFN1biBNaWNyb3N5c3RlbX
MxFjAUBgNVBAsTDUphdmEgU29mdHdhcmUxEzARBgNVBAMTClN0YW5sZXk
gSG8wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBALTgU8PovA4y59eb
oPjY65BwCSc/zPqtOZKJlaW4WP+UhmebE+T2Mho7P5zXjGf7elo3tV5uI
3vzgGfnhgpf73EoMow8EJhly4w/YsXKqeJEqqvNogzAD+qUv7Ld6dLOv0
CO5qvpmBAO6mfaI1XAgx/4xU/6009jVQe0TgIoocB5AgMBAAGgADANBgk
qhkiG9w0BAQQFAAOBgQAWmLrkifKiUYtd4ykhBtPWSwW/IKkgyfIuNMML
dF1DH8neSnXf3ZLI32f2yXvs7u3/xn6chnTXh4HYCJoGYOAbB3WNbAoQR
i6u6TLLOvgv9pMNUo6v1qB0xly1faizjimVYBwLhOenkA3Bw7S8UIVfdv
84cO9dFUGcr/Pfrl3GtQ==
-----END NEW CODE SIGNING ID REQUEST-----
```

Fig. 7 Exempel på en CSR (från www.verisign.com)

En CSR innehåller en kopia av utvecklarens publika nyckel samt en förvrängd form, en s.k. hash, av den information som skrevs in i steg 2. Den CSR kopieras och klistras sedan in i VeriSigns Sun Java Signing ID-formulär på VeriSigns hemsida för godkännande. När CSR är godkänd skickas ett digitalt ID, ett s.k. certifikat, till utvecklaren. Detta ID sparas i en fil för att sedan användas under signeringen av koden.

4. Importera det digitala ID:et:
Följande kommando, innehållandes sökvägen till ID:et, importerar ID:et till databasen, MyCert:

C:\>C:\jdk1.3\bin\keytool -import -alias MyCert -file VSSStanleyNew.cer

5. Packetera FileApplet i en JAR-fil:
Alla de klassfiler som applikationen innehåller sparas i en katalog, FileApplet, för att sedan packeteras i en JAR (Java Archive)-fil. Detta görs genom följande kommando:

C:>C:\jdk1.3\bin\jar cvf C:\FileApplet.jar .

6. Signera FileApplet:
För att signera JAR-filen används jarsigner och den privata nyckeln i MyCert:

C:\>C:\jdk1.3\bin\jarsigner C:\FileApplet.jar MyCert

Jarsigner signerar FileApplet och sparar en hash-variant av applikationen tillsammans med en kopia av det digitala ID:et i ovan nämnda JAR-fil

Den signerade appleten anropas på webbsidan på följande sätt:

<APPLET CODE=*TheApplet.class* ARCHIVE=*FileApplet.jar*>
<PARAM NAME= *currentFile* VALUE= *Sökvägen till den lokal fil som ska laddas upp* >

<PARAM NAME= *destinationFile* VALUE= *Sökvägen till den plats filen ska sparas på servern*>
<PARAM NAME= *check* VALUE= *Texten som valts*>
</APPLET>

När den signerade JAR-filen laddas ned till en klient visas det digitala ID:et för användare och användaren avgör om han/hon vill att appleten ska köras[82].

5.3.4.2 Upprättandet av en säkerhetspolicy

För att ytterligare stärka säkerheten kring signerade applets har Sun utvecklat policyfiler. När koden laddas ned till klienten ges den olika ”tillstånd” beroende på vem som signerat koden. Varje tillstånd specificerar hur applikationen får utnyttja de lokala resurserna, t.ex. läsa och skriva till en specifik katalog eller ansluta sig till en specifik port. De olika tillstånd applikationer från olika källor har upprätthålls av en policy-fil. Denna policyfil skapas och sparas på klienten.
En policyfil kan utformas i en vanlig editor eller via det grafiska gränssnittet Policy Tool. Policyfilen sparas sedan i jdk:s lib-katalog:

C:>C:\jdk1.3\lib\security\java.policy .

Följande policyfil, utformad i en editor, ger FileApplet, signerad av Resco, tillstånd att skriva och läsa från en katalog på klientens dator:

```
grant SignedBy ”resco” {
      permission java.util.PropertyPermission
            ”URL till projektets katalog”, ”read”
      permission java.io.FilePermission
            ”URL till projektets katalog”, ”write”
};
```

5.4 Diskussion

Under utvecklingen av FileApplet har jag stött på en rad problem vilket har resulterat i att applikationen, i sin helhet, ej har kunnat provköras eller implementerats. Svårigheterna har berott på att signeringen av applikationen ej lyckats. Detta är ett problem som jag inte kunnat råda över utan beror på missförhållanden i kommunikationen mellan VeriSign och Resco när det gäller skapandet av ett digitalt certifikat för signering av kod. Jag kan således enbart bygga resultatet på antaganden om att den kod och de klasser jag skapat kan utföra de, i kravspecifikationen, efterfrågade operationerna, när laddas ned till en klient. Filöverföringen, enligt den kod jag presenterat, har dock testats lokalt i vanliga Java applikationer som körs lokalt och då fungerat på ett tillfredställande sätt. Tekniken med att signera applets för att ge dem tillgång till klientens lokala system är något som beskrivits och diskuterats i olika diskussionsforum på Suns och Java Ranch´s webbsidor. Jag har deltagit i dessa forum och på så sätt dessutom fått information och hjälp, under utvecklingen av FileApplet, av erfarna Java-utvecklare. Även många

[82] http://www.verisign.com/rsc/wp/javaSigning/index.html

av de tutorials jag använt mig av har beskrivit hur denna typ av applets kan utvecklas, vilket stödjer kodens tillförlitlighet.

Det faktum att applikationen fortfarande är under utveckling gör det svårt att, utifrån denna resultatdel, svara på om Java applets är en bra teknik för att utveckla den här typen av klientbaserade applikationer. De slutsatser jag kan dra grundar sig på de erfarenheter jag anskaffat mig under arbetet med denna applikation och de är att Java är ett kraftfullt programmeringsspråk för Internetbaserade applikationer. I Javas API finns väl utvecklade fördefinierade klasser för att skapa de URL-strömmar som behövs för uppladdning och nedladdning av filer. Det jag ser som negativt med Java är att språket kan vara svårt att använda om man ej är insatt i hur API:n är uppbyggt, hur instansieringen av objekt går till o.s.v. Vill man, som utvecklare, använda sig av Java applets finns det dock mycket information och kodexempel att ladda ner, framförallt på Sun:s webbsida. Säkerheten i språket och användningen av signering tillsammans med säkerhetspolicys är en väl utarbetat strategi. När det gäller information om hur man ska gå tillväga för att skapa en säker klientbaserad applikation finns det många utförliga och bra manualer, både på Suns och VeriSigns hemsida.

Trots att jag inte kan garantera att FileApplet fungerar i praktiken, utan enbart i teorin, anser jag att detta kapitel levt upp till delsyftet att visa hur denna typ av programmering kan gå till. För Rescos delning kan jag rekommendera användningen av denna teknik. Denna slutsats grundar sig på programmeringsspråkets ovan nämnda egenskaper: starkt utvecklat stöd för nätverksoperationer, väl utarbetad säkerhetsstrategi samt stor tillgänglighet till material på Internet.

6 Sammanfattande diskussion och slutsats

Detta kapitel har avsikten att ge en övergripande bild av det resultat jag kommit fram till och sammanfatta tidigare resultatdiskussioner. Därefter redovisas de slutsatser jag dragit utifrån uppsatsen frågeställning och syfte. Kapitlet, och därmed även uppsatsen, avslutas med att ge exempel på eventuell fortsatt forskning inom problemområdet.

6.1 Sammanfattande diskussion

Uppsatsen huvudsakliga frågeställningar var att utröna om Java applets är en bra teknik för att åstadkomma klientbaserade applikationer. Jag valde att försöka svara på denna fråga genom att sätta upp tre delsyften för undersökningen. Det första var att undersöka begreppet klientbaserad programmering för att ta reda på hur denna typ av programmering växt fram., vilket behandlas i kapitel 3 ,Distribuerade system, respektive kapitel 4, Klientbaserad programmering. Sammanfattningsvis kan man utifrån innehållet i dessa kapitel säga att klientbaserad programmering härstammar ifrån distribuerade system och den client/server-modell som utvecklats för denna arkitektur. Under Internets explosionsartade utveckling på 1990-talet överfördes denna teknik på webbapplikationer och klientbaserad programmering ses i detta sammanhang som ett sätt att effektivt utnyttja både serverns och klientens resurser. Begreppet klientbaserad programmering innebär att exekveringen av kod delas upp mellan klient och server. Klientbaserade applikationer kan på så sätt avlasta belastningen på servern och denna typ av applikationer används i huvudsak för att tillföra dynamiska webbapplikationer och öka användarinteraktiviteten.

Avsikten med ovan nämnda delsyfte var även att beskriva i vilka sammanhang klientbaserad programmering kan användas samt ge exempel på några tekniker. I kapitel 3 görs därför en gruppering av olika tekniker utifrån användningsområde. De användningsområden jag tar upp är handskas med ny typ av information, tillhandahålla lokal databehandling samt integrering av objekt och applikationer. Exempel på tekniker inom det första användningsområdet är hjälpapplikationer och plugins. HTML 3.2 Client-side features, DHTML, script och Java applets är tekniker som tillhandahåller lokal databehandling och SSDI och ActiveX är tekniker för integrering av objekt och applikationer.

Det andra delsyftet med undersökningen var att jämföra två tekniker, Java applets och ActiveX, utifrån aspekterna plattformsoberoende, säkerhet och funktionalitet. Den första delen av undersökningens resultat, kapitel 5.1, redovisar denna jämförelse. Resultatet av redogörelsen kan sammanfattas enligt följande:

- **Plattformsoberoende**: ActiveX utvecklades av Microsoft för att enkelt kunna integreras i Windows-applikationer och är därför främst utformad för Windows och webbläsaren Internet Explorer. Java är däremot ett programmeringsspråk som utvecklats av Sun Microsystems med avsikten att skapa plattformsoberoende applikationer. Vid utvecklingen webbapplikationer kan man därför dra slutsatsen att Java är en bättre teknik för Internet, där användare med olika typer av plattformar och webbläsare laddar ner den klientbaserade koden. ActiveX klarar inte, i jämförelse med Java, att uppfylla denna typ av plattformsoberoende och passar därför bättre i intranätbaserade

webbapplikationer, där alla användare använder sig av Windows och Internet Explorer.

- **Säkerhet**: Java applets och ActiveX använder sig av olika angreppssätt för att upprätthålla säkerheten vid nedladdning och exekvering av klientbaserad kod. ActiveX använder sig av signering och förlitar sig på att användare, utifrån en verifiering av denna signatur, kan bedöma om koden är tillförlitlig. Java använder sig av sandlådemodellen för att begränsa applets funktionalitet och säkerställa att de inte utnyttjar användarens dator på ett otillbörligt sätt. Om det finns behov av att en applet ska ta sig ur sandlådan och utföra operationer på klienten använder sig Java av signering tillsammans med policyfiler. Javas metod anses därför säkrare än ActiveX, vilket är ytterligare en anledning till att Java passar bättre på Internet och ActiveX i intranät.
- **Funktionalitet**: Eftersom ActiveX inte är ett programmeringsspråk utan en teknologi för att skapa exekverbara objekt är det svårt att mäta funktionaliteten på denna teknik. Funktionaliteten hos ActiveX-kontroller beror till största delen på funktionaliteten i de programmeringsspråk som den specifika kontrollen är skriven i. Java funktionalitet bedöms utifrån det inbyggda stöd för bl.a. trådning, nätverkskopplingar, filsystemsoperationer m.m. Det finns ett väl utvecklat stöd för att utföra detta i Javas API och Javas funktionalitet kan därför anses som god. När det gäller exekvering och nedladdning av klientbaserad kod är ActiveX är snabbare än Java. Detta beror på att Java är ett interpreterat språk och ActiveX ett kompilerat samt att ActiveX endast behöver laddas ned en gång till klienten medan Java applets laddas ned varje gång webbsidan efterfrågas.

Det tredje del syftet med denna undersökning var att utveckla en klientbaserad filuppladdning/nedladdningsapplikation med Java applets för att kartlägga eventuella problem och begränsningar med denna teknik. Den andra delen av undersökningens resultat, kapitel 5.2, redovisar denna utveckling. Utveckling resulterade i applikationen FileApplet bestående av Java-klasserna TheApplet, UploadFile och DownloadFile. På grund av svårigheter med signeringen av denna applikation har FileApplet ej kunnat testköras och har därför mer syftet att teoretiskt visa hur utvecklingen av denna typ av applet kan gå till.

6.2 Slutsats

Avsikten med undersökningen var att svara på frågan:

Är Java applets en bra lösning för att utveckla en klientbaserad filuppladdning/nedladdningsapplikation med avseende på plattformsoberoende, säkerhet och funktionalitet?

Efter att ha undersökt ovan nämnda delsyften har jag dragit följande slutsats:

Kraven på plattformsoberoende och säkerhet är avgörande faktorer vid utvecklingen av klientbaserade applikationer på Internet. Java uppfyller dessa krav på ett, för både utvecklare och användare, tillfredställande sätt. Jag anser där för att Java applets är en bra lösning för denna typ av applikation.

6.3 Fortsatt forskning

Klientbaserad programmering är inget nytt begrepp och under de senaste åren har många tekniker för att åstadkomma denna typ av programmering växt fram. Som jag nämnt i kapitel 5.1.7 har Microsoft och Sun dels utvecklat nya tekniker för klientbaserad programmering och dels utvecklat tekniker för att kombinera ActiveX med Java. Under arbetet med denna uppsats har jag därför haft känslan av att ”ligga fel i tiden”. I ett framtidsperspektiv anser jag det därför intressant att undersöka hur man kan åstadkomma en applikation, liknande FileApplet genom att antingen kombinera ActiveX eller Java eller använda sig av en helt ny teknik, t.e.x. JavaBeans. Även en jämförelse mellan ActiveX och JavaBeans skulle vara av intresse.

7 Litteraturförteckning

7.1 Böcker

Capron, H.L. (1999), *Computers: tools for information age*, Prentice Hall Inc

Coulouris, George, Dollimore, Jean & Kindberg Tim (2001), *Distributed Systems: Concepts and Design*, Addison-Wesley

Fedorov, Alenader (1998), *Professional Active Server Pages 2.0*, Software Masters

Lewis, John & Loftus, William (1998), *Java Software Solutions*, Addison Wesley Longham Inc

McGraw, Gary & Felten, Edward W (1999), *Securing Java : getting down to business with mobile code*, Wiley Computer Pub.

Patel, Runa & Davidsson, Bo (1994), *Forskningsmetodikens grunder, Att planera, genomföra och rapportera en undersökning*, Lund Studentlitteratur

Thurén, Torsten (1991), *Vetenskapsteori för nybörjare*, Tiger Förlag AB

Wiedersheim-Paul, Finn & Eriksson, Lars (1997), Att utreda, forska och rapportera, Liber Ekonomi

7.2 Internet artiklar, böcker och rapporter

Barlotta, Michael J, *Distributed Application Development with PowerBuilder 6.0,* Manning Publications Com.
URL: http://books.manning.com/Barlotta/

Cert Coordination Center (2000), *Results of the Security in ActiveX Workshop,* Software Engineering Institute Carnegie Mellon University
URL: http://www.cert.org/archive/pdf/activeX_report.pdf

Committee on Information Systems Trustworthiness (1999), *Trust in Cyberspace,* National Research Council
URL: http://www.nap.edu/readingroom/books/trust/trustsum.htm

Eckel, Bruce (2000), *Thinking in Java,* Mindview
URL: www.mindview.net/Books/TIJ/

Flores-Méndez, Robert A (1997), *Programming Distributed Collaboration Interaction Through the World Wide Web,* University of Calgary, Department of Computer Science
URL: www.cpsc.ucalgary.ca/~robertof/publications/thesis

Griscom, Daniel (1999), *Code Signing for Java Applets,* Suitable Systems
URL: http://www.suitable.com/CodeSigningOverview.shtml#top

Hayes, Patricia (2000), *Technology briefing: Client-side technology*, ArrowWeb
URL: www.arrowweb.com/phayes/ec555/index.html

Hirsch, Fredrick J (1997), *Client Programming Tutorial Notes*, The Open Group Research Institute
URL: www.ultranet.com/~fhirsch/Presentations/www6_tutorial/notes/client_notes.html

Hodson, Scott (1998), *ActiveX and Java*
URL: http://www.scotthodson.com

Hoffman, Richard (1999), *Control Freaks and Java Junkies,* Technology Solution Center Network Computing
URL: http://www.nwc.com/1010/1010ws2.html

Kivi, Derek & Wong, Nelson (2000), *ActiveX, COM, and DCOM,* University of Calgary
URL: http://sern.ucalgary.ca/Courses/CPSC/547/W2000/webnotes/COM/COM.html

Kotz, David & Gray, Robert S (1999*), Mobile code: The Future of the Internet,* Dartmouth College, Department of Computer Science
URL: www.cs.dartmouth.edu/~dfk/papers/kotz:future/

LeBlanc, Camille, Jiang, Hialing, Kuntay, Aylin & Gerson, Kevin (1997), *Mobile Code: Java v. ActiveX*, University of California, SIMS
URL: http://www.sims.berkeley.edu/courses/is206/f97/GroupC/java.html

Murawski, Ron (1997), *Java – ActiveX*, Castlemall
URL: http://www.castlemall.com/ron/horizon6.shtml

Pollán, Raul Ramos (1998), *An introduction to Object Oriented Technology*, Cerns
URL: http://consult.cern.ch

Shoffner, Michael (1997), *JavaBeans vs ActiveX, Strategy Analysis*, JavaWorld
URL: http://www.javaworld.com/javaworld/jw-02-1997/jw-02-activex-beans.htm

Smith, Andrew (1997), *A Comparison of Three Network Programming Environments: Java, ActiveX, and Inferno*, Yale University, Department of Computer Science
URL: www.cs.yale.edu/homes/asmith/pap_crit.html

Smyth, Gavin (1998), *Java Applet Versus ActiveX,* Beesknees
URL: http://www.beesknees.freeserve.co.uk/articles/internet.html

Sol, Selena (2001), *Webprogramming 101 Part Two, Client Side Scripting*, Extropia
URL: www.extropia.com/tutorials/perl_cgi/pre_requisite_intro.html

Stanek, William R, Web *Publishing – Professional Reference Edition*, Zaphod
URL: http://zaphod.redwave.net/books/webpub/index.htm

Wilder, Clinton (2000), *IS Managers debate Java and ActiveX*, Information Week
URL:http://www.informationweek.com/newsflash/nf633/0603_st13.htm

Wisman, Ray (2001), *Client Side Programming*, Indiana University SouthEast
URL: homepages.ius.edu/RWISMAN/A348/html/Client.htm

von Pepel, Eva (1997), *Cascading Style Sheet,* Algonet
URL: www.algonet.se/~eva/ref/css/intro.html

Zapitis, Ioannis K (1998), *JavaBeans vs. ActiveX*, Intelligence Interactive Systems
URL: http://www.iis.ee.ic.ac.uk

7.3 WWW-sidor

Active-X.com
(http://www.active-x.com/articles/whatis.htm)

Chalmers Tekniska Högskola
(http://palver.dtek.chalmers.se/groups/d1proj-2000/d1proj13/allmant.html)

Högskolan Dalarna
(http://www.du.se/iot/gtbm/byggteknik/introkurs/forelasningar/intro/tsld025.htm)

Linköpings Universitet
(http://www-und.ida.liu.se/~petda482/it/vad.html)

Microsoft
(http://www.microsoft.com/activex/index.html)
(http://www.microsoft.com/com/tech/ActiveX.asp)
(http://www.ncompasslabs.com/scriptactive)
(http://www.microsoft.com/com/presentations/default.asp**)**

Resco
(www.resco.se)

Sun Microsystems
(http://java.sun.com/features/1998/05/birthday.html)
(http://sunsite.ics.forth.gr/javabeans/faq/faq.interop.html)
(http://java.sun.com/docs/books/tutorial/applet/index.html)
(http://developer.java.sun.com/developer/technicalArticles/Security/Signed/index.html
)

System Management
(searchsystemsmanagement.techtarget.com/sDefinition/0,,sid20_gci214004,00.html)

Stockholms Universitet
(www.internet.physto.se/fil/java.html)

Umeå Universitet
(student.educ.umu.se/~dis99-19/kallkritik/internetkritik.html)

University of Aberdeen
(http://www.abdn.ac.uk/diss/webpack/chap29.hti)

VeriSign
(http://www.verisign.com/rsc/wp/javaSigning/index.html)

Wilde´s WWW
(wildesweb.com/glossary/url)
(wildesweb.com/glossary/http)

World Wide Web Consortium
(http://www.w3.org/MarkUp/)

Bilaga 1

```
import javax.swing.*;
import java.applet.Applet;
import java.io.*;
import java.net.*;

public class TheApplet extends Applet implements runnable{

   String currentFile =  getParameter(”currentFile”);
   String destinationFile = getParameter(”destinationFile”);
   String check = getParameter(”check”);


   public void init(){
      System.out.println("TheApplet initialiseras");
   }

   public void start() {
      System.out.println("Applet starting.");

      If (check = = ”check out”) {

         Thread downloader = new Thread() {

            public void run() {
               DownloadFile client = new DownloadFile();
                     try{
                        client.download(destinationFile, currentFile);

                     }catch(Exception ex){}

         };
      }
      downloader.start();

      If (check = = ”check in”) {

         Thread uploader = new Thread() {

            public void run() {
              UploadFile client = new UploadFile();

            try{
                client.upload(destinationFile, currentFile);

            }catch(Exception ex){}
         };
      }
      uploader.start();
  }

  public void stop() {
                  System.out.println("Applet stopping.");
  }

  public void destroy() {
    System.out.println("Destroy method called.");
  }
}
```

Bilaga 2

```
import java.io.*;
import java.net.*;

public class UploadFile{

   protected static void download(String currentFile, String                    destinationFile){


URL url = new URL(destinationFile);
File fil = new File(currentFile);




HttpURLConnection urlCon = url.openConnection();
urlCon.setDoOutput(true);


DataOutputStream dos = new DataOutputStream(new BufferedOutputStream
(urlCon.getOutputStream()));
FileInputStream fis = new FileInputStream(fil);



                                        byte size = (byte)1024;
                                        byte nByte = 0;
                                        while (nByte < size){
                                          dos.writeByte(fis.readByte());

                                          nByte++;
                                        }
                                        dis.close();
                                        dos.close();

                                }
                                catch(MalformedURLException me)
                                {
                                System.err.println("MalformedURLException: " + me);
                                }
                                catch (EOFException ee)
                                {
                                System.err.println("EOFException: " + ee);
                                }
                                catch (SecurityException se)
                                {
                                System.err.println("SecurityException: " + se);
                                }
                                catch (IOException ioe)
                                {
                                System.err.println("IOException: " + ioe);                    }

                }

}
```

Bilaga 3

```
import java.io.*;
import java.net.*;

public class DownloadFile{

        protected static void download(String currentFile, String destinationFile){


URL url = new URL(currentFile);
File fil = new File(destinationFile);

File katalog = fil.getPath();
if (!katalog.isDirectory()){
    katalog.mkDirs();
}



HttpURLConnection urlCon = url.openConnection();
urlCon.setDoInput(true);

DataInputStream dis = new DataInputStream(new BufferedInputStream (urlCon.getInputStream()));
FileOutputStream fos = new FileOutputStream(fil);

                                        byte size = (byte)1024;
                                        byte nByte = 0;
                                        while (nByte < size){
                                          fos.writeByte(dis.readByte());

                                          nByte++;
                                        }
                                        dis.close();
                                        dos.close();

                                }
                                catch(MalformedURLException me)
                                {
                                System.err.println("MalformedURLException: " + me);
                                }
                                catch (EOFException ee)
                                {
                                System.err.println("EOFException: " + ee);
                                }
                                catch (SecurityException se)
                                {
                                System.err.println("SecurityException: " + se);
                                }
                                catch (IOException ioe)
                                {
                                System.err.println("IOException: " + ioe);
                                }

        }

}
```