Umeå universitet
Institutionen för datavetenskap
2006-06-12
Examensarbete D, 20p

Webbapplikationer med Ajax

Samuel Nyman

Intern handledare:
Thomas Johansson

Kursansvarig:
Per Lindström

Sammanfattning

Ajax är ett samlingsnamn för en rad etablerade tekniker som finns tillgängliga i moderna webbläsare. Dessa består i huvudsak av JavaScript, XMLHttpRequest och DOM. Med Ajax kan man skapa webbapplikationer som inte behöver ladda om hela sidor och därmed uppträder de mer som skrivbordsapplikationer. Denna rapport beskriver dels vad Ajax är för något samt vilka styrkor och svagheter tekniken har jämfört med andra tekniker. En av fördelarna med att använda Ajax är att det inte kräver att användaren installerar någon plugin i webbläsaren för att applikationen ska fungera.

Syftet med examensarbetet är att implementera en webbapplikation baserad på Ajax. Denna ska ersätta en befintlig webbapplikation i form av en Java Applet som dagligen används av marknadsavdelningen vid ICA AB. Den nuvarande applikationen innehåller en del brister, både funktionella och sådana som berör användbarhet. För att fånga upp dessa har den nuvarande applikationen testats utförligt och utvärderats utifrån olika användbarhetsprinciper.

Eftersom Ajax grundar sig på etablerad webbteknik kan man relativt enkelt uppgradera befintliga sidor så att de drar nytta av Ajax. Den stora nackdelen med Ajax är dock att den underliggande tekniken inte är standardiserad. Man måste därför hantera olikheter i olika webbläsare. Slutsatsen som dras är att det finns en framtid för mer avancerade webbapplikationer. Om dessa kommer att vara baserade på Ajax återstår dock att se.

Abstract

Ajax is based on technology that is available in most modern web browsers, mainly JavaScript, XMLHttpRequest and DOM. Ajax makes it possible to create web applications that resemble desktop applications. This thesis describes what Ajax is and what its strengths and weaknesses are compared to alternative technology. One advantage with Ajax is that there is no need for the user to install software in the web browser such as a plugin.

The purpose with this thesis is to implement a web application based on Ajax that will replace an existing Java Applet. The Applet is used on a daily basis by the marketing unit at ICA AB. There are some limitations with the current application in terms of usability and functionality. To solve these issues the application has been thoroughly tested and also evaluated according to known usability principles.

Since Ajax is based on established web technology it is possible to upgrade existing web sites with Ajax in order to benefit from its advantages. The downside with Ajax is that the underlying technology does not behave consistently on different web browsers and must therefore be handled accordingly. The conclusion of this thesis is that there is a future for more advanced web applications. If they will be based on Ajax or some other technology remains to be seen.

Innehållsförteckning

Figurlista

1 Introduktion

1.1 Inledning

En webbapplikation är en applikation som levereras från en webbserver i syfte att köras i en webbläsare, antingen över Internet eller över ett intranät. En av skillnaderna jämfört med skrivbordsapplikationer är att webbapplikationer använder en universalklient, själva webbläsaren. Detta gör att en användare inte behöver installera ett nytt program för varje applikation som ska köras men det begränsar å andra sidan funktionaliteten i webbapplikationen till vad som för tillfället är möjligt inom webbläsarens ramar [1].

I takt med att webbläsarna blir mer avancerade kan också mer avancerade applikationer köras över webben. Det finns en vision om att man i framtiden kommer att kunna köra alla sina applikationer över Internet och på så sätt komma åt alla sina dokument oavsett vilken dator man använder och var man befinner sig [2]. Dessa typer av applikationer har fått benämningar, exempelvis *Rich Internet Applications* och *Web 2.0*. Jag har valt att i denna rapport kalla dem för rika webbapplikationer. Ett av de mer kända exemplen av rika webbapplikationer är Googles webbaserade e-postapplikation Gmail[1]. Gmail är baserat på en teknik som på senare tid har kommit att kallas för Ajax.

1.2 Bakgrund

ICA är en av Sveriges största aktörer inom matvaruhandeln. Liksom många andra stora företag har ICA en egen webbportal, ICA.se[2]. Där kan ICAs kunder bland annat söka efter butiker, kontrollera saldon på sina ICA-kort, utföra bankärenden via ICABanken samt hitta information om och inspiration till matlagning. ICA.se har cirka 1,5 miljoner besökare varje månad. Receptsamlingen är en av kärnfunktionerna på ICA.se. Man har i dagsläget en receptsamling med drygt 6000 recept, nya recept läggs in varje vecka.

Alla recept ligger lagrade i en relationsdatabas. Varje recept har en titel, en ingredienslista och en receptbeskrivning. Ett recept kan tillhöra en eller flera kategorier såsom typ av rätt (”förrätter”, ”soppor & grytor”), tillagningssätt (”grilla”, ”woka”). Recepten har även olika egenskaper ”vegetariskt”, ”nyckelhålsmärkt” med mera. I recepten anges dessutom dess svårighetsgrad och en uppskattad tillagningstid.

Sökning av recept kan utföras på olika sätt. En variant är att skriva in sökord som därefter matchas i receptets titel eller ingredienslista, men det är också möjligt att hitta den typ av recept man söker genom att välja bland de olika kategorierna.

Att lägga in och hantera recepten i databasen sköts av marknadsavdelningen vid ICAs huvudkontor. Marknadsavdelningen har inte direkt tillgång till databasen utan de använder istället ett administrationsgränssnitt i form av en Java Applet.

[1] http://mail.google.com
[2] http://www.ica.se

Denna ligger åtkomlig inom ICAs innehållshanteringssystem (content management) Cobra, vilket är webbaserat.

Systemutvecklarna är inte nöjda med det nuvarande administrationsgränssnittet. Några av nackdelarna man ser med den nuvarande lösningen är följande:

- En Java Applet är beroende av att Java Virtual Machine (JVM) finns installerad på användarens dator. JVM är inte en del av ICAs standardinstallation.

- Java Appleten blir inom Cobra en applikation i applikationen. Detta blir förvirrande för användarna.

- Det är tidsödande och besvärligt för systemutvecklarna att underhålla och utveckla administrationsgränssnittet.

- Det tar lång tid för webbläsaren att ladda in appleten.

Man tror på ICA IT att en Ajax-baserad lösning skulle vara att föredra framför appleten eftersom denna skulle bli mer integrerad i Cobra och därmed kunna utgöra ett enklare användargränssnitt. En Ajax-lösning skulle eliminera behovet av JVM och troligen gå snabbare att starta. Dessutom tror man att en Ajax-applikation skulle vara enklare att underhålla eftersom den blir mer lik de övriga delarna i Cobra och därmed hamnar närmare systemutvecklarnas expertområde.

1.3 Uppgift/Mål

Målet med examensarbetet är att ta fram ett nytt användargränssnitt till receptadministrationen. Det nya användargränssnittet ska vara baserat på Ajax. Användargränssnittet ska erbjuda samma funktioner som finns i det nuvarande gränssnittet och kravet är att det ska fungera i ICAs standardinstallation.

Upplägget på arbetet kommer att vara enligt följande:

1. Inhämta fördjupad kunskap om Ajax och webbapplikationer, detta blir examensarbetets teoretiska fördjupning.

2. Kartlägga funktionaliteten i det nuvarande administrationsgränssnittet samt verifiera att dessa går att återskapa i en Ajax-baserad lösning.

3. Identifiera användargränssnittets brister samt möjligheter till förbättring.

4. Undersöka hur man på bästa sätt kan integrera en Ajax-baserad lösning i ICAs nuvarande arkitektur.

5. Ta fram krav och riktlinjer för en implementation.

6. Implementera en ny applikation i Ajax

7. Utvärdera Ajax och den utvecklade implementationen.

1.4 Metod

Ajax var ett nytt begrepp för mig när jag påbörjade examensarbetet. Den initiala fasen av examensarbetet gick därför åt till att skaffa kunskap om Ajax. Detta har jag gjort både genom att läsa om Ajax och genom att utförligt testa befintliga Ajax-applikationer.

När det gäller information som jag har hittat på webben har jag försökt ranka digniteten hos de olika källorna. Vidare har jag kontrollerat att det funnits stöd för påståenden från andra håll, även om jag bara refererar till en källa.

Eftersom målet med examensarbetet har varit att implementera en applikation som ska kunna användas av individer med relativt liten datorvana har jag lagt ner en hel del tid på att utvärdera den nuvarande applikationens användarvänlighet.

Implementationen av den nya applikationen har varit den största delen av examensarbetet. Mitt arbetssätt har varit att försöka dela upp uppgiften i flera mindre delar och sedan lösa dessa en efter en.

2 Ajax

Ajax *(Asynchronus JavaScript and XML)* är ingen ny teknologi i sig, snarare ett samlingsnamn för en rad etablerade tekniker som används tillsammans på ett visst sätt. Begreppet Ajax myntades av Jesse James Garrett vid Adaptive Path [3] och har därefter blivit allmänt accepterat, även om det finns de som är kritiska till begreppet [4]. Tanken med Ajax är att teknik som redan finns tillgänglig i webbläsaren används för att skapa rikare webbapplikationer.

I en traditionell webbapplikation levereras allt innehåll ifrån webbservern till webbläsaren. Webbläsaren fungerar som en tunn klient. När användaren interagerar med applikationen skickas information till servern och denna genererar därefter en ny sida som läses in av webbläsaren. Under den tid då servern arbetar kan inte användaren göra något med applikationen.

I en Ajax-applikation läggs mer funktionalitet över till klientens sida. Beräkningar och uppdateringar av användargränssnittet kan skötas av klienten och serverns huvudsakliga uppgift blir istället att leverera data till och ta emot data från klienten när detta behövs. Detta gör att interaktionen med användargränssnittet kan få ett bättre flyt eftersom klienten inte behöver vänta på att hela sidan ska laddas om [5].

Ajax fungerar i ett flertal moderna webbläsare men trots att webbläsare har stöd för Ajax är det inte säkert att den klarar alla Ajax-baserade webbsidor. Figur 1 visar vilka webbläsare (och från vilken version) som fungerar med Google Maps [6].

Webbläsare	**Version**
Internet Explorer	6.0
Firefox	0.8
Safari	1.2.4
Netscape	7.1
Mozilla	1.4
Opera	8.02

Figur 1: Webbläsare som stödjer Ajax.

2.1 Beståndsdelar

Ajax är inte någon fastställd standard och det finns inga strikta krav på hur en Ajax-applikation ska vara uppbyggd. Valet av verktyg och metod för dataöverföring kan variera men vanligen används följande tekniker.

2.1.1 CSS

Cascading Style Sheets (CSS) är en typ av formatmall som används för att bestämma utseendet hos en webbsida. Detta separeras ifrån innehållet (HTML-dokumentet) och lagras i en CSS-fil. CSS-filen består av regler som anger hur HTML-elementen ska presenteras, till exempel färg, marginaler och typsnitt. Elementen kan kopplas till en eller flera regler. Genom att använda CSS blir det möjligt att definiera utseendet hos en eller flera webbsidor med en och samma fil, detta gör det enklare att få ett konsekvent utseende som också är lättare att byta ut [5].

2.1.2 DOM

Document Object Model (DOM) är ett API som gör det möjligt för ett programspråk att manipulera en objektrepresentation av ett HTML- eller XML-dokument. DOM är en standard som upprätthålls av W3C[3] (The World Wide Web Consortium) och är inte bundet till en speciell plattform eller ett speciellt programmeringsspråk [7].

2.1.3 JavaScript

JavaScript är ett svagt typat programmeringsspråk som är baserat på ECMAScript. Programspråkets främsta användningsområde är webbsidor. JavaScript har en liknande syntax som Java men språken har i övrigt inte mycket gemensamt. Programkoden kompileras inte i förväg utan tolkas direkt av webbläsaren under körtid. Källkoden ligger antingen inbäddad i HTML-dokumentet eller som separata filer som HTML-dokumentet länkar till [5].

2.1.4 XMLHttp/XMLHttpRequest

Ett objekt som används för att i bakgrunden låta en webbapplikation kommunicera med en webbserver, kommunikationen sker med hjälp av Http-protokollet. Vanligtvis skickas data i XML[4]-format men andra textbaserade format är också möjliga [8].

[3] http://www.w3.org
[4] http://www.w3.org/XML/

2.2 Kommunikation

I en Ajax-applikation sker all kommunikation mellan klient och server enligt en request/response-modell som följer Http-protokollet. Kommunikationen sker med hjälp av ett XMLHttp- eller ett XMLHttpRequest-objekt. XMLHTTP utvecklades av Microsoft och är ett ActiveX-objekt vilket kan användas av bland annat JavaScript. XMLHTTP är inte någon utbredd standard utan används endast av Internet Explorer.

Mozilla har utvecklat ett eget objekt som de kallar för XMLHttpRequest, detta objekt är inbyggt i webbläsaren och används i alla Mozilla-baserade webbläsare, Apples webbläsare Safari samt webbläsaren Opera. Microsoft har antytt att även de kommer att använda sig av XMLHttpRequest från och med version 7 av Internet Explorer. XMLHttpRequest och XMLHttp fungerar i stort sett på samma sätt. Det finns dock vissa skillnader mellan de olika objekten. De stödjer lite olika metoder och skapas på olika sätt, XMLHttp-objektet skapas dessutom på olika sätt beroende på vilken version av Internet Explorer som används [9].

I denna rapport kommer endast de attribut och metoder som är gemensamma för båda objekten att beskrivas, dessa kan ses i Figur 2 respektive Figur 3 [10]. För enkelhetens skull kommer båda objekten fortsättningsvis att kallas för XMLHttpRequest.

Attribut	Beskrivning
onreadystatechange	Händelsehanterare som anropas varje gång tillståndet (readyState) i request-objektets anrop förändras.
readyState	Tillståndet för ett request-anrop. 0: uninitialized 1: loading 2: loaded 3: interactive 4: complete
responseText	Serverns svar representerat som en textsträng.
responseXML	Serverns svar i form av ett DOM-objekt av typen XML
status	Statuskoden för resultatet av request-anropet
statusText	Beskrivande text till statuskoden.

Figur 2: Egenskaper hos XMLHttpRequest

Metod	Beskrivning
abort()	Avbryter request-anropet
getAllResponseHeaders()	Hämtar all information i headern
getResponseHeader()	Hämtar information från headern
open()	Initierar ett request-anrop.
send()	Skickar ett request-meddelande
setRequestHeader()	Används för att göra inställningar i headern.

Figur 3: Metoder i XMLHttpRequest

För att en Ajax-applikation ska fungera i så många webbläsare som möjligt måste man bland annat hantera skillnaderna i hur XMLHttpRequest-objektet skapas. Detta löser man genom att använda en så kallad wrapper-funktion. Beroende på vilken webbläsare som används skapar denna funktion rätt objekt. Ett exempel på en wrapper-funktion som löser detta kan ses i Figur 4.

```
function getXRequest() {
    var xRequest=null;
    if (window.XMLHTTPRequest) {
        xRequest=new XMLHTTPRequest(); <-- Mozilla/Safari mm.
    }
    else if (typeof ActiveXObject != “undefined”){
        xRequest=new ActiveXObject(“Microsoft.XMLHTTP”); <-- IE
    }
    return xRequest;
}
```

Figur 4: Skapande av ett XMLHttpRequest-objekt.

2.2.1 Att skicka ett request-meddelande

När väl ett XMLHttpRequest-objekt finns till hands kan man börja skicka request-meddelanden till servern. För att göra detta måste man initiera anropet med hjälp av metoden *open*. Denna metod tar emot följande argument:

1. *method*: ”GET”, ”POST” eller ”HEAD”.
2. *url*: URL till servern.
3. *asynch*: ”true/false”.
4. *username*
5. *password*

Endast anropsmetod och URL är obligatoriska men för att öka tydligheten i koden anges oftast även om anropet ska ske asynkront eller inte. Om ett request-meddelande skickas synkront kommer klientapplikationen att vänta till dess att hela request/response-cykeln är avslutad. Skickas ett request-meddelande däremot asynkront fortsätter programkoden i den anropande funktionen att exekveras direkt efter att request-meddelandet skickats iväg. Användaren behöver därmed inte vänta på servern och det är på detta sätt request-meddelanden vanligen skickas i Ajax-applikationer. Vid vissa fall kan servern kräva autentiering. I dessa fall måste även användarnamn och lösenord skickas med som argument.

För att konfigurera headern i ett request-meddelande använder man metoden *setRequestHeader*. Med denna metod kan man bland annat specificera vilken typ av data som skickas. När allt är klart skickas request-meddelandet iväg med metoden *send*. Om data ska bifogas med anropet skickas detta med som argument [11].

Figur 5 visar ett exempel på ett request-meddelande av typen ”POST”. Data som skickas med i meddelandet är i xml-format och detta specificeras i headern i innan meddelandet skickas iväg.

```
function sendRequest(url, xml) {
    var xRequest = getXRequest();
    xRequest.open("POST", url, true);
    xRequest.setRequestHeader('Content-Type', 'text/xml; charset:UTF-8');
    xRequest.send(xml);
}
```

Figur 5: Request-anrop med XMLHttpRequest

2.2.2 Hantera svar från server

Vid asynkron kommunikation väntar inte klienten på ett svar ifrån servern. Svaret kan komma när som helst och klienten måste därför kunna hantera svaret när det kommer. För att kunna hantera detta använder man *onreadystatechange*. Detta attribut tilldelas en funktion som ska exekveras när anropet är slutfört, en så kallad callback-funktion. Varje gång tillståndet i XMLHttpRequest-objektet förändras kommer callback-funktionen att anropas. Beroende på vilken webbläsare som används kommer detta att göras upp till fyra gånger. Figur 6 visar hur en callback-funktion blir tilldelad.

```
function sendRequest(url, xml, callback) {
    var xRequest = getXRequest();
    xRequest.onreadystatechange = callback
    xRequest.open("POST", url, true);
    xRequest.setRequestHeader('Content-Type', 'text/xml; charset:UTF-8');
    xRequest.send(xml);
}
```

Figur 6: Objektet tilldelas en callback-funktion.

När ett request-anrop är slutfört kommer statusen att vara lika med 4, och callback-funktionen kommer att anropas för sista gången. Vid detta tillfälle vet man att ett fullständigt svar har kommit från servern, men något kan ha gått fel och man bör därför ta reda på hur det har gått. Detta gör man genom att kontrollera vilken statuskod[5] som servern har returnerat.

Figur 7 visar en enkel callback-funktion som testar att request-anropet har slutförts och att servern har returnerat ett meddelande med statuskod 200 (OK), därefter skickas svaret vidare till en funktion i klientapplikationen.

```
function callback() {
   if (x.readyState == 4) {
      if (x.status == 200) {
         processResponse(x.responseXML);
      }
   }
}
```

Figur 7: En enkel callback-funktion

[5] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Innehållet i serverns svar hamnar i attributen *responseText* och *responseXML*. I *responseText* lagras svaret som en textsträng vilket innebär att det kan vara på vilket textbaserat format som helst. Om svaret är av korrekt XML-format kommer det dessutom att i responseXML vara åtkomligt i form av ett DOM-objekt. Detta objekt innehåller olika metoder som gör det enkelt att komma åt informationen [9].

2.3 Dynamiskt användargränssnitt

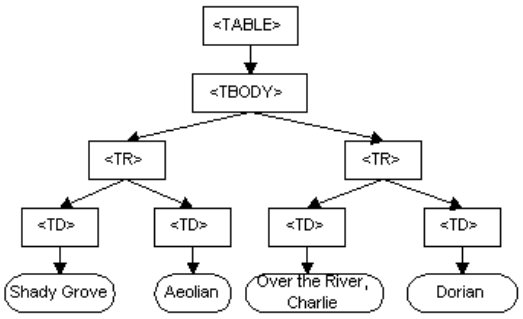
Användargränssnittet i en webbapplikation utgörs av webbsidan och är uppbyggt av HTML och CSS. I webbläsaren representeras en webbsida av ett DOM-träd. Varje enhet i HTML-dokumentet är representerad av en nod i detta träd. Med hjälp av ett programspråk är det möjligt att hämta information från noder i trädet men det går även att manipulera noderna samt att skapa nya noder. När strukturen i DOM-trädet förändras kommer resultatet av förändringen att synas direkt i webbläsaren. Det är denna egenskap som medför att Ajax-applikationer inte behöver ladda in en helt ny webbsida för att uppdatera användargränssnittet [12].

Det finns olika typer av noder, de vanligaste noderna för ett HTML-dokument är:

- *Document*: DOM-trädet, denna nod är inte en del av själva HTML-dokumentet.
- *Element*: HTML-element, till exempel `<a>` eller `<img>`.
- *Attribute*: Attributen hos ett element, exempelvis `href` i ett `<a>`-element.
- *Text*: All text i ett HTML-dokument, kan till exempel ha ett `<h1>`-element som förälder.

Noder av typen *attribute* ligger till skillnad från de övriga noderna inte i trädstrukturen. Dessa noder har därmed varken någon förälder eller några barn. Varje *element*-nod har istället en lista där alla dess *attribute*-noder lagras. Figur 8 visar ett HTML-dokument med ett motsvarande DOM-träd.

```
<TABLE>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```



Figur 8: HTML med motsvarande DOM-träd.

I JavaScript finns den aktuella webbsidans DOM tillgänglig, som ett objekt, genom den globala variabeln *document*. All interaktion med DOM:en sker genom detta objekt.

För att kunna navigera i DOM-trädet innehåller varje nod attribut som länkar noden till dess omgivning: *parentNode*, *previousSibling*, *nextSibling* och *childNodes.* Även om det fullt möjligt att navigera runt i trädet enbart med hjälp av dessa attribut så är inte detta att rekommendera då det kräver att man hela tiden har koll på trädets struktur. DOM-objektet erbjuder två metoder som gör det betydligt enklare att hitta de element man söker.

- *getElementById()*: returnerar det första elementet som har ett matchande id.

- *getElementsByTagName()*: returnerar en lista som innehåller alla element av en viss typ.

Metoden *getElementById()* anses säkrast att använda eftersom man inte gör ett lika stort antagande av strukturen i HTML-dokumentet, vilken på sikt kan förändras [5]. För att denna metod ska fungera krävs det dock att elementet man söker har ett definierat *id*. När man har hittat det element man söker kan man enkelt hämta eller ändra information och egenskaper i elementet.

Nya noder skapas beroende på typ med metoderna *createElement()* och *createTextNode()*. För att tilldela ett element olika attribut använder man metoden *setAttribute()*, denna metod tar attributets namn och värde som argument. När en nod har blivit skapad kan den läggas till i DOM-trädet med metoden *appendChild()*. För att ta bort en nod ur trädet kan man använda metoden *removeChild()* [12].

Det finns ett flertal andra metoder som kan användas för att manipulera DOM-trädet, dessa går att hitta på W3C.

2.4 Användningsområde

Ett exempel på en vanlig webbapplikation som kan dra nytta av Ajax är registrering av användarkonton. För att registrera en användare får man vanligtvis fylla i ett formulär med önskat användarnamn, lösenord, personliga uppgifter och så vidare. När allt är ifyllt klickar man på ”skicka” och väntar på svar från servern. Om användarnamnet är upptaget kommer man tillbaka till samma sida igen och får prova med ett annat användarnamn. Med hjälp av Ajax kan användarnamnet kontrolleras direkt och man kan därmed uppmanas att byta användarnamn direkt utan att behöva vänta på att hela sidan ska laddas om.

Ajax kan användas till att förbättra funktionaliteten och öka användarvänligheten på befintliga webbsidor, men tekniken kan även användas till att skapa webbapplikationer som påminner om vanliga skrivbordsapplikationer. Dessa är ofta uppbyggda som en enda webbsida. Det finns flera exempel på Ajax-applikationer som fungerar som skrivbordsapplikationer, några framgångsrika exempel är dessa:

- Google Maps[6] – En sökmotor för kart- och sattelitbilder. Vid sökning uppdateras enbart själva kartbilden. Det går bland annat att ändra vy genom att dra kartbilden med hjälp av muspekaren. Kartan är uppdelad i flera små bilder vilka läses in vartefter vyn ändras.

- Google Suggest[7] – En sökmotor som fungerar som vanliga Google fast med ”komplettera automatiskt”-funktionalitet (autocomplete). För varje bokstav som matas in skickas ett anrop till servern och en lista med förslag på sökord presenteras. Användaren kan välja något av dessa genom att stega med piltangenterna.

- Meebo[8] – En IM-klient (Instant Messaging) som kan köra ett flertal olika IM-protokoll samtidigt, bland annat ICQ och MSN Messenger. Meebo skiljer sig något ifrån andra Ajax-applikationer. Ett krav på en IM-klient är att applikationen måste uppdateras snabbt. Man ska i princip få ett meddelande så snart det har skickats iväg från avsändaren. Men Ajax erbjuder bara kommunikation via Http-protokollet vilket innebär att klienten måste fråga efter information för att kunna få någon. För att lösa detta problem används en teknik som kallas för Comet. Comet går ut på att applikationen skickar request-meddelanden till servern som sedan håller inne med svaret till dess att det finns information att svara med [13].

2.5 Alternativ till Ajax

I denna avdelning presenteras olika alternativ till Ajax. Lösningar som bygger på att användaren installerar och kör en separat skrivbordsapplikation kommer inte att tas upp. Med Java Web Start kan en applikation startas från webbläsaren men därefter körs den som en skrivbordsapplikation. Vidare har jag begränsat mig till tekniker som finns och används i dagsläget, XAML och SVG kommer inte med av denna anledning.

2.5.1 Macromedia Flash

Macromedia Flash används ofta för att göra interaktiva och animationsrika applikationer och presentationer. Flash har bland annat stöd för vektorgrafik, vilket innebär att objekt kan representeras som geometriska former, kurvor och dylikt. Dessutom stödjer Flash både strömmande ljud och video.

Som programspråk används Actionscript, som i likhet med JavaScript är baserat på ECMAScript, till skillnad från JavaScript är ActionScript strikt typat. Koden exekveras av Flash Player och behöver därmed inte skrivas i flera versioner för att fungera i olika webbläsare.

Något som anses vara en nackdel med Flash är att det krävs speciell kompetens inom Flash för att kunna utveckla applikationer. För att göra detta enklare för nybörjare har Macromedia tagit fram Flex, vilket är ett ramverk för att skapa presentationslager för rika webbapplikationer, ett gratisalternativ är att använda Open Lazlo.

[6] http://maps.google.com
[7] http://www.google.com/webhp?complete=1&hl=en
[8] http://www.meebo.com

För att kunna köra Flash-applikationer i en webbläsare krävs det att en plugin (Flash Player) finns installerad i webbläsaren. [14]

2.5.2 Java Applet

Java Appletar har funnits sedan 1995 och är en typ applikationer skrivna i Java som körs i en webbläsare med hjälp av Java Virtual Machine (JVM). Appleten kan antingen köras som en del av en webbsida eller i ett eget fönster.

Eftersom Appletar körs i JVM är en Java Applet helt oberoende både av vilken plattform och webbläsare den körs under. Webbläsaren måste därför ha JVM installerad i form av en plugin.

Appletar uppfattas av många användare som långsamma att starta och detta kan ha bidragit till att de inte har blivit så populära [15].

2.5.3 XUL

XML User Interface Language (XUL) är skapat av Mozilla och används för att ta fram användargränssnitt till Mozilla-applikationer, till exempel Mozilla Firefox. XUL kan både användas till applikationer som körs lokalt på en dator (local XUL) och till fjärrapplikationer (remote XUL), Remote XUL har vissa begränsningar.

I likhet med Ajax använder XUL JavaScript, CSS och XMLHttpRequest. XUL har dessutom stöd för att kunna kommunicera direkt med databaser.

En XUL-baserad webbapplikation kan köras direkt från en webbserver med en kompatibel webbläsare, det finns även sätt att kombinera XUL och HTML. Webbapplikationer skapade med XUL går enbart att köra med Mozilla-baserade webbläsare [16].

2.6 Fördelar/Nackdelar

Den stora fördelen med att använda Ajax jämfört med andra tekniker är att Ajax kan integreras i befintliga lösningar. Detta medför att användarna av applikationen kan fortsätta att arbeta i en för dem känd miljö, i bästa fall märker de inte av någon (negativ) skillnad. Det underlättar även för webbutvecklarna då dessa inte behöver sätta sig in en helt ny utvecklingsmiljö. För erfarna webbutvecklare anses inte steget över till Ajax vara lika långt [15].

En annan fördel med Ajax är att det till skillnad mot Flash och Java Applet inte kräver att någon plugin finns installerad i webbläsaren. Att installera en plugin kan tyckas vara ganska enkelt men det är dock en tröskel att ta sig över, särskilt för nybörjaranvändare. Macromedia menar att Flash Player finns installerat i 97,7 procent av alla webbläsare [17]. Om detta stämmer skulle det i och för sig betyda att detta inte är något större problem.

En del av de nackdelar som finns med Ajax är kopplade direkt till JavaScript. Dels måste JavaScript vara aktiverat i webbläsaren för att en Ajax-applikation överhuvudtaget ska kunna fungera. Eftersom koden inte kompileras så kan fel i

koden lättare slinka förbi oupptäckta, det finns olika verktyg för felsökning men dessa kräver att varje specifikt fall inträffar under körtid.

JavaScript beter sig på olika sätt under olika webbläsare. För att skapa en Ajax-applikation som fungerar under många webbläsare måste man hela tiden hantera dessa olikheter. Detta är både tidsödande och medför att programkoden inte blir elegant och lätthanterlig [18].

Det fanns en tid då JavaScript till största delen användes för att öppna upp popup-fönster fyllda med reklam. Detta ledde till att många användare rekommenderades att avaktivera JavaScript i deras webbläsare och skapade en viss misstro mot JavaScript. När dessa användare kommer in på en Ajax-baserad webbsida kommer de att rekommenderas att aktivera JavaScript igen.

Kommunikationen mellan klient och server är mer begränsad i en Ajax-applikation jämfört med både Applet och Flash. Ajax-applikationer använder enbart Http-protokollet till kommunikation vilket innebär att klienten bara kan få information från servern efter förfrågning. Flash och Java Applet har stöd för socket-kommunikation och därmed kan servern initiera händelser [19]. Detta kan visserligen lösas med hjälp av Comet-tekniken, men det är inte säkert att denna teknik är lika skalbar.

Ytterligare kritik som riktas gentemot Ajax-applikationer handlar om att dessa har en förmåga att bryta mot etablerade användarbeteenden i en webbmiljö. Användaren är van att bland annat kunna navigera med ”Bakåt”-knappen [20] och spara sidor som bokmärken [21]. När innehållet på en webbsida blir dynamiskt är det inte alltid säkert att det är möjligt att spara webbsidans aktuella tillstånd.

2.7 Framtidsutsikter

För att det ska gå lättare att skapa Ajax-applikationer håller flera företag på att ta fram olika ramverk och utvecklingsmiljöer. IBM har tillsammans med ett flertal stora aktörer på marknaden inlett arbetet med Open Ajax. Microsoft har skapat ett eget ramverk som de kallar för Atlas.

Ajax är hett just nu och det kommer ständigt nya applikationer. I och med att applikationerna blir mer och mer komplicerade ställs allt större krav på det underliggande systemet ska vara robust och skalbart. Frågan är om Ajax kommer att hålla måttet.

3 Nuvarande applikation

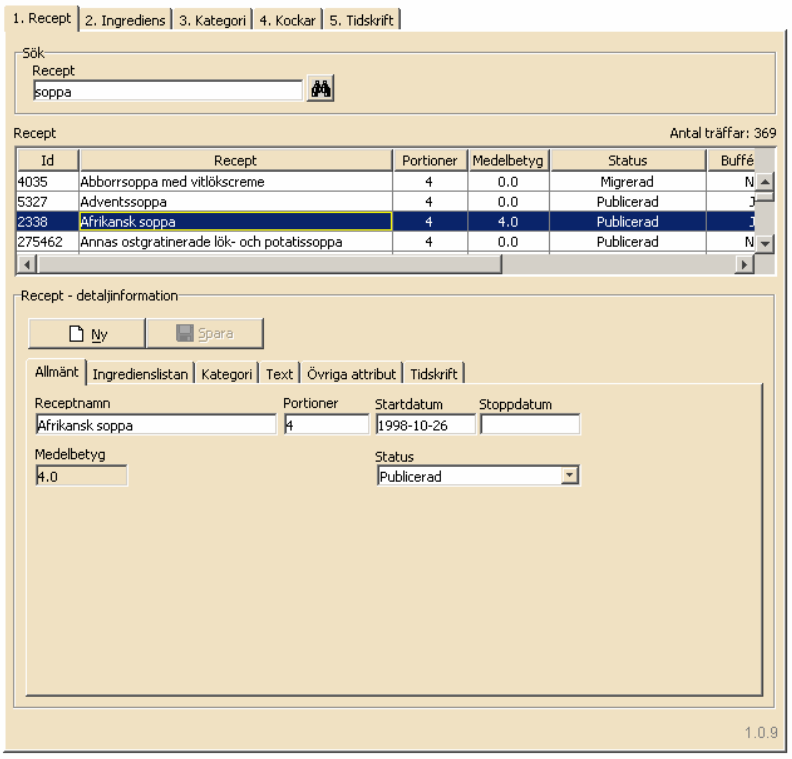
3.1 Översikt av applikationen

Den nuvarande applikationen för receptadministration är uppdelad i fem huvuddelar vilka är indelade i flikar:

- *Recept*
- *Ingrediens*
- *Kategori*
- *Kockar*
- *Tidskrift*

Recept är själva huvuddelen av applikationen och det är i den delen man bygger upp samt redigerar recept. Under *Ingrediens* skapar man och redigerar ingredienser, dessa kan sedan användas i flera recept. Övriga delar fyller inte så stor funktion. I delarna *Kategori* och *Tidskrift* kan användaren se vilka kategorier och tidskrifter som finns tillgängliga. Under *Kockar* är tanken att en användare ska kunna hantera olika kockar på samma sätt som med ingredienser men denna del är inte färdigutvecklad.

Som användare har man möjligheten att både skapa nya recept samt redigera befintliga. Väljer man att skapa ett nytt recept får man tillgång till ytterligare sex flikar, vilka kan ses i Figur 9.



Figur 9: Applet för receptadministration.

Information matas för det mesta in som text i olika fält. Exempel på information som fylls i på detta sätt är recepttexter, antal portioner, start och stoppdatum och så vidare. Annan typ av information såsom ingredienser, kategorier och tidskrifter får av naturliga skäl en annan inmatningsmetod. Man väljer kategorier och tidskrifter genom att markera dessa i en lista och klicka på en ”lägg till”-knapp. Ingredienslistan byggs upp genom att man söker efter ingrediensen i databasen, väljer ingrediens ur träfflistan, fyller i beskrivande information till ingrediensen och lägger sedan till denna i ingredienslistan. Ingredienslistan kan delas in med underrubriker vilket används om receptet innehåller en sås eller dylikt. När listan är uppbyggd är det möjligt att redigera ingrediensernas egenskaper och även ändra deras inbördes ordning i listan.

Det går att förflytta sig mellan de olika delarna i applikationen utan att någon information går förlorad. Om man kommer på att en ingrediens inte finns i databasen kan man alltså växla till ingrediensfliken, lägga till ingrediensen och sedan fortsätta att fylla i resten av receptet.

3.2 Arkitektur

3.2.1 MVC

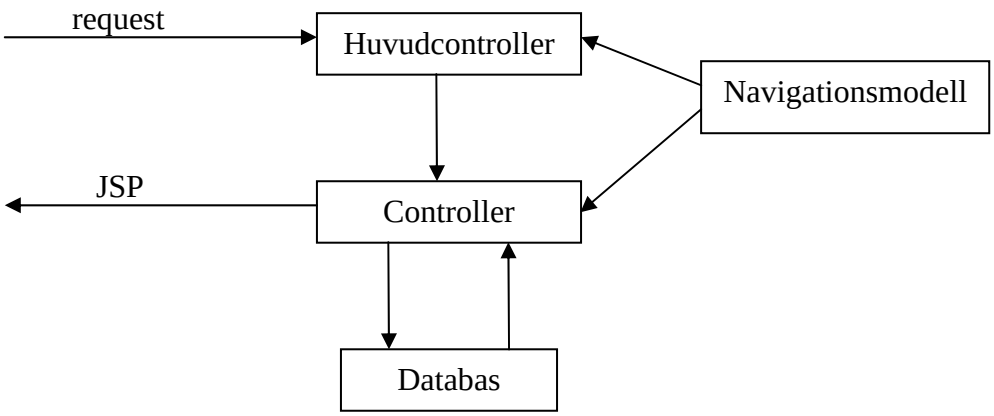
Model View Controller (MVC) är ett designmönster (design pattern) som används för att dela in en applikation och/eller delar av en applikation i olika komponenter. Genom att hålla komponenterna separerade från varandra kan man göra modifieringar i en komponent utan att de övriga komponenterna påverkas i alltför hög grad. MVC delar in en applikation i följande tre komponenter:

- *Model* – Applikationens datamodell.
- *View* – Den del av applikationen som presenteras för användaren, ofta i form av ett grafiskt användargränssnitt.
- *Controller* – Svarar på händelser och uppdaterar datamodellen samt vyn.

Tanken med en MVC-arkitektur är att applikationens datamodell och vy har så lösa kopplingar till varandra som möjligt. Det är controllerns uppgift att koppla ihop datamodellen och vyn. Controllern tar emot händelser från applikationens användargränssnitt och uppdaterar därefter datamodellen. När ändringar sker i datamodellen kan även vyn behöva uppdateras. Om MVC mönstret används strikt så sköts även uppdateringar i vyn utav controllern, en annan möjlighet är att låta vyn ”lyssna” efter förändringar direkt ifrån datamodellen [22].

3.2.2 ICA.se

ICA.se använder en MVC-arkitektur som är baserad på Java Servlet och JavaServer Pages (JSP), se Figur 10. Alla request-meddelanden tas om hand av en huvud-controller som beroende på parametrar i request-meddelandet anropar en specialiserad controller. Vilken controller som ska anropas finns definierat i en navigationsmodell. Controllern består av en Java-klass som har tillgång till datamodellen vilken består av en databas. När controllern har hämtat och/eller uppdaterat information i databasen bygger den upp en ny vy, i form av en JSP, som därefter skickas tillbaka som svar till den anropande klienten.



Figur 10: ICAs arkitektur.

3.2.3 Dataöverföring

All kommunikation mellan appleten och servern sker enligt ICAs MVC-modell. Appleten skickar data till servern som parametrar inbakade i request-meddelanden. Dessa är ordnade som namn/värde-par vilka är separerade med &-tecken. Svaret från servern är däremot i XML-format, vilket har byggts upp av en JSP. Figur 11 visar svaret från servern om man väljer att söka efter ingrediensen paprika.

Figur 11: Data som skickas från server.

3.2.4 Applikationen i Ajax?

Är det möjligt/lämpligt att implementera receptapplikationen med Ajax? Detta blir ett ställningstagande som man i stort sett får ta baserat på vad man tror och vilken erfarenhet man har. Den erfarenhet jag har av Ajax-applikationer är att jag har använt en hel del av de som finns tillgängliga och därmed fått en känsla av vad som fungerar.

Weiss [2] menar att Ajax kommer att göra det möjligt att flytta traditionella skrivbordsapplikationer såsom Microsoft Office ut till webben. Det finns flera exempel på när sådana överföringar också har genomförts. I detta fall handlar det om att en befintlig webbapplikation ska konverteras till en webbapplikation baserad på annan teknik. Detta ser jag som en betydligt mer genomförbar uppgift.

Eftersom det är möjligt att med XMLHttpRequest både skicka och ta emot data på samma format som används i den nuvarande applikationen ser jag inga problem med att integrera en Ajax-applikation i ICAs arkitektur.

3.3 Brister i applikationen

Den nuvarande applikationen innehåller en del brister, dels när det gäller applikationens användbarhet men det finns även ett antal rent funktionella brister. För att lättare kunna identifiera brister i applikationens användbarhet har jag tagit hjälp av olika riktlinjer. Det finns flera olika riktlinjer att välja mellan, jag har valt Nielsens tumregler och Schneidermans 8 gyllene regler [23]. Ofta påminner de olika riktlinjerna mycket om varandra och jag har därför tagit intryck ifrån båda av dessa och bakat ihop dem till följande sex utvärderingskriterier:

1. Enkelhet
Ett användargränssnitt bör vara så enkelt utformat som möjligt. Onödiga element i applikationen gör att användaren lättare kan tappa fokus och missförstå hur applikationen fungerar. Om man kan ta bort ett element ifrån ett användargränssnitt och applikationen fortfarande fungerar så bör elementet förbli borttaget

2. Ledtrådar
När det är möjligt ska användaren erbjudas ledtrådar som gör att han/hon lättare kan komma på hur applikationen fungerar. Det är bättre om användaren kan känna igen sig istället för att behöva komma ihåg hur systemet fungerar.

3. Alternativ för expertanvändare
Man bör erbjuda erfarna användare alternativa vägar för att kunna utföra operationer mer effektivt. Detta kan till exempel vara i form snabbkommandon via tangentbordet. Om detta fungerar på ett bra sätt kan både nybörjare och experter använda applikationen utan att applikationen utgör ett hinder för dem.

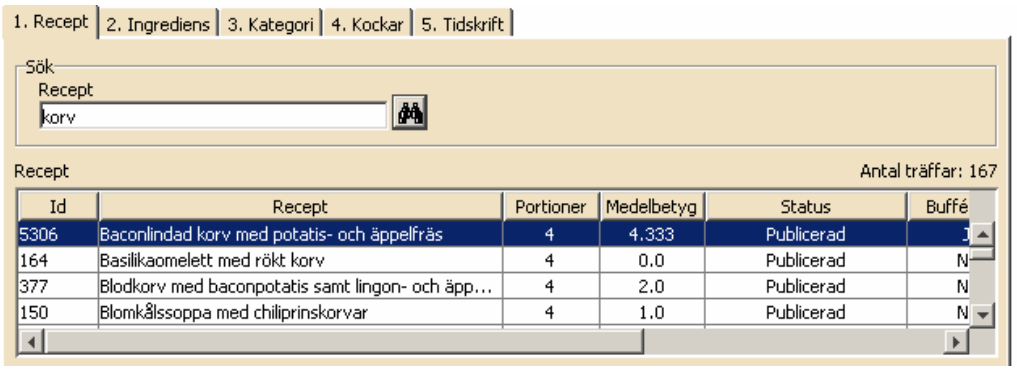
4. Återkoppling (feedback)
När en användare utför en operation i en applikation bör systemet svara på ett tydligt sätt för att indikera att operationen blivit utförd, alternativt visa ett felmeddelande om operationen av någon anledning inte kunde utföras.

5. Felhantering och felmeddelanden
Användargränssnitt bör i så stor utsträckning som möjligt förhindra användaren att göra fel. Om ett fel ändå uppstår bör användaren få ett informativt felmeddelande. Ett bra felmeddelande gör att användaren förstår vad som har gått fel så att han/hon kan avhjälpa felet.

6. Möjlighet att ångra sig
Det bör vara möjligt för en användare att enkelt kunna ångra sig. Både genom att kunna avbryta en pågående operation, men även att kunna ångra genomförda operationer. Detta kan leda till att användaren kan känna sig mindre orolig för att göra fel. I förlängningen kan det dessutom leda till användaren vågar utforska applikationen mer och därigenom få en bättre förståelse för hur den fungerar.

3.3.1 Användbarhet

Sökning av recept
Vid sökning av recept visar träfflistan all information som finns lagrat i receptposten i databasen. Detta leder till att information som inte är särkilt intressant visas i träfflistan, till exempel receptets medelbetyg. Följden av detta blir dessutom att information i sökresultatet hamnar utanför applikationens ram, se Figur 12.



Figur 12: Träfflista vid receptsökning

Inmatningsfältet och träfflistan är alltid synliga under receptfliken. Detta gör att träfflistan inte kan vara hur stor som helst och den har därför begränsats till att bara visa 4 recept åt gången. Om man använder sökordet ”korv” får man 167 olika recept i träfflistan. I detta sammanhang verkar storleken på träfflistan relativt liten. Dessa komponenter tar tillsammans upp ungefär en tredjedel av hela applikationens utrymme.

Att välja ett recept ur träfflistan går bra om man använder musen som inmatningsdon, men om man vill välja recept med hjälp av tangentbordet fungerar det inte lika bra. När markören har nått sista ingrediensen kommer den att vid nästa tabbning hoppa upp till den första ingrediensen igen. Det går med andra ord inte att med hjälp av tangentbordet förflytta sig ifrån träfflistan.

Sökfunktionen i applikationen skulle kunna göras bättre. Det finns en hel del information som kan tas bort ur träfflistan utan att den blir mindre användarvänlig. Det som är intressant i sökresultatet är receptets titel och i viss mån receptets id-nummer. Det senare kan vara bra att ha med eftersom det inte

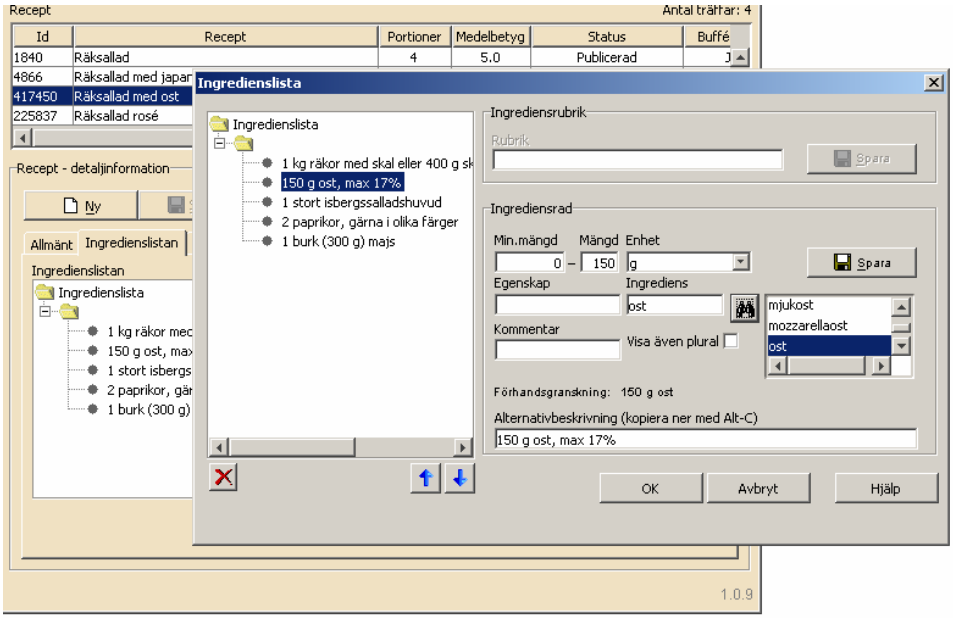
finns något funktionellt hinder för att två recept har samma titel. Att kunna söka fram ett recept är visserligen en central del av applikationen men när användaren väl har hittat det recept som ska redigeras är behovet av sökfunktionen borta. Applikationens främsta användningsområde är att lägga till nya recept och i detta fall används inte sökfunktionen alls.

Lägga till ingredienser

Ingredienserna ligger ordnade i olika listor. Den första av listorna innehåller receptets huvudingredienser och övriga listor innehåller ingredienser som räknas som tillbehör. En lista skapas genom att användaren skapar en rubrik och under denna rubrik kan därefter ingredienserna läggas till. När recepten visas på ICA.se har huvudingredienserna alltid rubriken ”Ingredienser”, oavsett vad användaren har specificerat i rubriken. Rubriken måste dock sättas vilket leder till att det första man som användare gör när man ska lägga till ingredienser är att skapa en ”tom” rubrik innehållande ett blanksteg.

Inmatningen av en ingrediens och dess beskrivande information är upplagd efter samma ordning som ingrediensen presenteras på ICA.se. Detta innebär att man som användare, om man följer ordningen, först anger ingrediensens mängd, enhet och egenskap innan man väljer vilken ingrediens som ska läggas till. För att få ett ingrediensnamn i plural måste detta anges innan sökningen. Om man söker efter ingrediensen ”tomat” och sedan kommer på att det skulle vara två tomater måste man göra om sökningen med rutan ”Visa även plural” markerad, trots att detta är samma ingrediens i databasen.

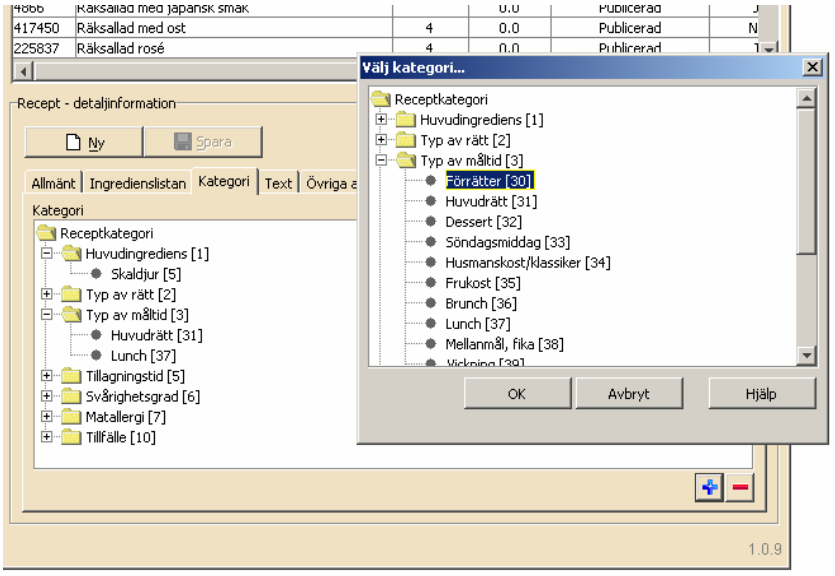
Knappen för att spara ingrediensen ligger placerad högst upp till höger inom ramen. Detta är en lite märklig placering eftersom inmatningen av ingrediensen sker uppifrån och ner. Det är lätt att av misstag istället klicka på OK-knappen som är placerad direkt under sista inmatningsfältet, men om man gör detta så stängs popup-fönstret och informationen går förlorad. Det går dessutom inte att med hjälp av tangentbordet förflytta sig till Spara-knappen. Figur 13 visar inmatningsfönstret för ingredienser.



Figur 13: Inmatning av ingredienser.

Lägga till kategorier

För att lägga till en kategori klickar man på en ”+”-knapp vilket leder till att ett popup-fönster öppnas. I detta fönster väljer man kategori i en hierarkiskt ordnad lista. När man valt en kategori klickar man på en ”OK”-knapp varefter popup-fönstret stängs, därefter finns kategorin synlig i kategorilistan. Det går inte att lägga till flera kategorier samtidigt utan denna procedur måste upprepas för varje kategori som ska läggas till. För att kunna se vilka kategorier som finns tillgängliga måste man som användare antingen välja att lägga till en ny kategori alternativt växla till *Kategori*-delen i huvudmenyn. Figur 14 visar hur inmatningsfönstret för kategorier ser ut.



Figur 14: Inmatning av kategorier.

3.3.2 Funktionella brister

Tidskrifter
I dagsläget är det inte möjligt att lägga till nya tidskrifter genom applikationen. För att göra detta måste de på marknadsavdelningen istället ta kontakt med systemutvecklarna, dessa lägger sedan in tidskriften manuellt i databasen.

Spara recept
När ett recept sparas för första gången lägger en ”recept-controller” i servern till information i databasen som inte finns tillgänglig i applikationen. Detta rör olika id-nummer som delar av receptet får för att de ska kunna kopplas samman i databasen. Applikationen uppdaterar dock inte alla dessa id-nummer när receptet har sparats. Om man sparar ett nyskapat recept, ändrar information i någon av recepttexterna och sedan sparar receptet igen så kommer inte ändringarna att uppdateras i databasen. För att kunna ändra på denna typ av information måste man se till att applikationen har en uppdaterad version av receptet inläst, detta kan åstadkommas genom att man söker efter receptet och läser in det i applikationen på nytt.

Försvinnande popup-fönster
Popup-fönster används ibland då information ska matas in. Dessa hamnar i ett eget fönster och är inte bundna till webbsidan. Om man klickar vid sidan av popup-fönstret hamnar detta i bakgrunden och går inte att komma åt ifrån verktygsfältet. Alla fönster framför popup-fönstret måste minimeras för att det ska bli synligt igen.

Logiska motsägelser
Det är möjligt att i ett recept lägga in kategorier som strider mot varandra logiskt. Man kan till exempel skapa ett recept som både har en tillagningstid på ”max 7 minuter ” och ”över 20 minuter”.

Ångra/upprepa (undo/redo)
Applikationen erbjuder inte någon möjlighet för en användare att ångra sig. Om man tar bort alla ingredienser i ett recept och receptet inte är sparat så måste man helt enkelt lägga till alla ingredienser igen vilket är ganska tidsödande.

4 Implementation

4.1 Kravspecifikation

Kraven för applikationen kan delas in i olika nivåer beroende på hur pass viktiga de är för slutresultatet. Kraven för implementationen har delats in i tre nivåer.
Det som måste uppfyllas, det som bör uppfyllas och slutligen saker som kan vara bra att göra om det finns tid över.

4.1.1 Krav som måste uppfyllas

- Applikationen ska fungera under ICAs standardinstallation. Detta betyder i dagsläget att den ska kunna köras under Internet Explorer 6.

- Applikationen ska vara kompatibel med ICAs nuvarande webbarkitektur, detta innebär att den ska kunna använda sig av befintliga controller-funktioner.

- Den nyutvecklade applikationen och den gamla applikationen ska kunna köras parallellt. Eventuella problem/fel i servern måste därmed hanteras i applikationen.

- Applikationen ska vara uppbyggd i moduler. Delar av applikationen ska enkelt kunna bytas ut och det ska även vara möjligt att bygga ut applikationen.

4.1.2 Krav som bör uppfyllas

- Det ska gå att lägga in nya tidskrifter med applikationen. Denna funktion finns inte i dagsläget och måste därför implementeras både i applikationen och i servern.

- Användargränssnittet ska göras tydligare och interaktionen förenklas. De brister som identifierats i den gamla applikationen ska så långt som det är möjligt byggas bort.

- Funktionella brister i befintliga funktioner ska rättas till. Det finns vissa fel som i dagsläget upptäcks först i servern. Dessutom finns det områden där det är upp till användaren att inte göra fel. Applikationen bör vara konstruerad så att så få fel som möjligt kan uppstå.

- Applikationen bör fungera i alla webbläsare som har stöd för Ajax.

4.1.3 Om tid finns

- De olika sökfunktionerna i applikationen kan göras mer avancerade. De skulle till exempel kunna erbjuda ”komplettera automatiskt”-funktionalitet.

- Applikationen skulle kunna erbjuda en högre grad av direkt manipulation. Det vore till exempel smidigt med drag&drop för att ändra ordningen i ingredienslistan.

- Bygga in stöd för undo/redo så att applikationen blir mer förlåtande.

4.2 Verktyg

Implementationen av applikationen har kommit till med hjälp av ett flertal olika verktyg.

Alla delar som har med ICAs webbarkitektur att göra har utvecklats i IBM Websphere Studio Application Developer 5.1.2.

För att skriva JavaScript-koden har jag använt mig av TextPad.

Applikationen använder enskilda HTML-dokument för de olika popup-fönstren, dessa har skapats med hjälp av CSE HTML Validator Lite 6.52.

Applikationen har testats under Internet Explorer 6.0 och till viss del Mozilla Firefox 1.0. För att enklare kunna hitta fel i JavaScript-koden har jag använt mig av Microsoft Script Debugger. Dessutom har jag använt IEWatch 3.1.2 för att kunna inspektera http-meddelanden samt IE DOM Explorer för att hålla koll på användargränssnittets dynamiska delar.

4.3 Funktion

Applikationens vy skapas i webbservern, detta sker med hjälp av en JSP. I denna specificeras strukturen på hur användargränssnittets beståndsdelar skall vara ordnade. En viktig del i utvecklingen av applikationen har varit att det ska vara relativt enkelt att förändra eller bygga ut delar av applikationen. Så långt det är möjligt byggs därför de olika elementen i applikationen upp dynamiskt, baserat på hur de representeras i databasen.

Applikationen består av moduler vilka är uppbyggda enligt MVC-arkitekturen. Applikationens datamodell representeras som objekt skapade i JavaScript. Det huvudsakliga objektet är själva receptobjektet, detta innehåller övriga objekt såsom ingredienslistor, kategorilistor och så vidare.

För att applikationens vy och modell hela tiden ska vara i fas sker uppdateringar alltid på följande sätt. När användaren har matat in information aktiveras en händelsehanterare. I de fall inmatningen sker i textfält används exempelvis textfält-elementets inbyggda händelsehanterare *onChange*. Händelsehanteraren anropar en controller som därefter uppdaterar datamodellen. När detta är gjort

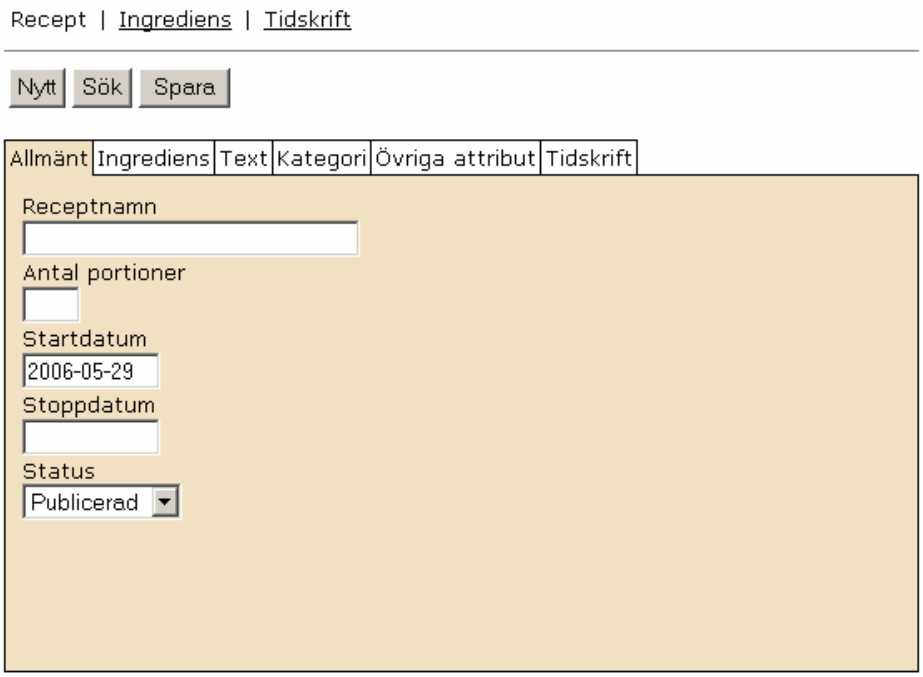
anropar samma controller en uppdateringsfunktion som ser till att vyn motsvarar den aktuella datamodellen. För att åstadkomma detta på enklaste sätt tar uppdateringsrutinen bort det aktuella elementet från vyn och lägger sedan in ett nytt element igen. Detta är kanske inte den effektivaste metoden att uppdatera vyn på, men den säkerställer att vyn och modellen alltid överensstämmer.

Kommunikationen med servern sker genom en kommunikationsmodul, denna modul anropas av en controller i applikationen och skickar därefter informationen till servern med hjälp av ett XMLHttpRequest-objekt. När svaret anländer från servern skickas detta tillbaka till controllern.

4.4 Användargränssnitt

Jag har valt att behålla den generella strukturen i användargränssnittet. Detta för att användarna ska kunna känna igen sig. De ska kunna komma igång och använda applikationen direkt utan större svårigheter. De brister som uppkommit i utvärderingen av den gamla applikationen har jag adresserat på följande sätt.

För att applikationen ska få ett enklare utseende har jag valt att plocka bort, alternativt dölja element som inte är av central betydelse. Detta har inneburit att flikarna *Kategori* och *Kockar* har tagits bort helt och hållet från huvudmenyn då de i dagsläget inte erbjuder någon interaktion med användaren. Vidare har jag valt att ta bort sökningen av recept från användargränssnittet. Detta ligger istället i ett separat popup-fönster som är åtkomligt via en knapp. Förutom att användargränssnittet får ett enklare utförande resulterar detta i att fler recept kan visas i träfflistan.



Figur 15: Övergripande vy av Ajax-applikationen.

När det gäller inmatning av kategorier har jag helt och hållet gått ifrån den gamla applikationens interaktionsmetod. Jag väljer istället att representera kategorierna i checklistor då detta är en etablerad funktion i webbmiljö. Detta gör att användaren hela tiden kan se vilka kategorier som finns tillgängliga under varje rubrik, på så sätt blir användaren påmind om vilka kategorier som kan vara applicerbara för det aktuella receptet.

Jag har även valt att bygga om ingrediensinmatningen. Alla enkla operationer, de som görs i ett eller två steg ligger åtkomliga för användaren direkt. Att lägga in en ny ingrediens är en mer komplicerad process och detta sker istället stegvis i ett popup-fönster. Först får användaren söka reda på en ingrediens, i nästa steg kan ingrediensens egenskaper fyllas i. Detta gör att användaren inte kan börja fylla i information om ingrediensen och sedan upptäcka att ingrediensen inte finns inlagd i databasen.

5 Resultat

5.1 Hur har kraven uppfyllts

Applikationen uppfyller alla krav som måste vara uppfyllda och dessutom flera av de krav som bör vara uppfyllda. Det som inte hunnits med är att göra applikationen kompatibel under flera webbläsare, mer om detta under nästa rubrik. Det har inte heller funnits någon tid att uppfylla de lösare kraven men dessa har funnits i baktanken under utvecklingen och det bör inte vara något problem att bygga ut applikationen om så skulle önskas. Applikationen är i ett funktionellt skick och kommer med stor sannolikhet att börja användas.

5.2 Problem och reflektioner

Den mest besvärliga delen av programutvecklingen har varit att få applikationen att fungera under olika webbläsare, detta har resulterat i att applikationen för närvarande bara fungerar felfritt i Internet Explorer. I början av arbetet försökte jag att kontinuerligt testa applikationen under olika webbläsare för att säkerställa att den fungerade under olika miljöer. Detta drog så småningom ut på tiden och gjorde att en alltför stor del av programmeringen gick åt till att lösa samma problem på flera olika sätt. En bidragande orsak till detta problem är att det har varit ganska svårt att hitta bra dokumentation som specificerar hur olika objekt ska användas. JavaScript lämnar dessutom en hel del att önska när det gäller informationen i olika felmeddelanden. Ibland har jag sökt efter hur ett visst objekt ska användas, hittat ett flertal exempel och därmed övertygats om att jag har gjort på rätt sätt. När det efter åtskilliga försök fortfarande inte har fungerat och det känns som att jag har provat allt, har jag kommit fram till att det måste vara något annat som felar. Till slut har jag hittat ett exempel som visar olika undantagsfall.

Det finns ett flertal olika ramverk som kan användas för att förenkla utvecklingen av Ajax-applikationer. Jag har valt att inte använda mig av dessa då jag har velat få full insikt i hur tekniken fungerar.

När jag skulle börja integrera Ajax-applikationen med ICAs arkitektur gick det inte att ta emot det XML-dokument som servern skickade. I början av utvecklingen hade jag skapat egna controller-klasser för att snabbt kunna komma igång med att hämta data ifrån servern. Dessa skickade XML-dokument till applikationen direkt med hjälp av en Java-metod. När jag provade att anropa servern med hjälp av webbläsaren fick jag till synes exakt samma svar med båda controller-klasserna. Det som visade sig vara felet var att det i JSP:n inte specificerats att innehållet var i XML-format. XMLHttpRequest är mindre förlåtande för fel än vad både Appleten och webbläsaren är vilket gjorde att jag inte upptäckte felet förrän jag började granska response-meddelandena i detalj. Lyckligtvis gick det att korrigera felet utan att detta påverkade Appleten.

Den information som fanns tillgänglig om Ajax var i början av examensarbetet relativt tunn. Det fanns två likvärdiga böcker skrivna om Ajax och ett fåtal artiklar som inte innehöll mycket mer än en introduktion till tekniken. Under arbetets gång har antalet böcker om Ajax ökat markant och ett flertal artiklar har publicerats.

5.3 Framtida arbete

För att göra applikationen mer framttidssäker och åtkomlig bör den byggas ut så att den går att använda under alla större webbläsare. Detta är ett arbete som inte bör vara särskilt svårt, men det kan vara ganska tidsödande.

Sökningar som är genererar många träffar blir ganska långsamma i båda applikationerna. Detta beror på att servern använder en relativt enkel sökalgoritm. På andra delar av ICA.se används Lucene[9] som sökmotor. Ett naturligt steg vore att även börja använda Lucene åt dessa sökningar. Detta implementeras helt och hållet i servern och det finns alltså inget i applikationen som begränsar.

De huvudsakliga användarna av applikationen har tyvärr, på grund av tidsbrist från deras sida, kommit in ganska sent i utvecklingen av applikationen. Problem eller önskemål som kommer upp när applikationen börjar användas regelbundet kommer därför att behöva fångas upp i ett senare skede.

5.4 Slutsats

Det har varit både utmanande och lärorikt att arbeta med Ajax. Innan jag påbörjade detta examensarbete hade jag inte arbetat i webbmiljö nämnvärt och jag hade aldrig kommit i kontakt med JavaScript. Detta har inneburit att jag har haft mycket information att ta in, men det kan också ha varit till en fördel eftersom jag har kunnat ta mig an uppgiften förutsättningslöst.

Den stora fördelen som jag ser med Ajax är att det bygger på vanlig webbteknik. Det går till viss del att ”smyga” in applikationer hos användarna utan att dessa behöver göra någon motprestation. Självklart kommer det alltid att finnas användare som inte har de tekniska förutsättningar som krävs för att kunna använda Ajax-applikationer. Därför är det viktigt att man tänker igenom vilken målgrupp man har och om möjligt erbjuder en mindre avancerad webbapplikation som dessa användare kan falla tillbaka på. Med en genomtänkt design kan man dra stora fördelar av att Ajax-applikationer och traditionella webbapplikationer har samma byggstenar.

Det har dykt upp många olika Ajax-applikationer på marknaden och flera andra är på gång. Som alltid när en ny teknik blir populär så finns det en risk att den används bara för sakens skull. Jag tror helt klart att det finns en framtid för avancerade webbapplikationer. Om dessa kommer att vara baserade på Ajax eller inte är dock svårt att avgöra.

[9] http://lucene.apache.org/java/docs/index.html

6 Tack

Jag vill tacka mina handledare på ICA, Martin Lindahl och Johan Qwerin samt min interna handledare på CS, Thomas Johansson för allt som ni har bidragit med under examensarbetets gång.

7 Referenser

1. Wikipedia. *Web Application* http://en.wikipedia.org/wiki/Web_application, (besöksdatum 2006-01-16)

2. Weiss. A. *WebOS Say Goodbye to Desktop Applications* netWorker, Volym 9, Nummer 4, December 2005, s. 18-26.

3. Garret. J. J. *Ajax: A New Approach to Web Applications* 2005-02-18 http://www.adaptivepath.com/publications/essays/archives/000385.php, (besöksdatum 2006-01-16).

4. Baekdal. T. *AJAX, XMLHTTPRequest or AHAX* 2005 http://www.baekdal.com/articles/Technology/ajax-xmlhttprequest-ahax/, (besöksdatum 2006-01-16).

5. Crane. D. et al. *Ajax in Action* Manning Publications Co, Greenwich, 2006.

6. Google *What web browsers does Google Maps support?* http://maps.google.com/support/bin/answer.py?answer=16532&topic=1499, (besöksdatum 2006-05-29).

7. W3C. *W3C Document Object Model* 2005-01-19 http://www.w3.org/DOM/, (besöksdatum 2006-01-16).

8. W3C. *The XMLHttpRequest Object* 2006-04-05 http://www.w3.org/TR/XMLHttpRequest/, (besöksdatum 2006-05-08).

9. McLaughlin. B. *Advanced requests and responses in Ajax* 2006-02-14, http://www.ibm.com/developerworks/web/library/wa-ajaxintro3/index.html, (besöksdatum 2006-04-28).

10. Apple Developer Connection *Dynamic HTML and XML: The XMLHttpRequest Object* http://developer.apple.com/internet/webcontent/xmlhttpreq.html, (besöksdatum 2006-04-28).

11. McLaughlin. B. *Make Asynchronus requests with JavaScript and Ajax* 2006-01-17, http://www.ibm.com/developerworks/web/library/wa-ajaxintro2/, (besöksdatum 2006-04-28).

12. McLaughlin. B. *Manipulate the DOM* 2006-04-11, http://www.ibm.com/developerworks/web/library/wa-ajaxintro5/index.html, (besöksdatum 2006-04-28).

13. Wikipedia. *Comet (programming)* http://en.wikipedia.org/wiki/COMET_%28programming%29, (besöksdatum 2006-05-15).

14. Eagle. M. *Integrating Macromedia Flex with Java* 2004-12-01 http://www.onjava.com/pub/a/onjava/2004/12/01/flexjava.html, (besöksdatum 2006-03-21).

15. White. A. *Weighing the alternatives* 2005-08-30 http://www.ajaxinfo.com/default~viewart~8.htm, (besöksdatum 2006-01-16).

16. King. B. *Remote Application Development with Mozilla* 2002-12-17 http://www.oreillynet.com/pub/a/mozilla/2002/12/17/app_dev.html?page=2, (besöksdatum 2006-01-20).

17. Adobe *Macromedia Flash Player Statistics* 2006-04 http://www.adobe.com/products/player_census/flashplayer/, (besöksdatum 2006-05-22).

18. Boutelle. J. *Flash RIAs vs. Javascript RIAs* 2005-03-03 http://www.jonathanboutelle.com/mt/archives/2005/03/flash_rias_vs_j.html, (besöksdatum 2006-02-02).

19. Wrobleswki. L. *Web Application Solutions: A Designer´s Guide* http://www.lukew.com/resources/WebApplicationSolutions.pdf, (besöksdatum 2006-01-20).

20. Field Expert *Ajax Best Practice´s: Don´t Break The Back Button* 2006-01-03 http://www.fieldexpert.com/2006/01/03/ajax-best-practices-dont-break-back/, (besöksdatum 2006-02-02).

21. Field Expert *Ajax Best Practice´s: Don´t Break Bookmarks* 2006-01-04 http://www.fieldexpert.com/2006/01/04/ajax-best-practices-dont-break-bookmarks/, (besöksdatum 2006-02-06).

22. Moore. J. *Pattern Model View Controller – Web Application Component Toolkit* http://www.phpwact.org/pattern/model_view_controller, (besöksdatum 2006-05-10)

23. Preece. J. et al. *Interaction design: Beyond human-computer interaction (26-27, 266-267)* John Wiley and Sons, Inc, 2002.