


Handelshögskolan
VID GÖTEBORGS UNIVERSITET
Institutionen för informatik

2 januari 2006

Ajax på webben

undersökt och analyserat

En ny term, Ajax, har dykt upp inom webbutveckling. Den utlovar något fundamentalt nytt för vad som är möjligt på webben, och säger att webbapplikationer nu kan komma nära applikationer man finner i persondatorer, med avseende på smidighet och respons. Detta sker genom asynkron kommunikation mellan klienten (webbläsaren) och servern. Är Ajax något nytt, hur kommer Ajax att påverka webben och finns det existerande teorier som går att applicera på Ajax? Uppsatsen besvarar dessa frågor genom litteraturanalys, och kommer fram till att Ajax inte är något nytt, att det inte kommer att innebära en överdriven revolution för webben (även om det kan bli en stor förbättring) och att det går bra att knyta Ajax till existerande teorier inom mjukvaruarkitektur.

Nyckelord: Ajax, XMLHttpRequest, JavaScript, klient-server, asynkron, webb

Författare: Andreas Wahlin

Handledare: Kjell Engberg

Magisteruppsats, 20 poäng

1. Introduktion 4

1.1. Bakgrund 4

1.2. Frågeställning 4

1.3. Syfte och avgränsning 5

1.4. Termer 6

1.5. Målgrupp 6

1.6. Disposition 7

2. Metod 8

2.1. Metodval 8

2.2. Urvalskriterier 9

2.3. Felkällor 9

3. Teori 11

3.1. Webbstandarder samt separation av innehåll och utseende 11

3.2. DOM 14

3.3. Objektdetektering 16

3.4. Försynta skript 16

3.5. XML 17

3.6. XSLT 17

3.7. iframe 19

3.8. XMLHttpRequest 21

3.9. Ajax enligt Garrett 22

4. Analys 26

4.1. Ajax och Sommerville 26

4.2. Kritik mot Ajax 33

5. Diskussion och slutsats 36

6. Referenser 38

Figurer

Figur 1, förhållande mellan HTML och CSS 13

Figur 2, olika utseende (CSS) men samma innehåll (HTML). Skärmdumpar från CSS Zen Garden 14

Figur 3, exempel på ett DOM-träd 15

Figur 4, två iframes 20

Figur 5, den traditionella modellen för webbapplikationer, och Ajax-modellen 24

Figur 6, skillnad i tidslinje mellan den traditionella modellen och Ajax-modellen 25

Figur 7, Ajax-motorn som distributionsmodell 28

Figur 8, sammanfattning av de olika modellerna för arkitekturdesign i Ajax-modellen 29

Figur 9, hur Ajax-modellen rör sig mellan tunn och fet klient beroende på implementation 31

Tabeller

Tabell 1, sammanfattning av komponentdesign 27

Tabell 2, sammanfattning av Ajax design 32

Tabell 3, jämförande översikt mellan Garrett och Apple 34

1. Introduktion

1.1. Bakgrund

Webben har vuxit explosionsartat, sedan den blev allmänt känd runt 1996[1]. Från början bestod webben mest av föga grafiskt tilltalande sidor som inte var mycket mer än vanliga tryckta källor, exempelvis en restaurangmeny, tillgängliga genom en dator. Nu har webben vuxit och utökats så att man till exempel kan se på film, boka resor eller biljetter, handla eller delta i auktioner, söka enorma databaser av information eller till och med läsa sin e-post på förhoppningsvis estetiskt tilltalande sätt. Webben har kort sagt blivit enormt mycket rikare, både innehållsmässigt, utseendemässigt och funktionsmässigt, och det på bara ett årtionde.

Med den nya funktionaliteten närmar sig webben stundtals allt mer de applikationer (program) som återfinns i persondatorer. Det finns redan nu flera e-postklienter, till exempel Gmail[2]. Till och med en ordbehandlare[3] är under utveckling. E-postklienter och ordbehandlare var tidigare program man bara fann som traditionella applikationer på en persondator.

Som alla webbanvändare vet finns det dock en kraftig begränsning associerad med webben; ständiga väntetider. Man måste vänta en kortare stund mellan varje länk man klickar på eftersom webbläsaren behöver hämta information från servern, en begränsning som vanliga applikationer inte har på samma sätt. Även om en traditionell applikation ibland kräver väntetider, så krävs det ingen märkbar tid i en traditionell e-postklient för att öppna ett redan hämtat brev, och däri ligger en stor skillnad mellan traditionella applikationer och webben.

1.2. Frågeställning

Den 18:e februari 2005, presenterade Jesse James Garrett en artikel där han myntade termen Ajax, som står för *Asynchronous JavaScript + XML*.[4] Garrett anser att Ajax *"represents a fundamental shift in what's possible on the Web"*, han påstår även att det är *"... a new approach to web applications that we at Adaptive Path have been calling Ajax."*. Ajax påstås alltså vara något både revolutionerande och nytt, något som kan förändra webben.

Tanken med Ajax är att användaren inte skall behöva vänta så mycket på information från servern. Vad detta innebära i praktiken är ett användargränssnitt som blir lika smidigt som i en traditionell applikation, där man inte behöva vänta en sekund efter varje klick man gör. Vad Garrett presenterar är således en teknologi som önskar minska, och kanske rent av eliminera, en av webbens största svagheter; väntetiden.

---

[1] *Wikipedia. (22 september 2005).* http://en.wikipedia.org/wiki/Internet

[2] http://www.gmail.com

[3] http://www.writely.com

[4] *Jesse James Garrett, Ajax: A New Approach to Web Applications (adaptive path, 18 september 2005),* http://www.adaptivepath.com/publications/essays/archives/000385.php

Peter-Paul Kochs första reaktion när han hörde talas om Ajax var *""What a silly name", and "Not really new, is it?"".*[5] Cameron Adams förstärker detta genom att säga att hämtning av data genom JavaScript har funnits länge, även om han erkänner att det inte använts så flitigt.[6] Således verkar varken Koch eller Adams hålla med Garrett om att Ajax representerar något nytt.

Oavsett om det är nytt eller inte kan Ajax ändå innebära en ny riktning för webben, Garrett själv hävdar åtminstone det. Koch är osäker och lutar mot att Ajax är en hype i slutet av sin artikel, även om han säger att ordentlig och seriös utveckling av Ajax kan leda till något bra.

Jason Fried säger att Ajax har potential att bli något bra och han verkar positiv, även om han inte sträcker sig så långt som att kalla det något fundamentalt nytt.[7] Fried nöjer sig med att säga att Ajax kan göra webben lite bättre. Adams tror inte att Ajax kommer att innebära någon större skillnad för webben rent generellt, eftersom nästan hela webben handlar om webbsidor, och Ajax bara gör någon större skillnad för webbapplikationer.

Garrett presenterar Ajax som ett sätt att bygga och kraftigt förbättra webbapplikationer, även om han inte går in så mycket på djupet på hur det skall fungera. För att bygga en applikation krävs en god grund att basera sin design på, men ingen artikel tar upp applikationsbyggande med Ajax på någon djupare nivå. Det finns dock många böcker som behandlar traditionell applikationsdesign, exempelvis Ian Sommervilles bok Software Engineering[8], som kan ha något att säga.

Detta ger upphov till bland annat dessa tre frågor som är lämpliga för en initial studie av Ajax:

- *Är Ajax något nytt och revolutionerande?*
- *Vad innebär Ajax för webben?*
- *Kan existerande teorier om mjukvaruarkitektur hjälpa oss med Ajax?*

1.3. Syfte och avgränsning

Syftet med uppsatsen är att utifrån artiklar undersöka Ajax och söka svar på de frågor som presenteras under frågeställningen. För att göra detta kommer även de byggstenar som utgör Ajax att behöva presenteras.

Målet är att få en helhetsbild för vad Ajax innebär, genom att sammanställa och kritisera åsikter från flera olika författare, med Garrett som utgångspunkt.

---

[5] *Peter-Paul Koch, Ajax, promise or hype? (QuirksMode, 13 mars 2005),* http://www.quirksmode.org/blog/archives/2005/03/ajax_promise_or.html

[6] *Cameron Adams, This is not another XMLHttpRequest article, (The Man in Blue, 2 mars 2005),* http://www.themaninblue.com/writing/perspective/2005/03/02/

[7] *Jason Fried, XMLHttpRequest, Ajax, and the customer experience, (Signal vs. Noise, 24 februari 2005),* http://www.37signals.com/svn/archives/001070.php

[8] *Ian Sommerville, Software Engineering 7th ed. (Addison-Wesley, 2004).*

Ingen ansats kommer att göras för att se hur man använder Ajax på bästa sätt eller hur man lämpligast bygger en Ajax-applikation, det är alltså ingen praktisk eller normativ uppsats. Sådana försök övervägdes, men togs bort bland annat på grund av tidsbrist, men också för att hålla fokus och inte bli för bred i frågeställningen.

1.4. Termer

De datatermer som används och som inte definieras i texten kommer främst att hämtas från Svenska datatermgruppen[9], övriga kommer att sättas eller översättas av författaren. Exempelvis hämtas termen *webbläsare* från Svenska datatermgruppen, medan termen *försynta skript* (unobtrusive scripting) inte gör det.

Webben kommer att användas för att generellt peka ut alla webbsidor i världen, och webbsida kommer att vara en individuell sida på webben. Webben kommer också att åsyfta de mer kulturella och abstrakta implikationerna av webben, precis som en bok kan anses vara en samling text, men också vara en samling idéer och perspektiv.

Termen webbapplikation används för en webbsida vars syfte är att ge en viss form av funktionalitet annan än att bara visa bilder och text, enligt Adams utläggning. Det som sägs generellt om webbsidor, kan också gälla webbapplikationer, såvida inte skillnader mellan webbsidor och webbapplikationer diskuteras. En webbplats är en samling webbsidor, men distinktionen kommer inte att spela någon viktig roll. HTML utesluter inte XHTML, såvida det inte specifikt anges.

1.5. Målgrupp

Målgruppen är webbutvecklare. Personer som bara önskar veta lite mer om Ajax bör läsa Garretts artikel istället. Uppsatsen går översiktligt igenom olika tekniker för webbutveckling, och kan därför vara en startpunkt för vidare studier, men lämpligare startpunkter borde gå att finna. En rent syntaktisk riktlinje för förståelse är nedanstående kod, om den förstås så borde resten av uppsatsen gå att förstå.

[9] http://www.nada.kth.se/dataterm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
<html>
<head>
  <title>Test</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <script type="text/javascript">
   function skrik(vad) {
    alert(vad);
   }
  </script>
  <style type="text/css">
   body {
    background-color: #ff00ff;
   }
  </style>
</head>
<body onload="skrik('Starta');">
  <h1>En rubrik</h1>
  <p>
   en paragraf
  </p>
</body>
</html>
```

1.6. Disposition

Kapitel 1 beskriver introduktion, frågeställning och liknande introducerande delar.

Kapitel 2 diskuterar metodiken och tar även upp saker som val av artiklar samt eventuella felkällor.

Kapitel 3 går komprimerat och översiktligt igenom de olika teknologier och tekniker som behövs för att kunna förstå diskussionen runt Ajax. Här sammanställs fakta från olika källor. Kapitlet avslutas med en återgivning av Garretts hela artikel.

Kapitel 4 börjar med att se hur man kan knyta Ajax till existerande teorier inom mjukvaruarkitektur, så som de presenteras av Sommerville. Kapitlet fortsätter sedan med att utifrån Kochs artikel “Ajax, promise or hype?”, kritisera och analysera Ajax. Kritikdelen följer dispositionen i Kochs artikel, och tar upp åsikter från andra författare som refereras i artikeln.

Kapitel 5 väver ihop diskussion och slutsats. Här sammanfattas svaren på frågorna från frågeställningen, blandat med författarens egna åsikter. Svaren har tidigare kommit fram i kapitel 4.

Kapitel 6 avslutar uppsatsen med att lista referenserna i sin helhet.

2. Metod

2.1. Metodval

Väldigt översiktligt kan man säga att det finns tre typer av rapporter; den kvantitativa, den kvalitativa samt forskningsöversikten.[10] Det som här benämns kvantitativ, är vad Backman kallar traditionell.

Kvantitativ forskning rör, som namnet antyder, kvantiteter med data. Den grundläggande tanken är att det finns en objektiv verklighet, som går att mäta objektivt och få säker kunskap om, eller ifrån.

Kvalitativ forskning behandlar verkligheten som något mer subjektivt, och inkluderar ofta forskaren själv som en del av verkligheten, i rak motsats till det kvantitativa synsättet.

Den här uppsatsen har begagnat sig främst av det kvalitativa synsättet, och stannat vid den litteraturgranskande delen. För att komma fram till svar på frågorna, har enbart litteratur lästs, ingen empiri har bedrivits. Man skulle även kunna se uppsatsen som en kvalitativ översikt, om man anser att de olika granskade artiklarna är forskningsmaterial. Med det förbehållet att uppsatsen inte på något sätt är heltäckande.

Genom analys av Garretts modell om hur Ajax fungerar, har existerande teorier inom mjukvaruarkitektur sammankopplats med Ajax, för att vidare söka utröna Ajax natur. Det passar in på Backmans beskrivning av fallstudier, där Ajax är ett fenomen och befinner sig i kontexten av olika webbutvecklingsteorier.

Metoden består i att författaren söker upp och läser igenom en representation av vad som skrivits om Ajax, för att sedan så objektivt som möjligt analysera och ställa de olika åsikterna mot varandra. De flesta utvalda artiklar har kommit till under en kort tidsrymd, som svar och kommentarer på varandra. När uppsatsarbetet började fanns inga böcker ägnade åt Ajax, även om till exempel Stuart Langridge[11] nämner det kort i sin bok.

Andra metodväl hade inte varit särskilt lämpliga, eftersom syftet var att reda ut de olika åsikter som kom fram i de artiklar som kom till runt tiden då termen Ajax myntades. Empiri blir i det här fallet överflödig. Intervjuer hade kunnat användas, men författarnas åsikter borde vara klara och tydliga i deras artiklar.

Undersökningen gick bra. Det kom fram många poänger i de olika artiklarna som belyste frågeställningen, det var också intressant att se mer allmängiltiga teorier från Sommerville fungera på Ajax. Det var inte förväntat att Garretts artikel skulle komma med så lite nytt rent tekniskt och designmässigt.

---

[10] Jarl Backman, Rapporter och uppsatser (Lund: Studentlitteratur, 1998).

[11] Stuart Langridge, Modern Web Design Using JavaScript & DOM, (Collingwood: SitePoint Pty. Ltd., 2005), 277.

2.2. Urvalskriterier

Litteraturgranskningen utgår från Garretts artikel, där Ajax presenterades som term för första gången. Allt annat kommer att jämföras mot den artikeln. Garrett är känd för arbete inom användbarhet på webben och har skrivit boken “The Elements of User Experience”. Kritiken mot Garrett tar sin avstamp från Kochs artikel “Ajax, promise or hype?”, och dispositionen av kritiken följer också någorlunda den artikelns upplägg.

Koch refererar i sin artikel till flera andra artiklar, som alla tar sig en kritisk kik på Ajax, eftersom de refereras från Kochs artikel anses de vara lämpliga som material. Koch själv är en auktoritet på JavaScript samt olika webbläsares skilda beteenden. Han driver sin egen webbplats som heter QuirksMode.org[12]. Han är känd för att skriva artiklar av kritisk eller undersökande natur. Uppsatsens författare läser regelbundet hans artiklar och det var så Kochs artikel hittades från första början.

Anledningen till att Kochs artikel valdes är att den presenterar en ordentlig och tämligen heltäckande kritik av Ajax till skillnad från andra funna artiklar. Medan andra artiklar oftast tar upp en eller två poänger till diskussion, försöker Koch att sammanföra många punkter och dessutom refererar han till andra artiklar, vilket ger en vägledning i artikelsökningen.

Sommervilles bok “Software Engineering” valdes som representation för traditionella teorier om mjukvaruarkitektur. Den täcker upp stora områden, och är nu inne på sin sjunde upplaga, vilket gör den till en pålitlig källa. Någon annan bok än Sommervilles borde inte gjort någon större skillnad i slutändan, eftersom Sommerville tar upp tämligen allmängiltiga saker. Han presenterar i stor utsträckning fakta, inte åsikter.

Till teoridelen användes främst två böcker från Sitepoint[13] samt flera artiklar från A List Apart[14]. Sitepoint har sedan gammalt samlat på sig rikligt med material för webbutvecklare, både tillgängligt via webben och via deras böcker som går att köpa från deras webbsida. A List Apart har också en lång tradition av att erbjuda material till webbutvecklare, deras artiklar kan anses vara av god kvalitet eftersom de har en granskningsprocess för allt inskickat material.

2.3. Felkällor

I och med att uppsatsen är kvalitativ spelar den mänskliga faktorn en stor roll. Nästan allt använt material är på engelska, medan uppsatsen skrivits på svenska. Något viktigt kan ha gått förlorat i översättningen och tolkningen.

I och med att uppsatsen är en litteraturanalys är det viktigt att författaren förblivit objektiv och verkligen tolkat artiklarna för vad de är och inte lagt in för mycket av sina egna idéer. En artikel kan även ha missförstått, varpå en poäng kan ha gått förlorad eller blivit förvriden.

---

[12] http://www.quirksmode.org/

[13] http://www.sitepoint.com/

[14] http://www.alistapart.com/

Kopplingen mellan Ajax så som Garrett lägger fram det, och de teorier som presenteras av Sommerville, har även de gjorts helt genom litteraturanalys. Det finns inga experiment som kan bekräfta de slutsatser som dragits, utan allt har baserats på jämförelser av skrivet material.

Kritiken av Ajax är nära knuten till Koch och rör sig bara kring Kochs artikel samt till de artiklar han refererar. Det kan finnas andra små öar av information ute på webben som missats, med helt andra poänger och infallsvinklar än de som kommit fram.

Slutsatsen skulle ha kunnat bli annorlunda om andra artiklar använts. Andra idéer skulle ha kunnat tas upp och författare skulle ha kunnat vinkla det annorlunda. Uppsatsen lutar sig således mot att Koch har refererat ett kvalitativt och heltäckande material av artiklar.

3. Teori

3.1. Webbstandarder samt separation av innehåll och utseende

Att börja skapa en webbsida är relativt lätt, det räcker med att göra en enkel textfil, vanligtvis med en filändelse på ".html" eller ".htm", som man sedan skriver lite text i. Öppnar man denna fil i en webbläsare kommer texten man skrev in att visas i webbläsaren. Att det är så lätt att göra webbsidor har bidragit till att webben har vuxit så fort som den har gjort, men också till det stundtals tveksamma innehållet.

I slutet av 90-talet så tolkades webbsidor väldigt olika av de webbläsare som fanns. Taggar blev visade lite olika, och det gick dessutom ibland att komma undan med strukturella konstigheter i webbsidorna. Alla vet att standarder är trevliga, och på senare tid har webbläsare mer och mer anpassat sig efter World Wide Web Consortium[15], eller W3C som de ofta kallas. W3C publicerar standarder som säger hur webbläsare skall läsa av webbsidor, till exempel HTML-standarden[16].

Det går att berätta för webbläsaren vilken typ ens webbsida är, och genom det vilken standard ens webbsida håller sig till. Det gör man genom att först i dokumentet inkludera en så kallad DOCTYPE-tagg, som främst är en länk till en DTD.[17]

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
  [... resten av dokumentet]
```

En DTD är en beskrivning av vilka element som är tillåtna för ett dokument. I praktiken innebär det vilka taggar som får finnas, och vilka attribut en tagg får ha. Det går även att skriva sin egen DTD, det är vanligt när man skriver XML-dokument eftersom XML inte kommer med en på förhand definierad uppsättning taggar, till skillnad från HTML och XHTML.[18]

Så genom att ange vad det är för dokument man har, anger man för webbläsaren hur man vill att den skall tolka webbsidan. Man kan då förvänta sig att om det är en någorlunda modern webbläsare kommer det att se likadant ut i den som man tänkt sig, eftersom alla moderna webbläsare skall följa samma standard. En annan fördel är också att man kan testa sin kod genom att köra den genom ett valideringsverktyg[19], och få en rapport över vilka fel man gjort, dvs. vad som strider mot den standard man uppget att man följer. Det är ett kraftfullt verktyg för att försäkra sig om att man håller sig till standarden.

---

[15] http://www.w3.org/

[16] *http://www.w3.org/MarkUp/*

[17] *Jeffrey Zeldman, Fix Your Site With the Right DOCTYPE!, (12 april 2002),* http://www.alistapart.com/articles/doctype

[18] *Thomas Mayer, XML Web Development With PHP, (Collingwood: SitePoint Pty. Ltd., 2005), 59-61.*

[19] *Langridge, 3-4.*

Att följa ovanstående är att följa bokstaven i lagen om webbstandarder, men det finns en anda också; separation av innehåll och utseende.[20]

De flesta, för att inte säga alla, HTML-taggar kommer så att säga med ett utseende. En em-tagg får ofta text att bli kursiv, en blockquote-tagg får text att bli indragen en bit, och så vidare. Att separera innehåll och utseende innebär dock att man inte använder taggar för att skapa utseende, utan för att skapa innehåll. En blockquote-tagg skall till exempel inte tolkas som “få texten att bli indragen en bit”, utan den skall tolkas som “här är ett citat”.[21] Alla taggar skall ha vad man ibland kallar för en semantisk mening; taggen skall berätta något om vilken typ av innehållen den har.

Några effekter av dessa tankebanor är att man inte längre skall använda b- och i-taggar för fet respektive snedställd text, utan man skall använda strong- och em-taggar för markerad respektive emfatisk text. Notera att en em-tagg inte innebär snedställd text, utan emfatisk, om emfasen sedan rent utseendemässigt innebär att texten snedställs är det en annan sak. Det handlar alltså om att beskriva vad det är för typ av text, inte vilket utseende den skall ha.

HTML-filen skall alltså skapas med fokus på innehåll, alla taggar skall vara meningsfulla och berätta något om innehållet. De vanliga br-taggarna bör givetvis undvikas eftersom de bara ger utseende, och inget innehåll alls. Sedan, när man skapat en god HTML-fil med fokus på innehåll, då skapar man en CSS-fil som ger liv och utseende åt innehållet.

En av milstenarna i kampen för att separera innehåll från utseende har varit att få bort de populära tabell-designerna av webbsidor. Även om det inte syns, så är många webbsidor uppbyggda av en stor och osynlig tabell, som ger layouten för sidan. Vanligtvis är vänsternavigeringen en cell, själva brödtexten en cell och eventuella högermarginaler en tredje cell. Sedan är varje sådan cell ofta indelad i celler, i extremfall är varje rad i navigeringen en egen cell. Exempelvis kan GU:s föregående mall för webbsidor nämnas, där varje sida är en tabell, som dessutom är satt till en bestämd vidd på 643 pixlar[22]. Den nya designen däremot, innehåller inga tabeller för att styra layouten.

Istället för att använda tabeller för att skapa layouten av sidan, skall man istället kapsla in innehållet med div-taggar. Med hjälp av CSS sätter man sedan utseendet för dessa div-taggar för att skapa layouten.[23] Navigeringsmenyer skall inte skapas av celler i en tabell, utan istället anges som listor.[24]

---

[20] *Jeffrey Zeldman, From Table Hacks to CSS Layout: A Web Designer’s Journey, (16 februari 2001),* http://www.alistapart.com/stories/journey/

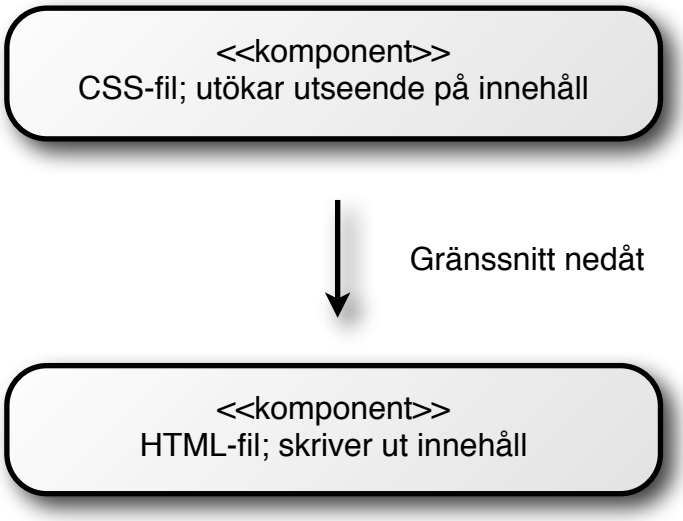
[21] *Langridge, 4-5.*

[22] http://wwwhost.gu.se/wwwguse_pdf/profil/GU_Grafisk_profil_02.pdf

[23] *Ryan Brill, Creating Liquid Layouts with Negative Margins (A List Apart, 15 juni 2004),* http://www.alistapart.com/articles/negativemargins

[24] *Mark Newhouse, CSS Design: Taming Lists (A List Apart, 27 september 2002),* http://www.alistapart.com/articles/taminglists

Man kan se HTML-filen och CSS-filen som två komponenter i vad Mathiassen et al. kallar för skiktad arkitektur[25], där HTML-filens funktion är att visa upp innehållet, medan CSS-filens funktion är att omforma och utöka innehållets utseende. Sett ur den vinkeln ligger HTML-filen under CSS-filen, som har ett gränssnitt nedåt mot HTML-filen.



Figur 1, förhållande mellan HTML och CSS

Effekten av det blir att om man ändrar om HTML-filen, är det inte längre säkert att CSS-filen kan fullfölja sin uppgift och ändra om utseendet på sidan. Däremot kommer HTML-filens innehåll alltid att kunna presenteras. Det är en viktig punkt, för det gör att om man separerar innehåll och utseende kommer innehållet alltid att kunna presenteras. Gamla webbläsare som inte klarar av att läsa CSS-filen, kommer ändå att kunna presentera innehållet, och läsaren får all information. Webbsidan har blivit med andra ord blivit tillgänglig.

Andra fördelar blir att så länge man i förväg kommer överens om hur webbsidan skall se ut, så kan arbetet på HTML- och CSS-filen utföras parallellt. Koden för webbsidan blir vanligtvis mindre, dessutom ligger CSS-filen kvar i webbläsarens minne och behöver inte hämtas om för varje ny sida. Detta sparar bandbredd som ger kortare laddningstider och sparar pengar. Jeffrey Veen tar upp dessa fördelar i “The Business Value of Web Standards”[26].

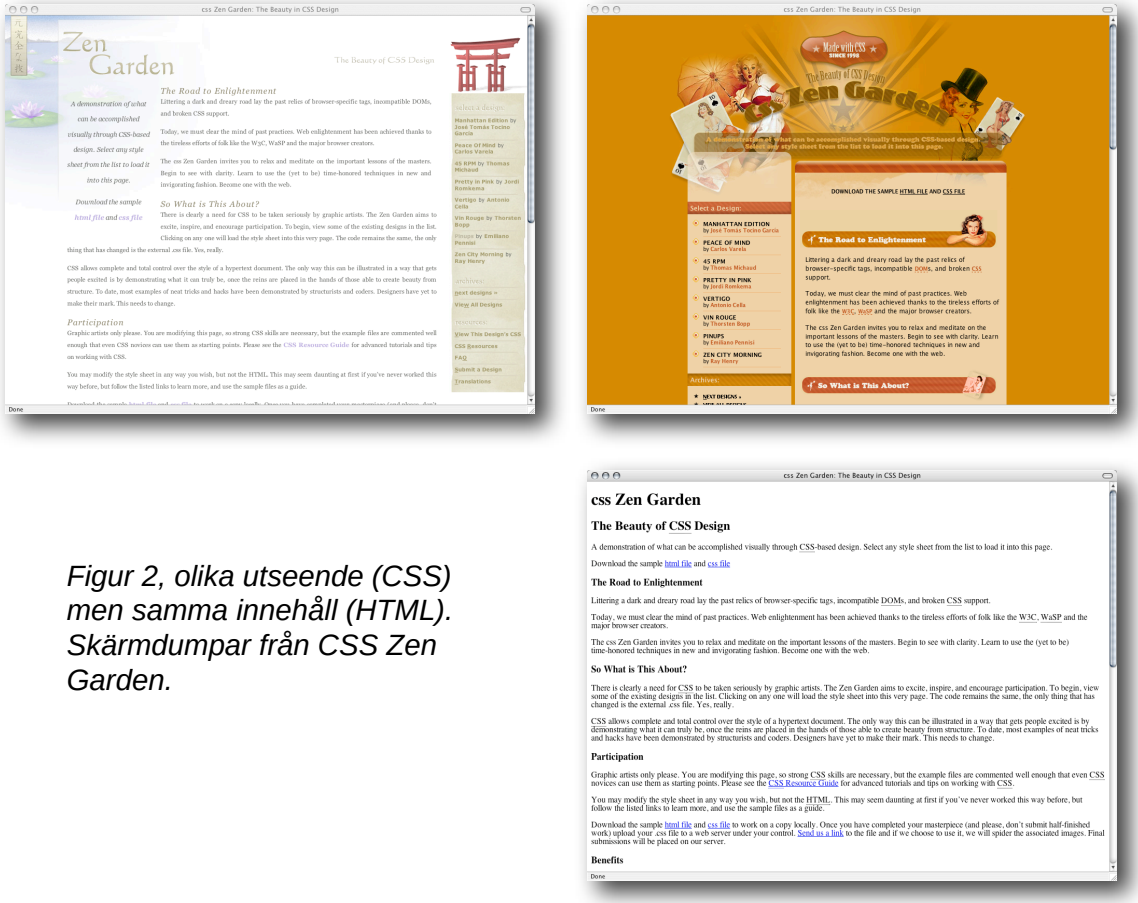
Ytterligare en fördel som designers ser på med blida ögon är det faktum att man kan ha flera stycken CSS-filer, och således få hela utseendet på en webbsida att ändras drastiskt beroende på vilken CSS-fil man använder sig av. CSS Zen Garden[27] är en sida vars enda syfte är att visa flexibiliteten hos CSS. Figur 2 visar tre skärmdumpar från CSS Zen Garden, två med olika CSS-filer, och en tredje utan CSS-fil. HTML-filen i alla tre skärmdumpar är samma, vilket gör att alla har samma text, men väldigt olika utseende.

---

[25] *Lars Mathiassen et al., Objektorienterad analys och design (Lund: Studentlitteratur, 2001), 227-228.*

[26] *Jeffrey Veen, The Business Value of Web Standards (adaptive path, 18 september 2003),* http://www.adaptivepath.com/publications/essays/archives/000266.php

[27] http://www.csszengarden.com/



Figur 2, olika utseende (CSS) men samma innehåll (HTML). Skärmdumpar från CSS Zen Garden.

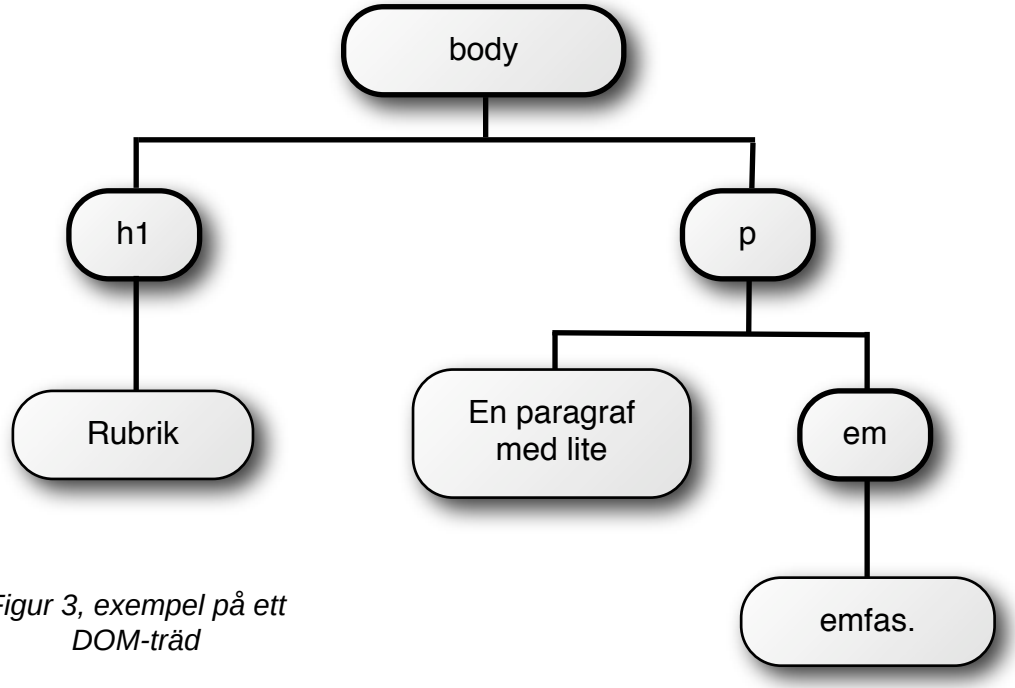
3.2. DOM[28]

En webbsida är ett HTML-dokument, DOM:en (dokumentobjektmodellen) presenterar detta dokument som ett träd, där varje tagg och varje textmassa finns representerad, och varje tagg har sina attribut. Exempelvis skulle följande pseudo-webbsida;

```
<body>
  <h1>Rubrik</h1>
  <p>
    En paragraf med lite <em>emfas.</em>
  </p>
</body>
```

representeras enligt figur 3.

[28] *Langridge, "The Document Object Model".*



Figur 3, exempel på ett DOM-träd

Här syns också vikten av validerad och korrekt HTML, annars vet inte webbläsaren hur trädet skall byggas upp och det kan se ut lite hursomhelst. DOM:en beskrivs av W3C såhär:

"*The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.*"[29]

Vad det innebär är att man med JavaScript kan påverka trädet, och således hela webbsidan. I exemplet ovan skulle man till exempel kunna gå in och ändra om em-taggen till en strong-tagg, eller ändra om class-attributet på em-taggen, för att på så sätt påverka hur texten "emfas." visas.

Man kan alltså lägga till, ändra och ta bort vilka enheter som helst från en webbsida dynamiskt, utan att behöva ladda om sidan. Att på det sättet ändra om en webbsida kallas ibland DHTML (dynamisk HTML), eller DOM-skriptning, lite beroende på vad man gör och vem man lyssnar på.[30] Även XML-dokument har ett DOM-träd som gör att man kan ändra om dem.

[29] http://www.w3.org/DOM/#what

[30] *Peter-Paul Koch, DHTML != DOM, (QuirksMode, 16 januari 2005),* http://www.quirksmode.org/blog/archives/2005/01/dhtml_dom_1.html

3.3. Objektdetektering

JavaScript har utvecklats i takt med att webbläsare blivit mer och mer avancerade. Det innebär att viss funktionalitet som finns i nyare webbläsare saknas i äldre. För att lösa det kan man läsa av en sträng som berättar om webbläsaren, och sedan köra olika kodsnuttar som stöder olika webbläsare. Det är en väldigt riskabel process, dels för att vissa webbläsare utger sig för att vara andra, och dels för att det är väldigt jobbigt att hålla sig a jour med alla webbläsare och deras egenheter.[31]

Vad man istället skall göra är att använda objektdetektering (object detection), och helt enkelt ta reda på om webbläsaren stöder en viss funktionalitet genom if-satser. Man kan testa vilka variabler och funktioner webbläsaren stöder. Oftast handlar det om att se om en webbläsare klarar av avancerad DOM-skriptning, men också om vissa variabler som gör samma sak men heter olika beroende på webbläsare.[32]

3.4. Försynta skript

Försynta skript (unobtrusive scripting) är väldigt likt separering av innehåll och utseende. Tanken är att skripten inte skall vara en del av innehållet, utan läggas ovanpå så att de blir tillgängliga för de webbläsare som klarar av dem. Man skall lägga upp ett gränssnitt i innehållet, som sedan skripten kan haka fast sina händelser vid. Det är vanligt att man använder sig av id- eller class-attributen så att skripten vet var de skall lägga in sina händelser.[33]

Man skall inte göra som nedan, där JavaScript-funktionaliteten är en del av innehållet,

```
<a href="http://www.gu.se" onmouseover="window.status = 'Gå till ' +
this.href;">GU</a>
```

utan istället lösa det såhär:

```
<a href="http://www.gu.se" class="navigering">GU</a>

<script type="text/javascript">
  var nav = document.getElementsByTagName("a");
  for (var i = 0; i < nav.length; i++)
    if (nav[i].className == "navigering")
      nav[i].onmouseover = new Function("window.status = 'Gå till ' +
this.href;");
</script>
```

Även om det blir mer kod, så blir det återanvändbar kod, eftersom JavaScript-funktionaliteten blivit en egen komponent. Det räcker med att skriva en JavaScript-fil och sedan länka in den på alla sidor där man vill ha beteendet, som sedan appliceras på rätt element. Länkade JavaScript-filer sparas dessutom i webbläsarens minne och minskar därmed laddningstiden.

---

[31] *Langridge, 76-77.*

[32] *Langridge, 77-80.*

[33] *Langridge, 8.*

3.5. XML

XML står för eXtensible Markup Language och är ett metaspråk för information.[34] Det är likt HTML på det sätt att det använder sig av taggar, men till skillnad från HTML får man i XML helt själv bestämma sina taggar, och vad de betyder. XML är tänkt att vara ett plattformsoberoende språk med fokus på att utbyta information mellan olika former av klienter eller servrar. Eftersom informationen lagras som ren text är det också möjligt för människor att relativt enkelt gå in och ändra informationen direkt i filen. Ett exempel på ett e-brev kan se ut såhär:

```
<?xml version="1.0"?>
<brev>
  <från visa="Andreas Wahlin">guswahand@student.gu.se</från>
  <till visa="Kjell Engberg">engberg@informatik.gu.se</till>
  <läst/>
  <innehåll>
   Uppsatsen går framåt!
  </innehåll>
</brev>
```

Alla taggar måste stängas, och taggar som inte har något innehåll, som “läst” i exemplet, måste avslutas med ett snedstreck för att visa att de är stängda. Taggar får ha attribut, som innehåller extra information om taggen.

Med en DOCTYPE-tagg kan man styra vilka element som får ingå i XML-filen, och på det sättet försäkra sig om att filen är korrekt strukturerad. Det är idealiskt om man skickar mycket information fram och tillbaka mellan olika klienter, där vissa är människor och vissa är applikationer. Även om alla ändrar om innehållet, kan man lätt försäkra sig om att strukturen är korrekt och läsbar för alla genom att validera XML-filen.

3.6. XSLT

XSLT står för eXstensible Stylesheet Language Transformations och är ett sätt att förvandla XML-dokument till andra typer av dokument.[35] Ett användningsområde är att konvertera olika XML-dokument med information till HTML för en snyggare presentation. Förutom konvertering kan man även utföra sorteringar, beräkningar och en hel rad saker. Här följer ett exempel på vad man kan göra med XML-filen som beskrev ett brev.

---

[34] *Myer, 2.*

[35] *Myer, 44.*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="brev.xslt"?>
<brev>
  <från visa="Andreas Wahlin">guswahand@student.gu.se</från>
  <till visa="Kjell Engberg">engberg@informatik.gu.se</till>
  <läst/>
  <innehåll>
   Uppsatsen går framåt!
  </innehåll>
</brev>
```

Notera den nya raden näst högst upp som anropar “brev.xslt”, det är den XSLT-filen som skall användas för att tolka XML-filen. Nedan följer XSLT-filen.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">

  <xsl:template match="/brev">
   <html>
    <body>
     <xsl:if test="läst">
      <em>Läst</em><br/>
     </xsl:if>
     <xsl:apply-templates/>
    </body>
   </html>
  </xsl:template>

  <xsl:template match="från">
   Från: <a><xsl:attribute name="href">mailto:<xsl:apply-templates/></
xsl:attribute><xsl:value-of select="@visa"/></a><br/>
  </xsl:template>

  <xsl:template match="till">
   Till: <a><xsl:attribute name="href">mailto:<xsl:apply-templates/></
xsl:attribute><xsl:value-of select="@visa"/></a><br/>
  </xsl:template>

  <xsl:template match="innehåll">
   <p><xsl:apply-templates/></p>
  </xsl:template>

</xsl:stylesheet>
```

Först skapas en mall av själva roten i brevet. I denna mall skrivs HTML-kod ut, och en test som kontrollerar om en läst-tagg finns definierad eller inte utförs. Om läst-taggen finns skrivs det som står mellan test-taggarna ut. Sedan kommer xsl-taggen `apply-templates`, som drar in alla mallar man skapat.

Sist skapas mallarna för de andra taggarna, som kommer att ingå i brevmallen eftersom de hämtas in i brevmallen med xsl-taggen `apply-templates`. Från- och tillmallen är nästan likadana, de börjar med att skriva ut lite text, för att sedan inkludera en länk-tagg.

Länk-taggen byggs upp lite speciellt eftersom det är en HTML-tagg som dessutom skall ha ett attribut; e-postadressen. För adressen dras värdet från taggen in med en `apply-templates`, för vad som skall visas som text hämtas värdet av visa-attributet in. Sist hämtas innehålls-taggen in och sätts inom en p-tagg. Resultaten blir HTML-kod som ser ut såhär:

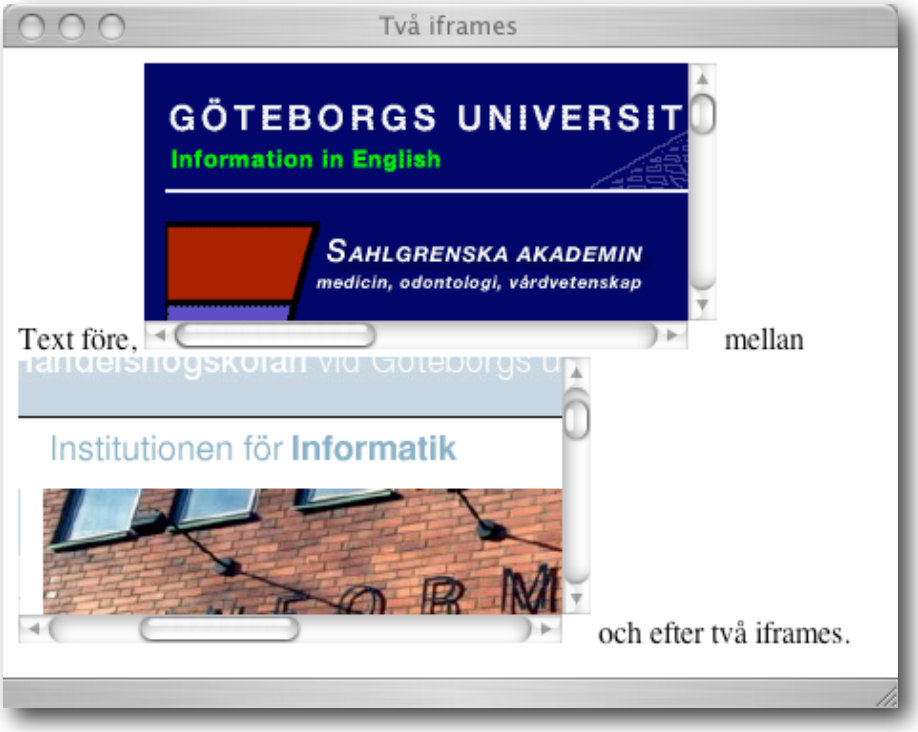
```
<html>
  <body>
    <em>Läst</em><br/>
    Från: <a href="mailto:guswahand@student.gu.se">Andreas Wahlin</a><br/
>
    Till: <a href="mailto:engberg@informati.gu.se">Kjell Engberg</a><br/>
    <p>
      Uppsatsen går framåt!
    </p>
  </body>
</html>
```

3.7. iframe

Iframe-taggen är en rätt speciell tagg, den fungerar som en ram (frame) i det vanliga flödet i HTML-dokumentet, därav namnet som kan läsas ut som inline frame. Man kan alltså ange en iframe-tagg var man vill och innehållet i den kommer att hämtas från en annan sida. Följande HTML-kod:

```
<body>
  Text före, <iframe src="http://www.gu.se"></iframe> mellan <iframe
src="http://www.informatik.gu.se"></iframe> och efter två iframes.
</body>
```

kommer att generera ett resultat som i figur 4.



Figur 4, två iframes.

Precis som med vanliga ramar kan man också ha länkar som ändrar om en iframes innehåll, och precis som vanliga taggar kan man ändra om deras utseende med CSS. De exakta specifikationerna står att finna hos W3C[36].

```
<body>
  <iframe name="iframe" src="http://www.gu.se" style=”border: 1px solid
red;”></iframe>
  <a href="http://www.informatik.gu.se" target="iframe">ändra</a>
</body>
```

Genom JavaScript kan en iframe referera bakåt till det dokumentet som skapade den, och därmed till alla JavaScript-funktioner som finns där. Det kan till exempel användas för att signalera till föräldern att iframen laddat klart.[37]

```
<script type=”text/javascript”>
  window.parent.funktionsNamn(argument);
</script>
```

Eftersom en iframe också kan ha vilken URL som helst, kan man givetvis skicka instruktioner som vanligt till servern i stil med `formvalidering.php?universitet=gu`. Man kan också skapa en iframe genom DOM-skriptning och sedan gömma undan den, så att användaren aldrig vet att den existerar. Med DOM-skriptning kan man sedan hämta information från en iframe, och använda den för att influera andra delar av sidan, även om det är en aning bökigt.

---

[36] http://www.w3.org/TR/REC-html40/present/frames.html#h-16.5

[37] *Langridge, 201-205.*

Allt detta gör att man med lite kreativt tänkande kan få en iframe att agera som en buffert mellan klienten och servern, som tillåter dynamiskt innehåll att flöda åt båda hållen utan att behöva ladda om hela sidan eller ens störa användaren.

Varje förändring hos iframen noteras av webbläsarens historik, vilket gör att bakåtknappen i en webbläsare inte nödvändigtvis beter sig som användaren tänkt, särskilt inte med gömda iframes. Det är möjligt att gå runt detta beteende genom att ändra iframens innehåll med `iframeReferens.location.replace(url)` istället. Den metoden kräver dock JavaScript och är alltså inte ett beteende man kan lita på, såvida man inte skapat iframen med DOM-skriptning. Man kan alltså välja om iframen skall ingå i webbläsarens historia eller inte. Det är viktigt att tänka på ur användbarhetssynpunkt hur man väljer att hantera webbläsarens historia.[38]

Det är värt att notera att iframes inte stöds i en strict DOCTYPE, utan bara en transitional eller frameset DOCTYPE, det gäller för både HTML och XHTML. Iframe stöds av webbläsare långt bak i tiden och kan således betraktas som något stabilt, även om felaktig hantering med JavaScript givetvis kan leda till katastrof.

3.8. XMLHttpRequest

Till skillnad från iframe-taggen så byggdes XMLHttpRequest från grund och botten som ett sätt för klienten att kunna kommunicera asynkront med servern. Med asynkront menas att man inte behöver vänta på svar innan man kan gå vidare. XMLHttpRequest implementerades först av Microsoft som ett ActiveX-objekt för deras webbläsare Internet Explorer, men har sedan dess kopierats av de flesta moderna webbläsare[39]. Diverse säkerhetsregler gäller, som främst innebär att man inte kan hämta innehåll från någon annan server än den ursprungliga.

XMLHttpRequest-objektet har fyra viktiga bitar; `onreadystatechange`, `readyState`, `open` samt `send`. `0pen`-funktionen tar tre argument; först vilken sorts HTTP-förfrågan man vill göra, vilket i praktiken antingen är “GET” eller “POST”, sedan anger man en adress till själva sidan, och sist anger man `true` eller `false` beroende på om anropet skall vara asynkront eller inte. Asynkront är vanligen att föredra, eftersom webbläsaren annars låser sig tills förfrågningen utförts, precis det man inte vill. Med `send`-funktionen initierar man sedan förfrågningen.[40]

När XMLHttpRequest-objektet börjat göra sitt anrop, så tickar `readyState`-värdet upp, när anropet är färdig så är värdet 4. Det är inte säkert att webbläsaren passerar alla värden på väg till 4, men alla webbläsare slutar i alla fall på 4.[41] Varje gång `readyState`-värdet tickar upp så anropas funktionen som är lagrad i `onreadystatechange`, så det är där man lägger koden för vad som skall hända när XMLHttPRequest-anropet är slutfört. När anropet sedan är klart hamnar resultatet i `responseText` eller `responseXML` beroende på vilken typ av resurs man hämtade.

---

[38] *Langridge, 206.*

[39] *Apple Developer Connection, Dynamic HTML and XML: The XMLHttpRequest Object, (Apple, 24 juni 2005),* http://developer.apple.com/internet/webcontent/xmlhttpreq.html

[40] *Langridge, 226-228.*

[41] *Peter-Paul Koch, XMLHTTP notes: readyState and the events, (QuirksMode, 21 september 2005),* http://www.quirksmode.org/blog/archives/2005/09/xmlhttp_notes_r_2.html

```
<script type="text/javascript">
  var xmlhttp = false;

  // initiera XMLHttpRequest för alla olika webbläsare
  try {
    xmlhttp = new XMLHttpRequest();
  }
  catch (e1) {
    try {
      xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch (e2) {
      try {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      catch (e3) { }
    }
  }

  if (xmlhttp) {
    xmlhttp.open("GET", "http://www.gu.se", true); // öppna en resurs
    // sätter funktionen som anropas vid statusförändring
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4) {
        if (xmlhttp.status == 200)
          alert(xmlhttp.responseText); // en alert-ruta med hela HTML-koden
som resultat
        else
          alert("Det blev bekymmer:\n" + xmlhttp.statusText);
      }
    }
    xmlhttp.send (null); // sätter igång anropet
  }
</script>
```

Bara för att objektet råkar heta XMLHttpRequest, innebär det inte att man är begränsad till att enbart hämta XML. Man kan hämta XML, HTML eller till och med vanlig text från en textfil på servern. XMLHttpRequest är alltså ett kraftfullt verktyg för att hämta information av olika slag, med den enda begränsningen att det inte går att hämta information från en annan server.

3.9. Ajax enligt Garrett

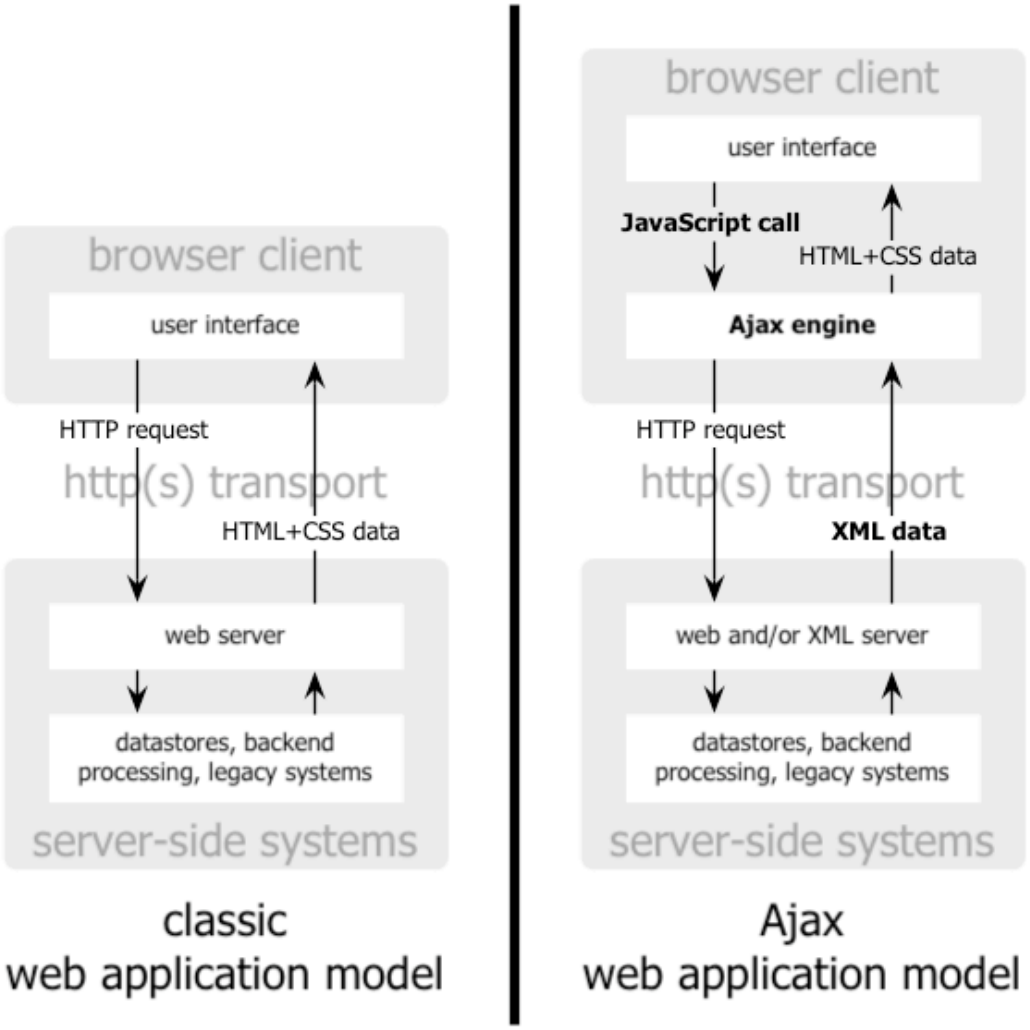
Det här avsnittet är en sammanfattning av Garretts artikel “Ajax: A New Approach to Web Applications”.

Ajax står för Asynchronous JavaScript + XML och är inte en teknologi i sig, utan en samling samverkande teknologier. Ajax består av:

- *standardiserad presentation som använder sig av XHTML och CSS*
- *dynamiskt utseende och interaktion med DOM:en*
- *utbyte och manipulering av data med XML och XSLT*
- *asynkron datahämtning med XMLHttpRequest*
- *och JavaScript för att binda ihop allt*

I den traditionella modellen för en webbapplikation så genererar den mesta interaktionen från användaren en HTTP-förfrågan tillbaka till servern. Servern hämtar sedan information, utför beräkningar, kommunicerar eventuellt med andra system och skickar sedan tillbaka en webbsida till klienten. Modellen är hämtad från webbens ursprungliga ändamål; att visa hypertext.

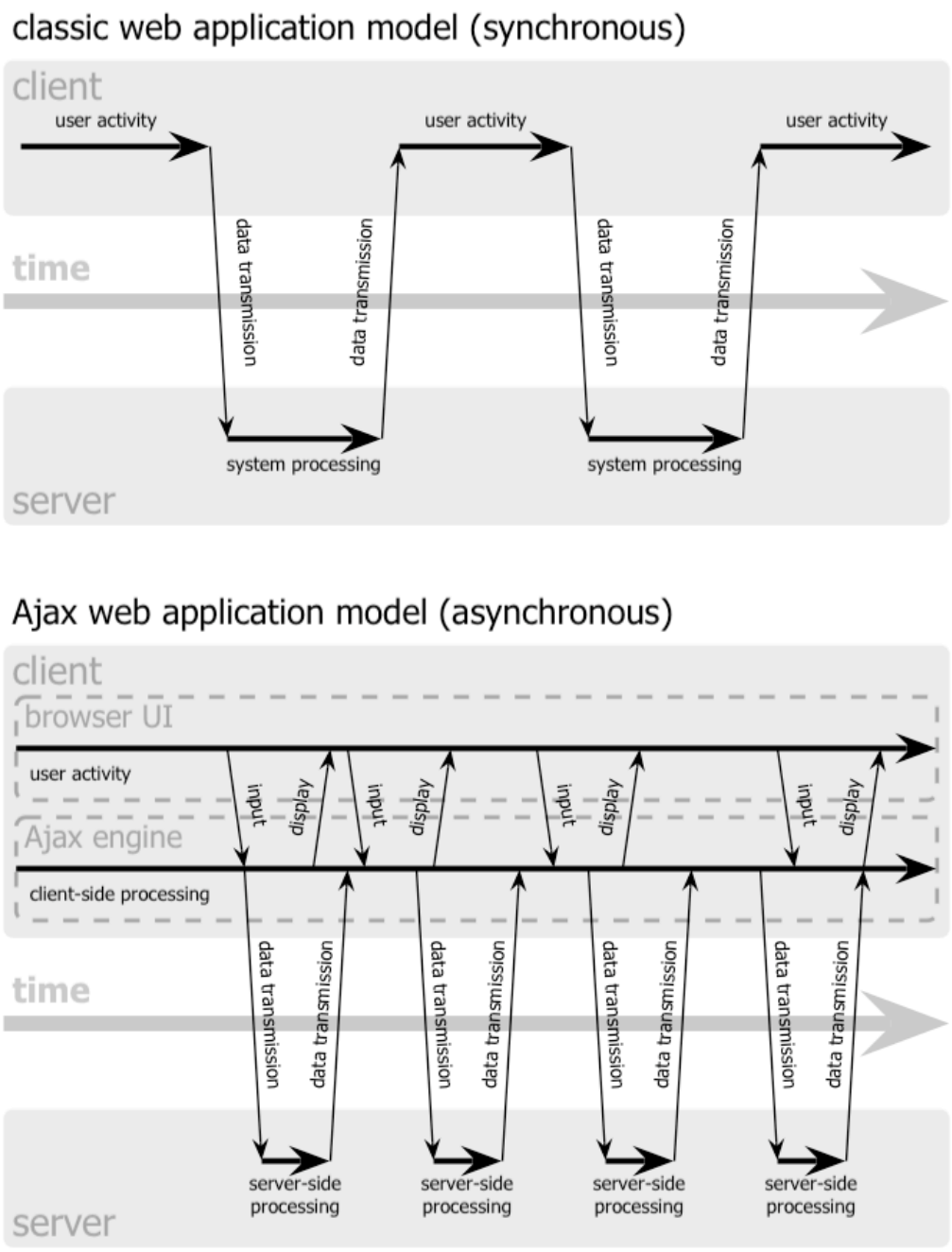
Ur en teknisk synvinkel är det en väldigt vettig modell, men det skapar ett dåligt användargränssnitt, eftersom användaren måste vänta varje gång servern gör något. Om vi från början hade designat webben för applikationer, hade vi inte skapat sådana förutsättningar. Varför skall användaren behöva vänta när gränssnittet redan är laddat, varför skall användaren ens se att applikationen behöver gå till servern? Figur 5 visar skillnaden mellan traditionella webbapplikationer och Ajax-modellen.



Figur 5, den traditionella modellen för webbapplikationer, och Ajax-modellen.
Not. Från "Ajax: A New Approach to Web Applications", av Jesse James Garret.
http://www.adaptivepath.com/publications/essays/archives/000385.php.
Reproducerad med tillstånd under Creative Commons License.

Ajax-modellen lägger ett skikt mellan användaren och servern, en Ajax-motor, som låter användaren kommunicera asynkront med servern. Ajax-motorn är skriven i JavaScript och ligger vanligtvis undangömd i någon ram. Den laddas istället för webbsidan när sessionen startar. Ajax-motorn visar användargränssnittet, och kommunicerar även med servern. Det gör att användaren aldrig behöver sitta och vänta på att servern skall göra något.

Varje sak användaren gör som normalt skulle genererat en HTTP-förfrågan, blir istället ett JavaScript-anrop till Ajax-motorn. Ajax-motorn tar själv hand om saker som inte kräver data från servern, såsom enklare validering av data, manipulering av data som redan ligger i minnet och även lite navigering. Skulle Ajax-motorn behöva något från servern, såsom att skicka in data, ytterligare kod för användargränssnittet eller hämtning av ny data, så gör motorn ett asynkront anrop till servern och hämtar information, vanligen i form av XML, utan att störa användaren. Figur 6 visar skillnaden mellan en synkron och en asynkron användarupplevelse.



Figur 6, skillnad i tidslinje mellan den traditionella modellen och Ajax-modellen.
Not. Från "Ajax: A New Approach to Web Applications", av Jesse James Garret.
http://www.adaptivepath.com/publications/essays/archives/000385.php.
Reproducerad med tillstånd under Creative Commons License.

Alla större produkter som Google introducerat under det senaste året är Ajax-applikationer, även Amazon använder Ajax-funktionalitet i A9.com[42]. Ajax är inte bara en teoretisk teknologi, utan en teknologi som fungerar ute i världen. Utmaningen med Ajax är inte de tekniska bitarna i sig, de är redan välfungerande och stabila. Utmaningen ligger i designen, att tänka bortom de gamla begränsningarna och tänka större.

[42] http://www.a9.com

4. Analys

4.1. Ajax och Sommerville

Sommerville behandlar mjukvaruarkitektur i kapitel 11-13 i sin bok “Software Engineering”. Även om dessa kapitel främst rör design av stora system, så kan man ändå finna många likheter mellan strukturen hos Ajax och de teorier som läggs fram i boken.

Sommerville börjar med att beskriva olika sätt att organisera systemet. Tre modeller läggs fram;[43]

- Lagerhusmodellen (the repository model)
- Klient-servermodellen (the client-server model)
- Den skiktade modellen (the layered model)

Lagerhusmodellen går bort eftersom den rör stora komplexa system där olika undersystem pratar med varandra och delar data, antingen genom en central databas eller genom att varje undersystem har sin egen data som sedan delas ut mellan system.

Man kan tycka att klient-servermodellen skulle vara idealisk för Ajax som verkligen handlar om just klienter och servrar, men den modell som beskrivs är något begränsande och realiserar inte Ajax fulla potential. Sommerville har följande att säga om klient-servermodellen; “*Essentially, a client makes a request to a server and waits until it recieves a reply.*”[44]. Den beskrivningen stämmer in på den traditionella webbmodellen som Ajax försöker utveckla, och räcker således inte till för att beskriva Ajax fullt ut.

För att verkligen kunna beskriva alla egenskaper hos Ajax behöver den skiktade modellen tas till, den anses nämligen “*... organises a system into layers, each of which provide a set of services.*”[45]. Garrett har redan definierat upp dessa skikt, se figur 1 i hans artikel;

- Användargränssnitt (HTML och CSS)
- Ajax-motor (JavaScript)
- Webb- och/eller XML-server (server-sidespråk, till exempel PHP)
- Databas (till exempel MySQL)

De fyra skikten bildar tillsammans Ajax, där varje del står för sin bit. I jämförelse med den traditionella webbmodellen är det bara Ajax-motorn som är något nytt, men det är också just den biten som får Ajax att inte bara vara en klient-servermodell, utan istället röra sig mer mot en skiktad modell.

---

[43] *Sommerville, 11.2.*

[44] *Sommerville, 249.*

[45] *Sommerville, 250.*

Efter att man bestämt sig för en övergripande systemorganisation är det dags att bestämma hur undersystem skall brytas ned i komponenter och hur man designar dessa komponenter, det Sommerville kallar för modular decomposition style. Det finns två olika varianter;[46]

- Objektorienterad (object-oriented decomposition)
- Funktionsorienterad (function-oriented pipelining)

Det intressantaste undersystemet är Ajax-motorn, som kan vara antingen objekt- eller funktionsorienterat, beroende på hur man programmerar det. JavaScript är nämligen ett såpass avancerat och flexibelt språk att det kan göras objektorienterat.[47]

Sannolikt lär smärre databehandling, som att behandla formulär innan de skickas in till servern, vara funktionsorienterade. Större webbsidor med mer applikationsmässigt beteende, som Google Maps[48], lär vara mer objektorienterade. I slutändan beror det dock helt på den specifika implementeringen.

Webbservern kan också vara antingen objekt- eller funktionsorienterad beroende på språk, PHP 5 tillåter till exempel objektorienterad programmering. Databasen följer sina regler som ligger utanför den här uppsatsen. HTML och CSS är textfiler som tolkas av en webbläsare och ligger därför nära det funktionsorienterade synsättet där data hämtas in (HTML- och CSS-filerna) och förvandlas till utdata (när webbläsaren ritar upp innehållet).

Skikt	Komponentdesign
Användargränssnitt	Påminner om funktionsorienterat
Ajax-motor	Funktions- eller objektorienterat
Webb- och/eller XML-server	Funktions- eller objektorienterat
Databas	Ej behandlad

Tabell 1, sammanfattning av komponentdesign

När det gäller kontroll av systemet lägger Sommerville fram två huvudkategorier, som var och en har två underkategorier;[49]

---

[46] *Sommerville, 11.3.*

[47] *Douglas Crockford, JavaScript: The World's Most Misunderstood Programming Language, (www.crockford.com, 2001),* http://www.crockford.com/javascript/javascript.html

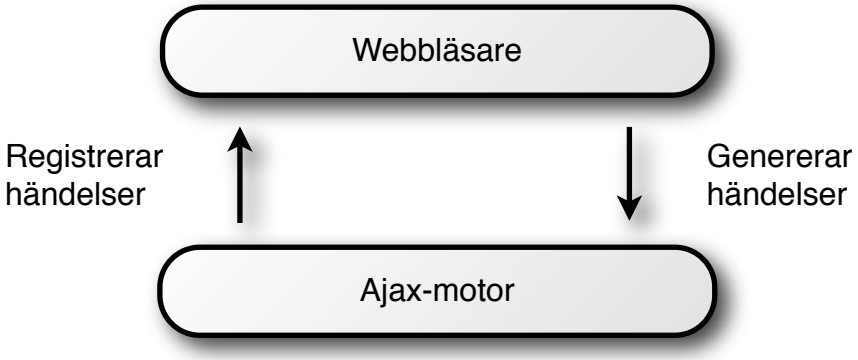
[48] http://maps.google.com/

[49] *Sommerville, 11.4.*

- Centraliserad kontroll (centralised control)
  - Anrops-svarsmodellen (the call-return model)
  - Administratörsmodellen (the manager model)
- Händelsedrivna system (event-driven systems)
  - Distributionsmodellen (broadcast model)
  - Avbrottsmodellen (interrupt-driven models)

Sommerville beskriver administratörsmodellen som att "*One system component is designated as a system manager and controls the starting, stopping and coordination of other system processes.*"[50]. Det stämmer bra in på Ajax, där Ajax-motorn ligger som ett skikt i mitten och styr både vad som syns i webbläsaren samtidigt som den pratar med, och styr, servern. Ajax-motorn styr och påverkar alltså både HTML- och CSS-undersystemet samt server-sideundersystemet, och indirekt även databasen.

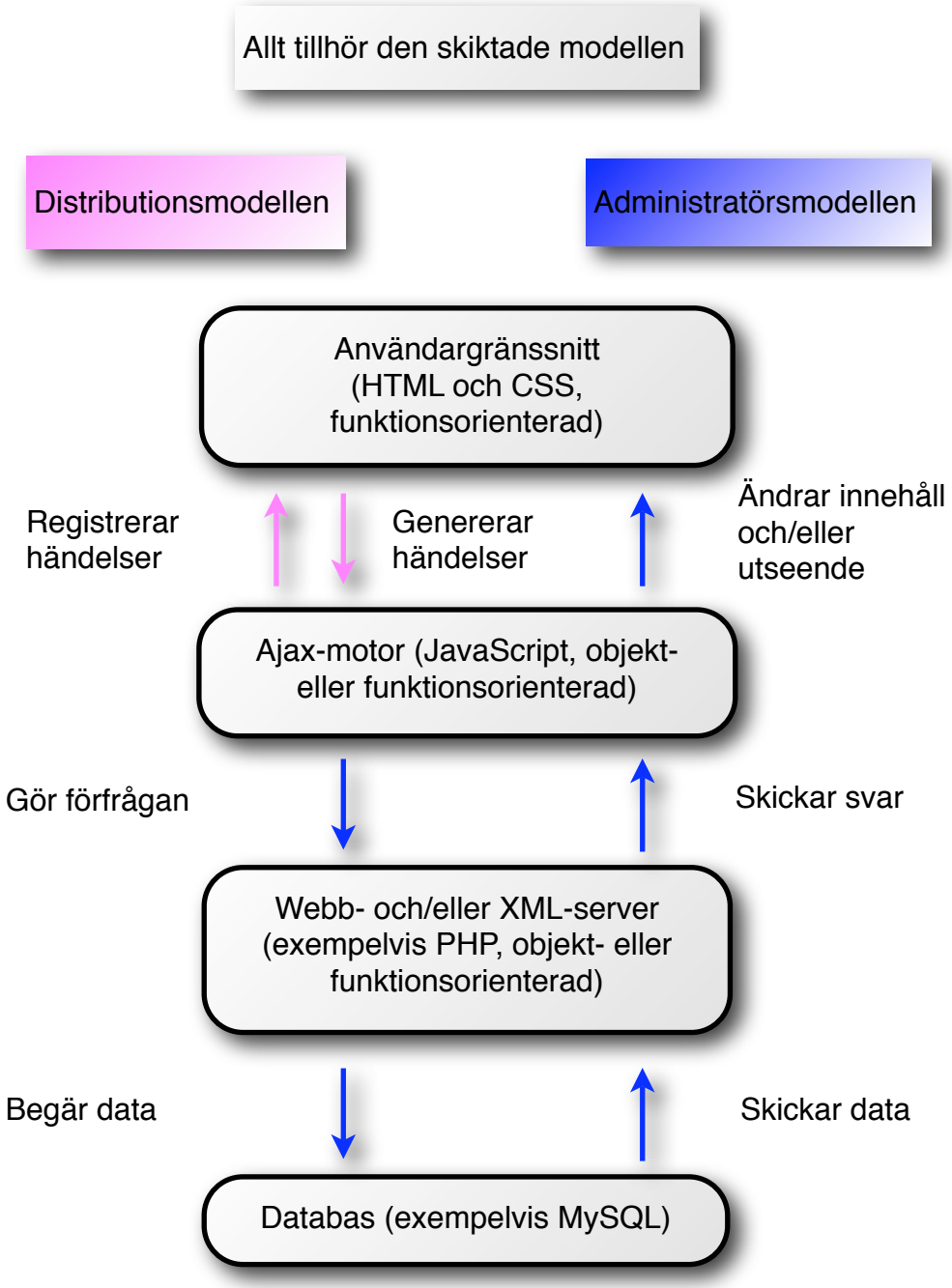
Själva Ajax-motorn kan dock sägas tillhöra distributionsmodellen, om man ser den som ett system i sig. Modellen säger att undersystem själva väljer att prenumerera på händelser, och sedan avgör vad de skall göra med dem. Det synsättet påminner mycket om försynta skript. I försynta skript söker skripten upp alla element i webbsidan och registrerar händelselyssnare där de finner det lämpligt, så att klick, rörelser från musen och annat genererar händelser som sedan skripten kan ta hand om.



Figur 7, Ajax-motorn som distributionsmodell

Man kan sammanfattningsvis säga att Ajax-modellen är ett system uppbyggt enligt administratörsmodellen, medan Ajax-motorn är uppbyggd enligt distributionsmodellen. Det innebär att alla delar i Ajax-modellen kontrolleras av Ajax-motorn, som i sin tur baserar sitt beteende på att avlyssna händelser som genereras av användaren genom interaktion i webbläsaren.

[50] *Sommerville, 256.*



Figur 8, sammanfattning av de olika modellerna för arkitekturdesign i Ajax-modellen

Sommerville definierar distribuerade system som ett system där "*... the information processing is distributed over several computers rather than confined to a single machine*"[51]. Ajax är ett distribuerat system eftersom det sträcker sig över åtminstone två datorer; dels datorn med webbläsaren och dels datorn med servern. Sommerville anger flera möjliga arkitekturer för distribuerade system;[52]

[51] *Sommerville, 267.*

[52] *Sommerville, 12.1-12.3.*

- Flerprocessor (multiprocessor architectures)
- Klient-server (client-server architectures)
  - Tunn klient (thin-client model)
  - Fet klient (fat-client model)
- Distribuerade objekt (distributed object architectures)

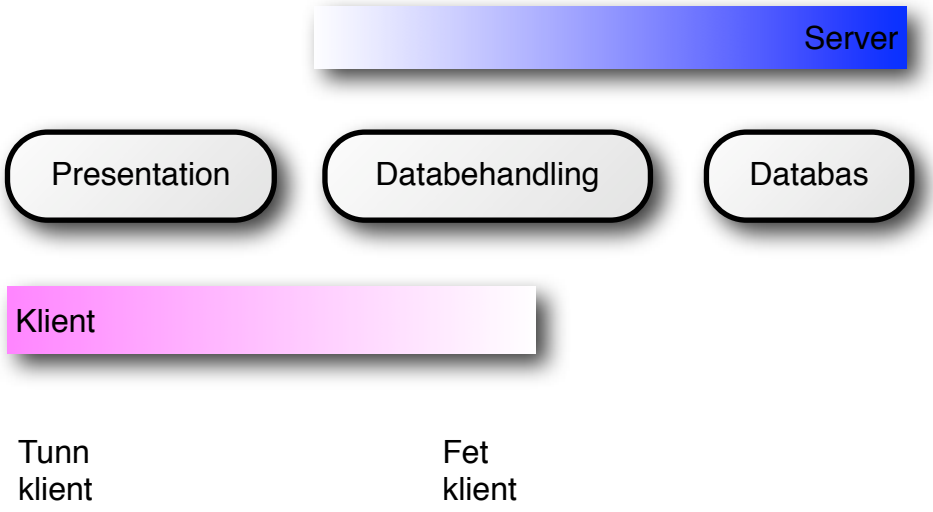
Att Ajax är klient-server har redan etablerats, frågan är om Ajax är tillhör den tunna eller feta klientmodellen. I den tunna klientmodellen hanterar klienten bara presentation medan servern hanterar allt annat, något som stämmer bra överens med den traditionella webbmodellen. I den feta klientmodellen så hanterar klienten presentation och databehandling, medan servern står för datalagringen.

Ajax-modellen ligger någonstans mitt emellan tunna och feta klienter, eftersom Ajax-motorn tar hand om vissa former av databehandling och navigering, samtidigt som servern också tar hand om liknande uppgifter. När man bygger Ajax-motorn väljer man hur mycket databehandling den skall klara av, och genom det hur mycket den skall avlasta servern. Sommerville bekräftar att en sådan modell kan finnas:

"*The advent of mobile code (such as Java applets and Active X controls), which can de downloaded from a server to a client has allowed the development of client-server systems that are somewhere between the thin- and the fat-client model. Some of the application processing software may be downloaded to the client as mobile code, thus relieving the load on the server. The user interface is created using a web browser that has built-in facilities to run the downloaded code.*"[53]

Även om Sommerville inte pratar om Ajax, så är konceptet likt, med några skillnader. Istället för att ladda ned Java-kod eller Active X kontroller, så laddas istället en Ajax-motor. Ajax-motorn är vanligtvis byggd med avancerad JavaScript-kod, något som de flesta moderna webbläsare har stöd för. Syftet med Ajax är dock inte enbart att avlasta servern, utan snarare att utöka funktionaliteten genom att skapa asynkron istället för synkron dataöverföring.

[53] *Sommerville, 273.*



Figur 9, hur Ajax-modellen rör sig mellan tunn och fet klient beroende på implementation

Ajax-modellen verkar inte tillhöra några av de två interorganisatoriska distributionsarkitekturerna.[54] Eftersom det är en stor skillnad mellan klienten och servern så är Ajax inte en icke-hierarkisk (peer-to-peer) arkitektur, inte heller är den en service-orienterad arkitektur. Ajax i sig är nämligen en applikation, och kan således inte vara en service. Däremot kan Ajax mycket väl använda sig av olika så kallade web services (informationsutbyte mellan webbplatser).

Garrett pratar om Ajax som en applikation, för honom är det inte bara ett sätt att göra asynkron överföring mellan server och klient, utan Garrett ser Ajax som ett sätt att skapa applikationer på webben. Sommerville tar upp tre huvudsakliga applikationsarkitekturer;[55]

- Datasystem (data-processing systems)
- Transaktionssystem (transaction-processing systems)
- Händelsesystem (event-processing systems)

Även om Ajax skulle kunna vara ett datasystem, så är det inte i linje med Ajax syfte. Datasystem är ofta system som bör vara ganska säkra, och sannolikt system som inte heller behöver den totala åtkomst som är webbens stora egenskap. Däremot kan ett datasystem ligga och arbeta mot samma databas som Ajax har tillgång till.

[54] *Sommerville, 12.4.*

[55] *Sommerville, 13.1-13.3.*

Ajax lämpar sig bättre som ett transaktionssystem. Sommervilles beskrivning av sådana system går väl i linje med exempelvis Google Suggest[56]; "*Transaction-processing systems are usually interactive systems where users make asynchronous requests for service*"[57].

Figur 13.3 hos Sommerville visar ett transaktionssystem som är lik modellen för tunn och fet klient som beskrivits ovan. Figur 13.6 visar upp ett transaktionssystem i skikt, där skikten kan associeras till skikten som Garrett ger för Ajax-modellen. Kort sagt, transaktionssystemet uppvisar många egenskaper som tidigare i det här kapitlet attribuerats till Ajax.

När det gäller händelsesystem så "*... respond to events in the system's environment or user interface.*"[58]. Händelsesystem är ofta ordbehandlare och liknande applikationer. Redan DOM-skriptning kan låta en webbsida uppfylla kravet att kunna svara mot händelser, det behövs ingen Ajax för det. Ajax kan däremot användas för att göra en webbsida ännu bättre, och ge funktionalitet utöver den som DOM-skriptning kan ge, som till exempel en automatisk sparfunktion i en ordbehandlare som inte behöver störa användarens arbetsflöde.

Ajax kan vara både transaktions- och händelsesystem. Skillnaden mellan dessa båda system är att när man använder Ajax som ett transaktionssystem är det mest för att skapa ett rikt användargränssnitt eller en liten specifik applikation, som i Google Suggest, medan Ajax använt som ett händelsesystem ger en större och mer komplex applikation eller service, som Google Maps eller Gmail.

På grund av Ajax natur att vara en klient som arbetar mot en databas, så ligger händelsesystemen i Ajax värld ganska nära transaktionssystemen. Skillnaden är att i ett transaktionssystem är användaren mer medveten om att den gör just en transaktion, som till exempel att fylla i ett frågeformulär, medan i ett händelsesystem så utförs transaktionen transparent av systemet när användaren genererar en händelse, till exempel om användaren klickar och drar en bild.

Kategori	Ajax	Kapitel
Systemorganisation	Klient-server utökad med skikt	11.2
Distributionsarkitektur	Mellan tunn och fet klient	12.2
Komponentdesign	Vanligen valfri mellan objekt- eller funktionsorienterad	11.3
Systemkontroll	Administrationsmodell, Ajax-motorn är distributionsmodell	11.4
Applikationsarkitektur	Transaktions- eller händelsesystem	13.2-13.3

Tabell 2, sammanfattning av Ajax design

[56] http://www.google.com/webhp?complete=1&hl=en

[57] *Sommerville, 298.*

[58] *Sommerville, 304.*

4.2. Kritik mot Ajax

Koch kritiserar flera aspekter av Ajax i sin artikel “Ajax, promise or hype?”, dels kommer han med egna reflektioner, dels refererar han till flera andra artiklar. Han börjar med att kritisera de byggstenar som enligt Garrett ingår i Ajax.

Koch säger att XML inte borde tagits upp, framförallt för att det inte är nödvändigt för en Ajax-applikation. Vidare anser han att XMLHttpRequest inte heller borde nämnts för att det inte heller är nödvändigt för en Ajax-applikation. Avslutningsvis tycker Koch inte att XSLT är nödvändigt eftersom det gör ungefär samma sak som JavaScript men är mer komplicerat, och JavaScript hade inte heller behövt nämnas eftersom det är underförstått från de andra punkterna.

Vad som då finns kvar är XHTML, CSS och DOM (som också ger JavaScript). Det tycker inte Koch är särskilt nytt eller revolutionerande. Koch skalar visserligen bort vissa icke-essentiella punkter från Garretts överblick, Garrett erkänner själv i frågedelen att XML inte alls är nödvändigt, problemet är dock att när Koch skalar bort den specifika komponenten XMLHttpRequest, så ersätter han den inte med någon generell komponent för asynkron dataöverföring.

Koch tar alltså bort just den delen som ger Ajax möjligheten till asynkron kommunikation, den delen som är själva hjärtat i Ajax. Koch nämner dock osynlig dataöverföring (en referens till asynkron överföring) direkt efter sin avskalning så han tänkte nog på det ändå, men han skriver i så fall fel.

Vad som återstår som en viktig punkt i kritiken är att Garrett använder väldigt specifika teknologier, och att han ibland tar till onödiga sådana, som XML och XSLT. Steve Jenson håller med om detta och hävdar att Garrett inte borde definierat en arkitektur med sina komponenter, utan istället med en underliggande filosofi.[59]

Det hävdas att Garretts idéer inte alls är nya, som exempel refereras en artikel från Apple[60] från 2002 som skall ha tagit upp tekniker för asynkron kommunikation på webben. Apples artikel innehåller mest kodexempel, men det finns också en del övergripande förklaringar i introduktionen. Tabell 3 visar likheter och olikheter mellan de båda artiklarna.

---

[59] *Steve Jenson, Blog utan titel, (saladwithsteve, 21 februari 2005),* http://saladwithsteve.com/2005/02/ajax_21.html

[60] *Apple Developer Connection, Remote Scripting with IFRAME, (Apple, 28 januari 2002),* http://developer.apple.com/internet/webcontent/iframe.html

Garrett	Apple
Pratar specifikt om webbapplikationer snarare än webbsidor	Pratar specifikt om webbapplikationer snarare än webbsidor
Ajax är en blandning av flera teknologier	Fjärrskriptning är en klient-serverteknologi
Ajax-motorn eliminerar start-stop-start-stop från den traditionella webben	Fjärrskriptning löser problemet med webbsidor som måste laddas om varje gång klienten kommunicerar med servern
Ajax-motorn är ett extra lager skrivet i JavaScript som fångar upp användarinteraktion och pratar med servern	En JavaScript-applikation gör anrop till servern
Ajax-motorn laddas istället för en webbsida och stoppas ofta in i en gömd ram	Klient-komponenten (JavaScript-applikationen) är en del av webbsidan
Ajax använder sig vanligtvis av XML för att hämta data	Anger inget specifikt format för data
Använder XMLHttpRequest som komponent för den asynkrona kommunikationen	Nämner flera lösningar för fjärrskriptning, men fokuserar på iframe
Ger exempel på existerande webbapplikationer som använder Ajax	Ger en teknologisk genomgång för hur man skapar ett system med fjärrskriptning
Ajax är en fundamental förändring för vad som är möjligt på webben	
Skriven 18 februari 2005	Skriven 28 januari 2002

Tabell 3, jämförande översikt mellan Garrett och Apple

Den största skillnaden mellan artiklarna är att Garretts är mer lättillgänglig och inte tar upp någon kod. Garrett presenterar en teknologi och berättar för läsaren vad den innebär, medan Apple presenterar en teknologi och berättar för läsaren hur man använder den, Apple låter sedan läsaren fundera själv på vad det innebär.

Det verkar således som om idéerna bakom Ajax är flera år gamla. I frågedelen säger Garrett också själv att det nya med Ajax inte är teknologierna i sig, utan att de först nu börjat användas på riktigt i fungerande applikationer på bred front.

Adams säger att datahämtning genom JavaScript fortfarande är en omogen teknologi vad gäller hur den används. Adams argumenterar för en uppdelning mellan webbapplikationer och webbsidor, där webbapplikationer är menade att uppfylla en specifik uppgift, medan webbsidor är till för att tillhandahålla allmänt tillgänglig information.

Adams uppdelning innebär, enligt honom själv, att webbapplikationer inte behöver följa standarder så noga och inte vara fullt tillgängliga för användare som till exempel inte har JavaScript påkopplat, medan informationen på webbsidor däremot bör vara fullt tillgänglig för alla oavsett klient. Eftersom Ajax handlar om webbapplikationer, så kommer det inte att vara något revolutionerande på grund av att webben till största del består av webbsidor.

Koch går sedan in på tillgänglighet, något som Garrett inte tar upp det minsta. Koch lägger också upp ett förslag för hur man kan göra datahämtningen tillgänglig, som följer samma princip som försynta skript. Om man knyter tillbaka till Adams kan det vara försvarbart för Garrett att inte ta upp tillgänglighet, just eftersom Ajax är en webbapplikation. Koch kritiserar dock inte Garrett direkt för utelämnande av tillgänglighet, han tar bara upp det till diskussion.

Koch rundar av sin artikel med att hävda att användbarhet knappt tas upp i Ajax-diskussionen, inte heller denna kritik riktas explicit mot Garretts artikel. Garrett själv pratar dock väldigt mycket om användbarhet. Han refererar till sin egen bok som handlar om användbarhet på webben, han väver in användbarhet i sin diskussion hela tiden och han talar till och med för det mesta om användaren istället för den mer tekniskt sterila termen klient.

Garretts huvudpoäng rörande användbarhet, är att Ajax-applikationer har asynkron kommunikation, vilket gör att användaren inte skall behöva vänta på svar från servern. Det skall ta upp webbapplikationer på en nivå som liknar traditionella applikationer vad gäller respons.

Bättre respons genom asynkron kommunikation är dock Garretts enda poäng. Fried anser att Ajax smidighet också kan vara en fara, när saker händer så fort att användaren inte riktigt vet att något har hänt. Om användaren sedan vill använda bakåt-knappen för ökad inblick i vad som pågår, är det inte säkert att den har ett beteende liknande ångra-knappen i traditionella applikationer. För att underlätta för användaren föreslår Fried rik visuell feedback för ökad insikt i vad som pågår.

Jeffrey Veen hävdar att Ajax kan få webben att kännas inhemsk (native).[61] Det är ett ganska starkt påstående eftersom inhemsk skulle innebära att webbsidan måste kunna anpassa sitt utseende och beteende efter användarens operativsystem, för att inte nämna alla egna anpassningar av sitt operativsystem användaren gjort. Kanske syftar Veen bara till samma smidiga dataflöde som Garrett. I vilket fall som helst behövs inte just Ajax till att ändra utseende och beteende på en webbsida, det räcker med DOM-skriptning.

---

[61] *Jeffrey Veen, Scrubbing Innovation into Interaction: Ajax, (Jeffrey Veen, 20 februari 2005),* http://www.veen.com/jeff/archives/000689.html

5. Diskussion och slutsats

Ajax är inte nytt, verkligen inte. Däremot har Garrett en poäng i att det är först nu som Ajax verkligen börjat användas på bred front. Det är dock inte den bilden Garrett har ansträngt sig för att framföra, han verkar medvetet sensationell i sin dramatiska formulering om att Ajax representerar något fundamentalt nytt på webben.

Inte nog med att tekniken han presenterar egentligen är gammal, han verkar dessutom i princip ha plagierat en tre år gammal artikel rätt av. Artikeln från Apple och Garretts artikel delar väl många likheter för att det skall vara en tillfällighet. Framförallt är Garretts konstiga påstående att Ajax-motorn oftast göms undan i en ram ett väldigt tydligt tecken på att han sneglat mycket på Apples artikel. Att gömma undan den asynkrona komponenten ligger i linje med användande av en iframe-tagg, men en XMLHttpRequest-komponent är ren JavaScript, som man inte har någon anledning att gömma undan i en ram.

Garretts artikel är enligt den här författaren inget annat än en popularisering av den annars tekniska artikeln från Apple och ett litet utrop som säger “Titta på Google, de gör fräcka saker”. Att sedan kalla det för något fundamentalt nytt och säga att det kommer att förändra webben, utan att ens nämna sina källor eller göra någon form av referens till tidigare arbete, är högst oseriöst.

Garrett har å andra sidan onekligen skakat liv i ett lämpligt teknologiskt paradigm, som säkerligen inte hade fått den renässans eller uppmärksamhet det förtjänar om inte Garretts artikel hade skrivits. Problemet är inte att han populariserar fenomenet, problemet är att han inte ger något erkännande till de som varit före. Istället försöker han sätta sin egen stämpel på det så mycket att han till och med hittar på en ny och egen term, istället för att använda den gamla och fullt dugliga termen fjärrskriptning.

Ajax kommer sannolikt inte att revolutionera webben i stort, dels för att webben som Adams säger är till för webbsidor snarare än webbapplikationer, och dels för att Ajax egentligen inte kan ge någon ny funktionalitet, utom möjligen automatiska sparfunktioner och liknande. Ajax ger nämligen asynkron kommunikation och inget annat, det innebär smidigare och snabbare gränssnitt, inte någon revolutionerande ny funktionalitet. DOM-skriptning är det som gör webbsidor till något dynamiskt där det händer saker, med andra ord utökad funktionalitet, Ajax är bara en förbättring av gränssnittet. Å andra sidan kan korrekt använd Ajax ge en otroligt förbättrad upplevelse för användaren, och det kan vara farligt att underskatta kraften bakom ett smidigt och effektivt gränssnitt.

Eftersom Ajax trots allt introducerar en del förändringar för webbsidor, så tarvas det helt klart mer forskning. Webbsidor med Ajax kommer att bete sig lite annorlunda än andra webbsidor. Främst handlar skillnaderna om att saker kan hända utan att användaren gjort något just då, som ett resultat av den asynkrona kommunikationen, samt oväntat beteende av bakåt- och framåtknappar. Man borde kunna komma långt på vägen genom att kombinera forskning om hur användare upplever traditionella webbsidor, med forskning om hur användare upplever traditionella applikationer, eftersom Ajax är en sorts fusion av de båda.

I och med att man på förhand inte kan veta om en webbsida är förbättrad med Ajax eller inte kommer det att krävas visuell feedback för att tala om det för användaren, så att det alltid är tydligt vad som händer. Eftersom Ajax-applikationer kan bete sig ganska olika beroende på hur man designar dem, skulle en rekommendation från W3C vara på sin plats. Den bör i så fall ta upp saker som beteende för bakåt- och framåtknappar, och vilka situationer som kräver visuell feedback.

Det positiva inför framtiden är att de teknologier som bygger upp Ajax har varit med ganska länge och är ganska stabila. Det finns även beprövade teorier inom mjukvaruarkitektur att luta sig mot i byggandet av Ajax-applikationer. Alla komponenter för utveckling av Ajax finns alltså redan, det kommer dock att krävas utvecklare eller grupper av utvecklare med kompetens inom användbarhet både för webben och för traditionella applikationer, klient- och serverprogrammering samt mjukvaruarkitektur. Alla ingredienser för bra Ajax finns redan, de skall bara kombineras på nya sätt.

6. Referenser

Adam, Cameron. This is not another XMLHttpRequest article. The Man in Blue, 2 mars 2005. http://www.themaninblue.com/writing/perspective/2005/03/02/

Apple Developer Connection. Dynamic HTML and XML: The XMLHttpRequest Object. Apple, 24 juni 2005. http://developer.apple.com/internet/webcontent/xmlhttpreq.html

Apple Developer Connection. Remote Scripting with IFRAME. Apple, 28 januari 2002. http://developer.apple.com/internet/webcontent/iframe.html

Brill, Ryan. Creating Liquid Layouts with Negative Margins. A List Apart, 15 juni 2004. http://www.alistapart.com/articles/negativemargins

Crockford, Douglas. JavaScript: The World's Most Misunderstood Programming Language. www.crockford.com, 2001. http://www.crockford.com/javascript/javascript.html

Fried, Jason. XMLHttpRequest, Ajax and the customer experience. Signal vs. Noise, 24 februari 2005. http://www.37signals.com/svn/archives/001070.php

Garrett, Jesse James. Ajax: A New Approach to Web Applications. Adaptive path, 18 februari 2005. http://www.adaptivepath.com/publications/essays/archives/000385.php

Backman, Jarl. Rapporter och uppsatser. Lund: Studentlitteratur, 1998.

Jenson, Steve. Blog utan titel. saladwithsteve, 21 februari 2005. http://saladwithsteve.com/2005/02/ajax_21.html

Koch, Peter-Paul. Ajax, promise or hype? QuirksMode, 13 mars 2005. http://www.quirksmode.org/blog/archives/2005/03/ajax_promise_or.html

Koch, Peter-Paul. DHTML != DOM. QuirksMode, 16 januari 2005. http://www.quirksmode.org/blog/archives/2005/01/dhtml_dom_1.html

Koch, Peter-Paul. XMLHTTP notes: readyState and the events. QuirksMode, 21 september 2005. http://www.quirksmode.org/blog/archives/2005/09/xmlhttp_notes_r_2.html

Langridge, Stuart. Modern Web Design Using JavaScript & DOM. Collingwood: SitePoint Pty.Ltd. 2005.

Mathiassen, Lars, Andreas Munk-Madsen, Peter Axel Nielsen, Jan Stage. Objektorienterad analys och design. Lund: Studentlitteratur, 2001.

Myer, Thomas. XML Web Development With PHP. Collingwood: SitePoint Pty. Ltd. 2005.

Newhouse, Mark. CSS Design: Taming Lists. A List Apart, 27 september 2002. http://www.alistapart.com/articles/taminglists

Sommerville, Ian. Software Engineering, 7th ed. Addison-Wesley, 2004.

Veen, Jeffrey. The Business Value of Web Standards. Adaptive path, 18 september 2003. http://www.adaptivepath.com/publications/essays/archives/000266.php

Veen, Jeffrey. Scrubbing Innovation into Interaction: Ajax. 20 februari 2005. http://www.veen.com/jeff/archives/000689.html

Wikipedia. 22 september 2005. http://en.wikipedia.org/wiki/Internet

Zeldman, Jeffrey. *Fix Your Site With The Right DOCTYPE! A List Apart, 12 april 2002.* http://www.alistapart.com/articles/doctype

Zeldman, Jeffrey. *From Table Hacks to CSS Layout: A Web Designer's Journey. A List Apart, 16 februari 2001.* http://www.alistapart.com/stories/journey/