2007:164 CIV

EXAMENSARBETE

Asynkron webbkommunkation

Web 2.0

Peter Wikström

Luleå tekniska universitet

Civilingenjörsprogrammet
Medieteknik
Institutionen för Systemteknik

2007:164 CIV - ISSN: 1402-1617 - ISRN: LTU-EX--07/164--SE

KnowIT

Asynkron webbkommunikation

Web 2.0

Peter Wikström
1/1/2007

Sammanfattning

Detta examensarbete har haft som uppgift att undersöka möjligheterna för hur man skulle kunna förbättra gränssnittet för användarna av WIS, Webbaserat informationssystem (1), genom att göra webbapplikationerna mindre dataintensiva.

Examensarbetet görs i samarbete med Know IT Candeo AB i Luleå som är utvecklare och förvaltare av systemet WIS.

Examensarbetet tar upp några exempel på hur många webbapplikationer fungerar och är uppbyggda idag. Perspektivet är hur man kan förbättra gränssnittet för användarna med hjälp av asynkron webbkommunikation.

De bakomliggande principerna för asynkron webbkommunikation kommer att behandlas för att sedan kunna beskrivas mer specifikt i AJAX(Asynchronous JavaScript and XML) tekniken. För att kunna utvärdera om asynkron webbkommunikation är användbart i just detta specifika projekt ska några enkla prototyper tas fram.

Abstract

The purpose of this master's thesis has been to investigate methods for improving the experience for users of WIS (1), by trying to make them less data intensive .

This thesis is done at Know IT Candeo AB who has developed the system and also acts as trustee.

This master's thesis will describe some of the techniques that many web applications use today and how these applications could be improved with asynchronous communication. The thoughts behind asynchronous communication will be described and more specific techniques such as AJAX (Asynchronous JavaScript and XML). In order to evaluate if asynchronous communication is suitable for this specific project some prototypes will be developed.

Innehåll

Figurer, exempel och tabeller

Förkortningar och begrepp

AJAX	Asynchronous JavaScript and XML.
ASP	Active Server Pages
PHP	Serverscriptteknik för Internet. Linuxbaserat
JSP	Java Server Pages. Javabaserat
ASP.NET	Active Server Pages(för .NET Framework)
Atlas	Microsofts implementation av AJAX för ASP.NET 2.0.
Visual Studio .NET	Microsofts utvecklingsmiljö för .NET applikationer.
WIS	Krisberedskapsmyndighetens webbaserade system för underlätta kommunikation under kriser.
HTML	HyperText Markup Language. Språket som används för att presentera data på Internet
CSS	Cascading Style Sheet. Ett sätt att definiera layout för html på Internet
JavaScript	Scriptspråk för webbläsare, syntax som Java
VBScript	Scriptspråk som ofta används i samband med ASP, syntax som Visual Basic
ActiveX	Teknik för att köra aktivt material på webbsidor. Utvecklat av Microsoft.
API	Application Programming Interface. Gränssnitt för utvecklare för att kunna använda ett bibliotek eller applikation.
DOM	Document Object Model. Webbläsarens API för manipulering av användarens gränssnitt.
IDE	Integrated Development Enviroment. En utvecklingsmiljö.
Http	HypeTExtTranfer Protocoll. Protokollet som används för att skicka data mellan klient och server på Internet
XML	Hierarkist sätt att representera datastrukturer.

1 Inledning

Inom webbutveckling har det länge funnits olika begrepp som ofta har hävdats vara den nya banbrytande tekniken som kommer att revolutionera upplevelsen av webben. Ett av dessa begrepp som har fått stor uppmärksamhet de senaste åren är AJAX(Asynchronous JavaScript and Xml). Idén bakom denna teknik är inte alls ny utan har funnits ganska länge. Remote Scripting (4) i Microsofts Active Server Pages är en av dessa som har funnits ända sen 1998. Grundtanken är att man inte ska behöva uppdatera hela webbläsaren varje gång en klient gör ett anrop till en server utan att man ska kunna uppdatera delar av en webbsida med asynkrona anrop. Resultat av detta är att man kan undvika scenarier där en klient blir låst under den tid som en server behandlar klientens anrop. Vilket leder till ett gränssnitt som känns mer dynamiskt och interaktivt. Utifrån detta så ville Know IT Candeo AB undersöka om det finns några fördelar med att implementera AJAX i WIS(Webbaserat informationssystem) 2.0 (1) som ska utvecklas under 2007 samt hur en implementation av AJAX i det befintliga systemet WIS 1.6 skulle kunna genomföras. Detta examensarbete är resultatet av dessa frågeställningar och är genomfört av en student på civilingenjörsprogrammet inom Medieteknik på Luleå tekniska universitet.

Rapporten har jag valt att skriva på svenska och detta val leder till att rapporten innehåller flera begrepp vilka är svåra att hitta en direkt översättning till. Dessa har markerats i kursiv stil.

1.1 Målgrupp

Målgruppen för denna rapport är i första hand studenter vid data och media programmen samt även yrkesverksamma systemutvecklare, webbutvecklare och IT-arkitekter.

1.2Syfte

Syftet med detta examensarbete är att undersöka hur man kan implementera AJAX i WIS 2.0 samt i det redan befintliga systemet WIS 1.6 med hjälp av att tillverka ett antal prototyper.

1.3 Frågeställning

Examensarbetet ämnar att besvara ett antal frågor gjorda tillsammans med Know IT Candeo AB.

- Principer samt för- och nackdelar med asynkron webbkommunikation, främst via AJAX och Microsofts implementation ASP.NET AJAX.
- Hur kan man använda AJAX för att göra applikationen mindre dataintensiv.

1.4 Avgränsningar

Sammanställningen av existerande och nya funktioner, lämpliga för att utökas med AJAX funktionalitet i WIS projektet, ska begränsas till ett fåtal under projektet funna funktioner. Examensarbetet kommer att fokusera på att utveckla prototyperna i Microsoft Visual Studio .NET 2005 för WIS 2.0 samt Visual Studio .NET 2003 för WIS 1.6.

2 Problemställning

För att kunna tillverka tillfredställande prototyper så väcktes följande frågor.

1. Hur påverkar AJAX redan implementerad kod?
2. Hur ska man gå tillväga för att minimera dataintesiteten?
3. På vilket sätt kommer den minskade datamängden att påverka kommunikationen mellan klient och server?

2.1 Förkrav

Kraven på implementationen är att det nya funktionaliteten inte ska påverka redan befintliga funktioner. Sedan ska implementationen göras från ett antagande där man tror att AJAX funktionalitet verkligen förbättrar applikationen. Implementationen ska bygga på standardiserade design rekommendationer. När det gäller implementation i nästa generation av WIS så är AJAX funktionaliteten med redan från start, det innebär en mer naturlig implementering av denna extra funktionalitet.

2.2 Krisberedskapsmyndigheten

KBM, Krisberedskapsmyndigheten (9), är en myndighet med uppgift att samordna arbetet med att utveckla krisberedskapen i det svenska samhället. KBM har kontor i Stockholm och Sollefteå. I Stockholm finns verksledningen och ledningen för alla enheter utom tekniska enheten, som finns i Sollefteå. På KBM arbetar 230 personer, varav 65 arbetar i Sollefteå (10).

2.2.2 Behov

Vid händelse av kris så har KBM fått i uppdrag av regeringen att främja användandet av Internet som informationskanal. Detta har lett till utvecklandet av WIS som tillhandahålls gratis för alla organisationer som vill ansluta sig till systemet. Målet är att detta system ska användas på bred front och underlätta vid krishantering samt leda till en samordnad spridning av krisinformation. Behovet av att göra ett bra och intuitivt gränssnitt för all typer av användare är av stort intresse vilket innebär att alla eventuella förbättringar som AJAX kan innebära är av stor vikt. För att öka användandet och intresset av systemet så håller KBM utbildningar i detta system för alla anslutna organisationer.

2.2.3 Verksamhetskrav

Detta examensarbete handlar till största del om hur man skulle kunna förbättra gränssnittet hos WIS genom att göra det mer levande och interaktivt. Kraven som finns på WIS innehåller väldigt mycket mer än vad detta examensarbete kommer att ta upp men några specifika krav som handlar om det grafiska gränssnittet kan man ta fasta på. Ett

övergripande krav för det grafiska gränssnittet är att det ska kännas enkelt att använda med tydliga Microsoft Windows associationer.

2.3 WIS

WIS är ett icke hierarkiskt informationsdelningssystem som är generellt användbart före, under och efter kriser. Icke hierarkiskt menas här att alla som använder systemet har samma värde, ingen myndighet eller organisation står så att säga över någon annan. Systemet bygger på att organisationer och myndigheter ansluter sig kostnadsfritt till systemet och kan sedan dela med sig av information i dagboksform till varandra. Varje aktör har en krisledningsgrupp som agerar som redaktör i systemet och bestämmer huruvida informationen som en specifik användare har skrivit ska publiceras. WIS är byggt för icke-sekretessbelagd information och använder sig av krypterad kommunikation(128-bitars SSL[1]).

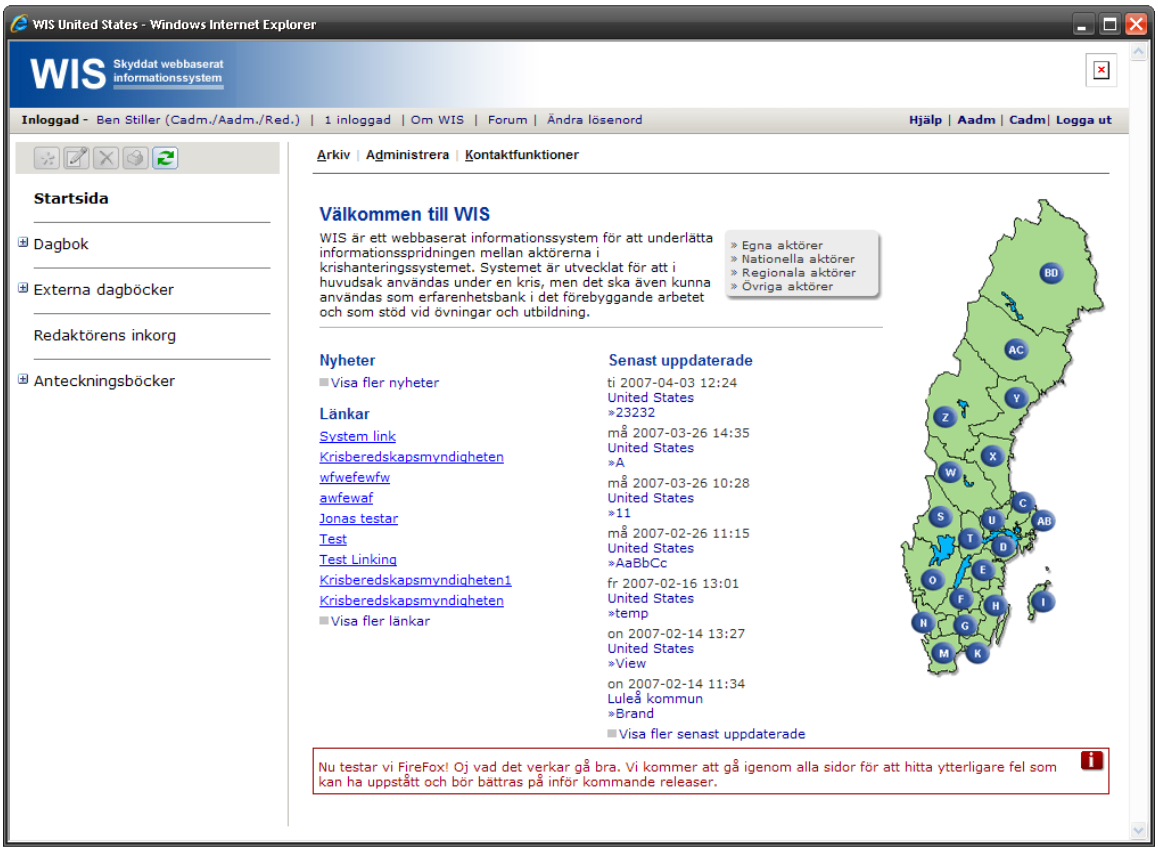
2.3.1 Allmänt

WIS, Webbaserat Informationssystem, är som namnet antyder ett system som är gjort för användning över Internet. Gränssnittet är html som laddas ner till klienten och uppdateras genom att skicka data tillbaka till servern eller anropa JavaScript funktioner på klientsidan. Hela systemarkitekturen är byggt enligt en domänobjektmodell som har starka band till ett objektorienterat sätt att programmera och strukturera. Detta lett till valet av C# som programmeringsspråk, dels tack vare dess objektorienterade natur samt dess goda integrering i Microsofts utvecklingsmiljö. Databasen som används är en relationsdatabas.

2.3.2 WIS 1.6

Denna generation av systemet är byggt på *Microsoft .NET Framework 1.1* och använder sig av ASP.NET som presentationslager. Affärslogiken i systemet är skriven i C# och som databas används *Microsoft SQL Server 2000*. För övrigt så används en del tredjepartsprodukter för vissa delar av systemet, t.ex. så används en *Aspose.Words* (5) komponent för export från html till *Microsoft Word*. En central del i systemet är komponenten där användarna skriver in sina anteckningar som sedan publiceras. Även den är en tredjepartsprodukt kallad *FCKEditor* (6). Den gör det möjligt att formatera texten som man vill ha den innan man lagrar informationen i systemet som html. Det finns även andra delar av systemet som använder tredjepartsprodukter såsom loggning och statistik.

[1] Secure Socket Layer



Figur 1 Startsidan för WIS 1.6

2.3.3 WIS 2.0

Nästa generation av WIS (WIS 2.0) kommer att vara byggt på *Microsoft .NET Framework 2.0* vilket möjliggör användning av ASP.NET AJAX Extensions, Microsofts AJAX implementation. Detta leder till bättre integrering i utvecklingsmiljön och mer standardiserade klasser för AJAX funktionalitet.

3 Historik

Utvecklingen inom datakommunikation har gått från tunna klienter där det mesta jobbet gjordes på kraftfulla servrar till explosionen av PC-marknaden där man flyttade det mesta av logiken till klienterna. Detta skapade en miljö av relativt isolerade klienter utan större behov av att vara uppkopplade mot en server. De applikationer som behövde uppkoppling, såsom e-post och nyhets prenumerationer, kunde med lätthet köras med en terminal emulator. Allt detta förändrades när en ny typ av applikation slog igenom, webbläsaren. Med denna kunde man med en tunn klient ta del av information som var lagrad på centrala servrar. Informationen presenterades i form av bilder och text med hjälp av html(HyperText Markup Language)[2]. Behovet av mer och mer avancerade webbplatser tvingade webbläsarna att bygga in stöd för olika typer av scriptspråk som till exempel JavaScript[1] och VBScript[1]. Det språket som blev en typ av webbstandard var ECMA-Script senare känt som JavaScript. Tekniken var i grunden synkron, en klient begär information och väntar till en server levererar ett svar. Begränsningar i den synkrona kommunikationen gav upphov till att man i slutet på 1990-talet började titta på asynkron kommunikation i samband med webbläsare. Med hjälp av denna typ av kommunikation skulle man kunna begära information från en server utan att uppdatera hela sidan. Microsoft introducerade 1998 begreppet *Remote Scripting* (3) och i och med deras nya version av webbläsare Internet Explorer 5 byggdes stöd in för att kunna skapa ett ActiveX objekt som hanterade just asynkrona anrop till en server. Logiken på klientsidan byggde på JavaScript och svaret från servern levererades i form av XML(eXtensible Markup Language)[1]. Detta är grunden till AJAX(Asynchronous JavaScript And Xml). Dock så hade tekniken svårigheter att få fotfäste bland utvecklarna, mycket på grund av stora säkerhetsproblem med ActiveX komponenter. I dagens moderna webbläsare så har man istället för att använda sig av ActiveX[1] implementerat ett särskilt JavaScript objekt kallat XmlHttpRequest (4) som hanterar denna logik. Man brukar prata om Web 2.0 som ett samlingsbegrepp för nästa steg mot mer interaktiva och användarvänliga applikationer på Internet.

3.1 Webbapplikationer

Från en användares perspektiv förväntas idag en webbapplikation nästan ha samma funktionalitet som en ”vanlig” applikation som exekveras lokalt på en dator. Eftersom webbkommunikation mellan klient och server till stor del är uppbyggd av synkron kommunikation så uppfattas ofta ett webbgränssnitt som statiskt och långsamt i jämförelse med en vanlig *windows* applikation. Det finns dock exempel på webbapplikationer där man använder sig av asynkron kommunikation för att göra applikationen mer interaktiv och ge användarna ett mer levande gränssnitt. Detta är ett steg mot Web 2.0 (5).

[2] se sidan 8 för förklaring av begreppet

3.1.1 Allmänt

Traditionella webbapplikationer består av en webbserver som antingen tillhandahåller statiska webbsidor eller dynamisk information som levereras med hjälp av någon serverteknologi som t.ex. *ASP.NET*[3], *JSP*[2] eller *PHP*[2]. Med hjälp av dessa kan man göra mer dynamiska applikationer där informationen kan hämtas från en databas och sedan visas för användaren som html. På detta sätt kan man bygga mer levande applikationer som kan blir lättare att uppdatera, mer skalbara samt även bättre säkerhet. Nästa steg blir att isolera logiken på servern till en applikationsserver som kopplas till webbservern, detta leder till ännu bättre skalbarhet.

3.1.2 Teknik

En webbapplikation består av en klient, webbläsaren, samt en server, webbservern. Klienten begär information från server i form av ett *request-meddelande* (se exempel 1) som kan vara av olika typer men när det gäller att hämta information i form av html från en server används GET och när det gäller data som skickas från ett webbformulär till en server så används POST. Nedan finns ett exempel på hur ett sådant meddelande kan se ut. Meddelandet innehåller vilken typ av *request-meddelande* det är samt vilken fil som ska hämtas och på vilken server denna fil finns lagrad.

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Exempel 1 Ett http request meddelande

Servern besvarar ett sådant anrop med ett *response-meddelande* (se exempel 2). Detta meddelande innehåller status för den begärda sidan där klienten blir informerad om eventuella problem, vilken typ av information som sidan består av samt även själva html-koden för innehållet. I detta fall är status för just denna resurs *"200: OK"*, vilket innebär att allt har gått enligt planerna. Sidan finns tillgänglig och skickas till klienten. En annan typ av status är en vanlig syn på Internet, *"404: file not found*". Detta innebär att filen som klienten efterfrågade inte finns tillgänglig.

[3] se sidan 8 för förklaring av begreppet

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354
<html><body>
<h1>Exempel<h1>
Ett exempel på ett response meddelande
</body>
</html>
```

Exempel 2 Exempel på ett http response meddelande

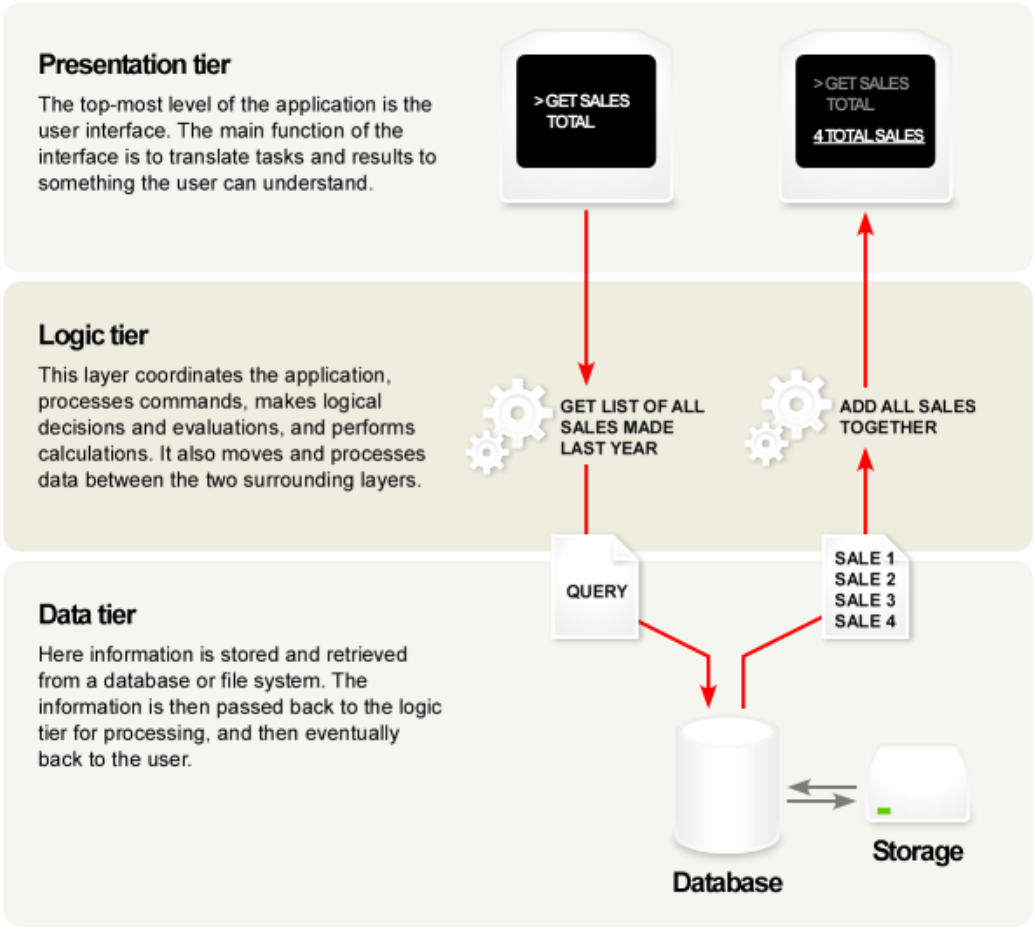
Kommunikation mellan klientens webbläsare och en webbserver fungerar på följande sätt. Servern väntar på inkommande anrop på port 80. Klienten begär information med hjälp av ett *request-meddelande*. När servern får in ett *request-meddelande* från en klient så skapar servern en anslutning där servern skickar ett *response-meddelande* som innehåller information om klientens begäran samt själva innehållet. När denna procedur är genomförd så stängs anslutningen ner och klienten och servern har ingen kontakt med varandra längre. Detta leder till att en webbserver inte har någon vetskap om klienter och i vilket tillstånd dessa klienter är. Detta medför att för att kunna hantera detta måste man använda tekniker som till exempel *cookies*[4]. Detta beroende av http gör webben till en i grunden statisk plattform där information skickas på ett synkront sätt mellan server och klient.

3.1.3 Design

En modern webbplats idag har ett gränssnitt i form av html som användarna laddar ner till sin dator. Denna html är många gånger genererad av olika serverkomponenter eftersom mycket information som hittas på moderna webbplatser i grunden kommer från en databas.

Gränssnittet antas vara byggt efter W3C:s riktlinjer för html och den specifika layouten är specificerad med CSS(Cascading Style Sheets)[5]. För att göra systemet så skalbart som möjligt så delar upp applikationen i logiska lager (se figur 1).

[4] En fil som sparas hos klienten för att kunna hantera sessioner och dylikt.
[5] se sidan 8 för förklaring av begreppet



Figur 2 Blockdiagram som visar en applikation uppdelad i logiska lager (6).

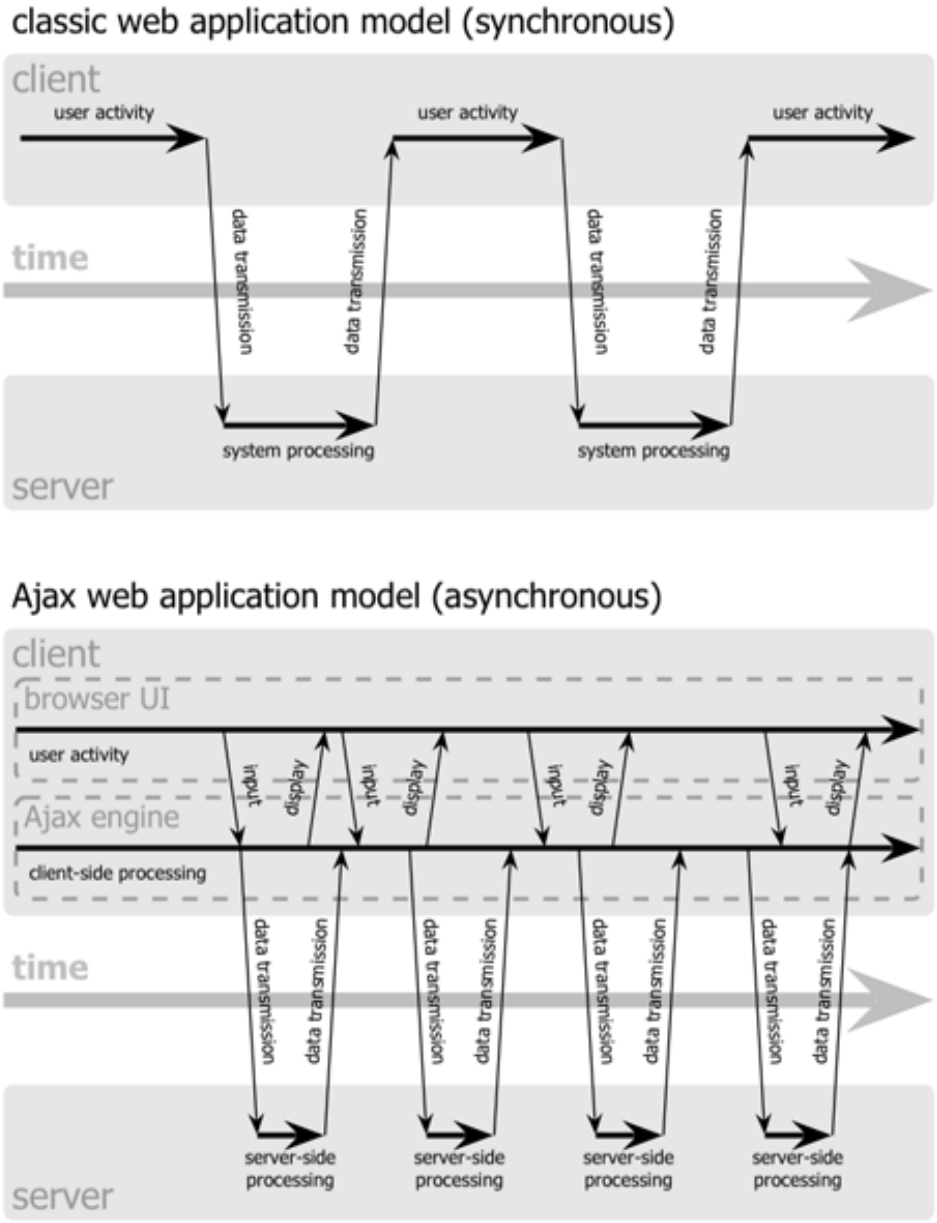
Det innebär att man har ett lager som hanterar gränssnittet mot användarna, ett andra lager som hanterar verksamhetslogik samt ett tredje lager som består av ett dataskikt där informationen ligger lagrad. När användarna skickar information till servern finns det möjlighet för webbapplikationen att agera på olika sätt baserat på innehållet i det aktuella *request-meddelandet*. Exempelvis visa speciell information för just den användaren, göra kontroller av indata eller bara hämta information från det underliggande datalagret. Det finns även möjlighet att påverka det nedladdade gränssnittet med hjälp av JavaScript. På så sätt kan man göra gränssnittet mer interaktivt och levande.

2.2 Asynkron webbkommunikation

Protokollet för kommunikation mellan en webbläsare och en webbserver är i första hand http(HyperTextTransfer Protocol)[6]. För varje nytt anrop från en klient så laddas information om i klientens webbläsare (se figur 2). Detta resonemang tar inte hänsyn till att moderna webbläsare *cachar* data hos klienten till exempel bilder som inte laddas ner igen utan läses direkt från användarens hårddisk. Detta är principen för synkron kommunikation. Konsekvenserna av att http fungerar på detta sätt blir att klienten måste ladda om en ny sida varje gång och blir tvingad att invänta svaret från servern, även om det bara är en liten del av informationen som uppdateras. Detta problem kan åtgärdas om man tillämpar asynkron kommunikation mellan klient och server. Det skulle innebära en

[6] se sidan 8 för förklaring av begreppet

miljö där klienten kan uppdatera delar av sitt gränssnitt och bara hämta den informationen som verkligen är uppdaterad. Detta resonemang avser asynkron kommunikation ur webbläsarens perspektiv. Kommunikationen över http är i grunden alltid synkron mellan en klient och en server.



Figur 3 Skillnaden på synkron jämfört med asynkron webbkommunikation (7)

3.2.1 Java

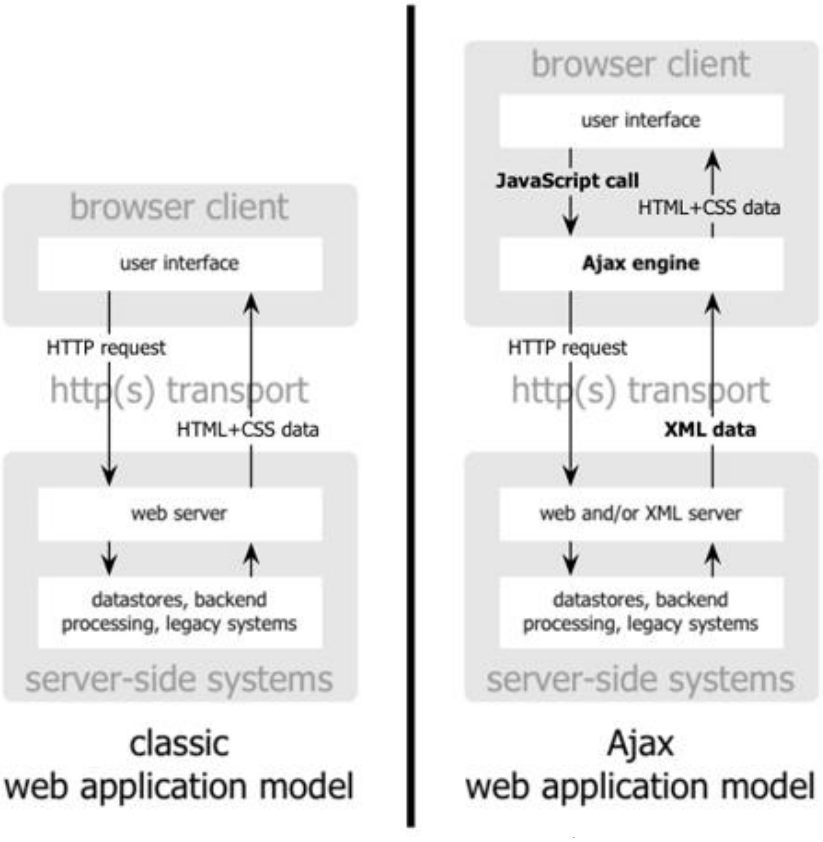
På grund av begränsningarna i http protokollet så har det vuxit fram olika tekniker för att kunna kommunicera mellan klient och server utan att behöva ladda om hela gränssnittet hos klienten. En sådan teknik är *Java-applets*. Det är en komponent som är skriven i Java och som exekveras inne i webbläsaren med hjälp av en tillägg. Eftersom en *applet* körs som en egen process så kan det hantera kommunikationen mellan klient och server på ett asynkront sätt utan att behöva vara begränsad av webbläsaren. Nackdelen med *applets* är att det bygger på att webbläsaren är utrustad med det nödvändiga tillägget i webbläsaren. Om detta inte finns installerat måste det först laddas ned.

3.2.2 Macromedia Flash

En annan teknik att undvika begränsningarna i webbläsaren är att använda en annan typ av tillägg, *Macromedia Flash*. Den har samma syfte som *Java-applets*, att förbättra gränssnittet och kommunikationen mellan klient och server. Det är dock samma problem som för *Java-applets*, *Flash* kräver också att ett tillägg i webbläsaren laddas ned och installeras.

3.2.3 AJAX

AJAX (se figur 3) är inte ett programmeringsspråk ett koncept som bygger på standardiserade webbtekniker såsom XML och JavaScript. Ett sätt att åstadkomma detta är att använda JavaScript objektet XmlHttpRequest (4) som finns inbyggt i alla moderna



Figur 4 Schematiskt bild över AJAX kommunikation (7)

webbläsare. Med hjälp av detta objekt kan man skicka ett *request-meddelande* på ett asynkront sätt, utan att göra en helt ny omladdning av sidan. XmlHttpRequest-objektet gör en GET eller POST i bakgrunden och levererar ett *response-meddelande* till klienten.

3.2.3.1 XmlHttpRequest

När Microsoft implementerade stöd för XmlHttpRequest-objektet (8) i Internet Explorer 5 så valde man att, till motsats från de övriga webbläsarna göra det med hjälp av att skapa en ActiveX objekt istället för en standard JavaScript. Implementationen gjordes som en del av deras webbklient till Microsoft Exchange Server, Outlook Web Access. På grund av skillnaderna i implementation så medför detta att man bör kontrollera vilken webbläsare som klienten använder innan man kan använda objektet. Microsoft har från och med Internet Explorer 7.0 valt att följa W3C rekommendationerna och implementerat XmlHttpRequest som ett standard JavaScript objekt. XmlHttpRequest-objektet är ett API som tillhandahåller metoder och egenskaper som en utvecklare kan använda sig av för att hantera asynkrona anrop mellan klient och server (se tabell 1). Objektet skapar en dedikerad kanal där utbyte av data genomförs. När man konstruerade objektet var XML det standardiserade sättet att skicka strukturer därav den specifika responseXml metoden.

Namn	Typ	Förklaring
open(metod, url, async)	metod	skapar en anslutning till en server som specificeras i url, vilken metod som kan användas(GET,POST e.t.c.), samt om anropet ska ske asynkront eller inte.
send(content)	metod	skickar ett http-request genom en befintlig anslutning
onreadystatechanged	event	ett event som avfyras varje gång tillståndet för objektet ändras.
responseText	egenskap	returnerar svaret i form av text
responseXml	egenskap	returnerar svaret i form av xml
readyState	egenskap	olika tillstånd för objektet. 0 = uninitialized 1 = open 2 = sent 3 = recieving 4 = loaded

Tabell 1 De viktigaste metoderna och egenskaperna för objektet XmlHttpRequest

```
if (window.XMLHttpRequest)
{
// IE7, Mozilla, Safari, Opera, etc.
xmlHttp = new XMLHttpRequest();
}
else if (window.ActiveXObject)
{
      // IE 5.x and 6
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}

xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("POST",url,true)
xmlHttp.send(null)

function stateChanged()
{
      if( xmlHttp.readyState == 4 && xmlHttp.status == 200 )
      alert( xmlHttp.responseText )
}
```

Exempel 3 Kodexempel på hur man använder XmlHttpRequest

Det allra enklaste fallet när man använder sig av XmlHttpRequest-objektet sker på följande sätt (se exempel 3). Först kontrolleras vilken webbläsare som klienten har och utifrån detta så skapar man en referens till XmlHttpRequest-objektet, antingen ActiveX[7] eller JavaScript. Sedan kopplar man en funktion till *onreadystatechange* händelsen som hanterar de olika tillstånd som objektet kan ha. Slutligen skapar man kopplingen mot servern och skickar det asynkrona anropet. Svaret från servern behandlas i funktionen *stateChanged*.

3.2.3.2 Frames och IFrame

Ett annat sätt att skapa asynkron kommunikation mellan klient och server över http är att använda sig av html elementen *Frame* eller *IFrame* (se exempel 4). Detta gör det möjligt att dela upp en sida i flera delar där varje del har en egen separat html sida som man sedan kan uppdatera med hjälp av JavaScript. Upplevelsen för användarna blir liknande den man åstadkommer med XmlHttpRequest objektet.

```
<script>
function GetData()
{
try
{
      var obj = document.getElementById('dataFromServer');
      obj.src = "IFrameDataGenerator.aspx";
}
catch(e){alert(e.message);}
}
```

[7] se sidan 8 för beskrivning av begreppet

```
</script>
<h1>Här kommer serverns svar att visas:</h1>
<iframe id="dataFromServer";"></iframe>
<input id="btn_FillIFrame" type="button" onclick="GetData()"/>
```

Exempel 4 Asynkron kommunikation med hjälp av IFrame

3.2.3.3 Script injection

Ännu ett sätt att använda sig av AJAX är att med JavaScript lägga in ett scriptobjekt med hjälp av att använda webbläsarens DOM[8] (se exempel 5). Inom detta scriptblock anropas sedan en sida som returnerar ett JavaScript som sedan kan exekveras hos klienten.

```
<script>
//---skapar ett script object dynamisk
script = document.createElement( 'script' );
document.body.appendChild( script );

//---laddar ner ett script när klienten trycker på en knapp
function GetData()
{
      try
      {
            script.src = 'DOMScriptInjectionDataGenerator.aspx';
      }
      catch(e){alert(e.message);}
}

//---fyller ett div element med information från servern
function SetData(data)
{
      try
      {
            var obj = document.getElementById( 'content' );
            obj.innerText = data;
      }
      catch(e){alert(e.message);}
}
</script>

<div id="content"></div>
<input id="btn_FillDiv" type="button" value="Get data from server"
onclick="GetData()"/>
```

Exempel 5 Asynkron kommunikation med hjälp av Script injection

[8] se sidan 8 för förklaring av begreppet

I detta exempel så exekveras ett JavaScript som genereras från servern via. Servern skapar en html sträng som innehåller ett anrop till funktionen SetData där attributet data är genererat från servern. Svaret från servern ser ut på följande sätt.

```
SetData('Här kommer ett svar från servern');
```

Exempel 6 Exempel på ett genererat JavaScript från servern

Textsträngen är genererad på servern och för enkelhetens skull så är det i det här fallet endast en enkel sträng. Det kunde lika gärna vara en XML struktur som man sedan tolkar på klientsidan. Eftersom detta läggs i ett scriptblock som finns hos klienten så exekveras det i samma stund som det genereras från servern, utan omladdning av sidan.

4 Metod

För att kunna besvara problemställningen som denna rapport bygger på krävs en djupare inblick i själva systemet som ska förbättras med hjälp av asynkron webbkommunikation. För att göra en skalbar och robust implementation i ett redan befintligt system så måste man förstå bakgrunden till systemet, vilka funktioner som är viktiga och hur systemet är uppbyggt. Nedan beskrivs arbetsgången för detta examensarbete.

1. Fördjupningar i WIS, Skyddat Webbaserat Informationssystem.
2. Fördjupningar i teknikerna bakom asynkron webbkommunikation för att kunna ta upp fördelar respektive nackdelar med denna teknik.
3. Fördjupningar i teknikerna runt AJAX och olika tekniker för att implementera detta i ett befintligt system som bygger på Microsoft .NET Framework 1.1
4. Identifiera ett antal användarfall där man skulle kunna förbättra gränssnittet med hjälp av AJAX.
5. Fördjupningar i Microsofts ramverk för AJAX, ASP.NET AJAX, för att kunna skapa en så robust och skalbar implementation av asynkron webbkommunikation i nästa version av WIS som är byggt på Microsoft .NET Framework 2.0.
6. Identifiera ett antal användarfall där man skulle kunna förbättra gränssnittet med hjälp av AJAX.
7. Tillverka prototyper utifrån dessa användarfall.

WIS applikationen transporterar mängder med information fram och tillbaka mellan klient och server. Ett grundläggande mål för alla webbapplikationer är att minimera mängden data som skickas för att få så snabb responstid som möjligt mellan klient och server. För att kunna avgöra vilka funktioner som skulle kunna förbättras har jag undersökt hur WIS applikationen hanterar utbytet av information mellan klient och server. WIS 1.6 är byggd som en klassisk webbapplikation och då sker utbytet mellan klient och server synkront. Den framtida versionen av WIS kommer till stor del att använda sig av synkron kommunikation men eftersom den versionen är under kravfångst så finns det stora möjligheter att få med ett asynkront kommunikationssätt redan innan prototypprocessen startar under hösten 2007. När funktioner som skulle kunna förbättras har identifierats så börjar arbetet med att tillverka några enkla prototyper som beskriver på vilket sätt gränssnittet har förbättrats för användarna. Under realiseringen av prototyperna skall även designmönster som min handledare har föreslagit undersökas hur man använder dessa i asynkron kommunikation.

4.1 Systemfördjupningar

WIS 1.6 är baserat på Microsoft.NET Framework 1.1 och följer ett arkitektiskt design koncept kallat Domain Driven Design som har en stark koppling till objektorienterad utveckling. Detta är av mindre betydelse för mitt examensarbete eftersom den asynkrona

kommunikationen sker i det yttersta lagret av applikationen men för att få bättre förståelse för hur applikationen är uppbyggd har jag även orienterat mig inom detta område. Det som är av större intresse är vilket designmönster som man ska använda vid implementationen av AJAX.

För att skicka information mellan klient och server använder sig WIS 1.6 både av http GET med data från klienten i själva url:en samt av POST med data från ett html-formulär. Det är väldigt vanligt att man blandar dessa metoder beroende på vad det är för typ av data man hanterar. Sedan så har man i väldigt stor uträckning använt sig av Microsofts UserControls teknik som även själva ASP.NET använder sig av. Där kan man isolera design och logik i en separat komponent som sedan kan återanvändas på flera platser. Detta leder till en modulbaserad design där en webbsida består av en bas i form av ett *WebForm* som i sin tur innehåller en mängd *Controls*. En *UserControl* är en egen klass som man själv har skapat. Den kan i sin tur innehålla *Controls* som till exempel textrutor för input från användare. Många av kontrollerna laddas dynamiskt baserat på en mängd olika parametrar. På de flesta ställen inom presentationslagret så laddas all html som ska presenteras till klienten ner på en gång. Om sedan informationen ska ändras så laddas hela gränssnittet om, inklusive de dynamiskt laddade kontrollerna.

Förutom att undersöka hur man implementerar stöd för asynkron kommunikation i ett befintligt system så har även examensarbetet omfattat att ta reda på vilken ny funktionalitet i en framtida version av WIS som skulle kunna. Här är man inte lika bunden vid en redan fastställd arkitektur utan kan tänka mer fritt. Eftersom kravfångsten till denna version kommer att pågå under våren 2007 så kommer denna del examensarbetet endast bidra med konceptuella förslag på ny funktionalitet. Samarbete med IT-arkitekter för den nya versionen av WIS ligger som grund för den nya funktionaliteten.

4.2 Vetenskapliga fördjupningar

För att kunna implementera AJAX så måste man fördjupa sig i teknikerna bakom AJAX som är JavaScript och i viss mån XML. De nya versionerna av de vanligaste webbläsarna har implementerat stöd för standard JavaScript XmlHttpRequest –objektet, för den tidigare versionen av Internet Explorer måste man använda sig av ActiveX-objektet istället.

4.2.1 AJAX

För att göra asynkrona anrop över http använder man sig till exempel av XmlHttpRequest-objektet som skickar data asynkront från en klient till en server och tar emot svaret i form av en textsträng eller som en XML-struktur. När svaret som returneras från servern innehåller data som har ett förhållande till varandra till exempel att klienten frågar efter information om en viss händelse och svaret är i form av ett meddelande med en rubrik samt en brödtext kan det vara lämpligt att returnera detta som XML som sedan

tolkas med hjälp av JavaScript hos klienten. För att underlätta utvecklingsarbetet under examensarbetet så har jag undersökt en mängd olika JavaScript bibliotek som finns tillgängliga på Internet. De flesta bibliotek är väldigt mycket mer än bara ett AJAX-bibliotek utan är mer ett ramverk för JavaScript utveckling i allmänhet. Om man är ute efter att göra mer än bara den asynkrona kommunikationen så kan det vara intressant att titta på dessa men i mitt fall så är det främst själva kommunikationen mellan klient och server som är intressant. Därför har jag fokuserat på de ramverk som hanterar enbart AJAX-funktionalitet och använt dessa som referens när jag har fördjupat mig vidare i JavaScript delarna.

4.2.3 JavaScript

En webbsida som skickar data från en klient till en server använder sig av synkrona http anrop som standard. För att kunna påverka på vilket sätt man skickar data så måste man använda sig av logik som klienten har laddat ner i form av JavaScript. Det finns ingen möjlighet för en server att påverka en klient på grund av den tillståndslösa egenskapen som http innehar. Med hjälp av JavaScript kan man se till att kommunikationen som sker mellan klient och server sker på ett asynkront sätt samt även påverka användarens gränssnitt utan att det sker något nytt http anrop till servern. Ett stort problem med JavaScript är hur olika det har implementerats i olika webbläsare vilket leder till att mycket tid går åt till att lösa problem som inte i sig har med det verkliga problemet att göra utan bara skillnader mellan olika webbläsare. Ett klassiskt exempel är hur man kommer åt specifika element i en webbsida via DOM. Det har genom årens lopp implementerats olika vilket innebär att samma sida uppför olika eller kanske inte alls fungerar mellan olika webbläsare.

4.2.3 ActiveX

På grund av att de tidigare versionerna av Internet Explorer använde sig av ActiveX objekt för att skapa referenser till XmlHttpRequest objektet så undersökte jag vad ActiveX är för något. Till skillnad från många ActiveX komponenter ute på Internet så är det inga säkerhetsproblem med denna komponent. Den är inbyggd i webbläsaren Internet Explorer och behöver därför inte laddas ner till klienten. En vanlig missuppfattning är att ActiveX komponenter är att de laddas ner och alltid medför säkerhetsbrister. När det gäller just detta ActiveX objekt så är det bara ett sätt som Microsoft valde att implementera asynkron kommunikation för Internet Explorer 5.5+. Den senaste versionen av Internet Explorer har dock stöd för bägge objekten.

4.2.4 Xml

Html är inte speciellt bra lämpat för att definiera data. Det är främst tänkt att agera som ett sätt att presentera ett innehåll. Ett sätt att separera data från presentationen är att använda sig av XML som är en hierarkisk representation av data. Man kan sedan generera html på olika sätt från Xml för att publicera innehållet på till exempel Internet. Andra användningsområden för Xml är för att skicka data mellan olika applikationer på ett strukturerat sätt. Som exempel kan nämnas att Xml används för transport av data i SOAP, applikationsprotokollet för WebServices.

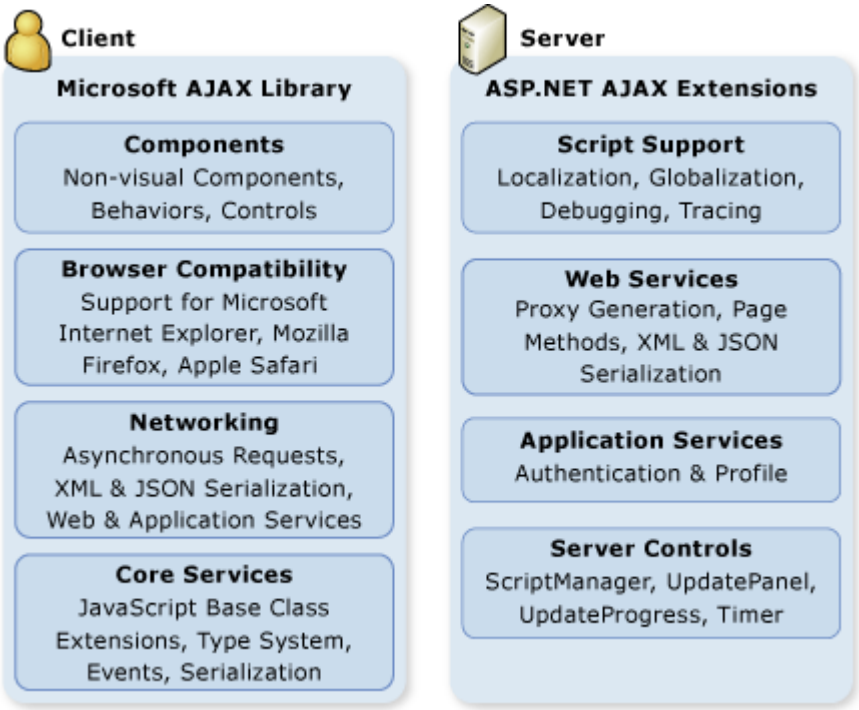
4.2.5 .NET Framework

WIS 1.6 är baserat på .NET Framework 1.1 som är ett ramverk som används för att bygga applikationer. Det har stora likheter med JVM(Java Virtual Machine) konceptet eftersom man inte kompilerar maskinkod från sina källkodsfiler utan det blir ett så kallat

IML(InterMediate Language). Detta IML kompileras sedan i ramverket till maskinkod som sedan körs på respektive plattform. ASP.NET bygger på detta ramverk och har till motsats från den tidigare ASP tekniken som inte körde kompilerad kod, bättre prestanda och stöd för äkta objektorienterade språk som till exempel C#.

4.2.6 ASP.NET AJAX

ASP.NET AJAX är en implementation av AJAX för ASP.NET 2.0. Detta innebär att ASP.NET 1.1(för .NET Framework 1.1) inte kan ta del av detta bibliotek. Ramverket runt ASP.NET AJAX innehåller både en klientdel och en serverdel (se figur 4) med flera olika block som tar hand om många grundläggande problem som ofta förknippas med JavaScript programmering, såsom problem med olika implementationer av DOM i olika webbläsare.



Figur 5 Blockdiagram över arkitekturen i ASP.NET AJAX (11)

4.2.6.1 ASP.NET AJAX Library

Klientlagret av ASP.NET AJAX kan användas separat för att ge utvecklarna en möjlighet att använda JavaScript på ett objektorienterat sätt. Man använder sig av funktionalitet i JavaScript som gör det möjligt att använda sig av klassiska objektorienteringsmönster såsom *arv*, *interface*, *namespace* och *enumerationer*. JavaScript saknar inte stöd för objektorientering utan det finns objekt och ett primitivt sätt att skapa klasser men ASP.NET AJAX kapslar in den funktionaliteten och skapar ett mer familjärt sätt jobba med objekt. Ramverket bygger också ut funktionalitet på redan befintliga objekt i JavaScript.

Som exempel kan nämnas String-objektet som har utökats med formaterings funktionalitet som påminner om det som finns i C# och Java. Även Array-objektet har utökats med riktiga metoder för att rensa, lägga till och ta bort element. En annan viktig funktion i ASP.NET är att tillhandahålla ett konsekvent sätt att komma åt och påverka gränssnittet hos användaren med hjälp av webbläsarens DOM. På grund av skillnader i implementation av DOM i olika webbläsare så får ofta webbutvecklare brottas med en massa problem att en webbsida inte uppför sig på samma sätt i olika webbläsare. Detta får man hjälp med i ASP.NET AJAX tack vare ett API som hanterar alla DOM funktioner och ger utvecklaren ett konsekvent sätt att komma åt specifika element. Samma sak gäller för händelsehantering i JavaScript. Olika webbläsare har implementerat hur händelser hanteras på olika sätt vilket man får hjälp med genom ramverket.

4.2.6.2 ASP.NET AJAX Extensions

Förutom ett rikt bibliotek för att skriva script på klientsidan så innehåller även ASP.NET AJAX ett API för utvecklare som hellre skriver logiken för asynkrona anrop på serversidan. AJAX är i grunden något som körs i klientens webbläsare, det är där alla http-anrop körs mot servern. Det som ASP.NET AJAX Server hjälper till med är ett sätt att definiera klientkod på serversidan som sedan genereras till klienten. Med hjälp av detta kan man alltså skriva AJAX webbapplikationer utan att egentligen behöva skriva någon kod för klientsidan.

4.2.6.3 ScriptManager och ScriptManagerProxy

Den absolut viktigaste delen i ASP.NET AJAX är server komponenten ScriptManager. Det är den som hanterar ramverket runt JavaScript programmeringen och gör det möjligt för utvecklarna att lägga mindre energi på skillnader mellan olika webbläsare och istället koncentrera sig på att utveckla själva funktionaliteten på klienten. Den ser till att ladda ner rätt JavaScript filer till klienten beroende på vilken webbläsare som används. Den hanterar även alla andra JavaScript resurser man skriver. Det kan bara finnas en ScriptManager på varje sida, om man skulle behöva använda sig av flera så finns det en ScriptManagerProxy klass man kan använda sig av. Skillnaden mellan dessa är i stora drag att man hanterar alla generella script i sin ScriptManager och mer specifika script, som kanske inte finns på alla sidor, med en ScriptManagerProxy. Det enda som behövs för att använda sig att ASP.NET AJAX Klient är att lägga till en ScriptManager på sin ASP.NET sida. Detta gör så att när en klient anropar denna sida så ser denna kontroll till att ladda ner rätt JavaScript beroende på vilken webbläsare som används. När sedan dessa filer finns tillgängliga på klienten så är det fritt fram att använda alla API:er som finns tillgängliga i ASP.NET AJAX. Detta är egentligen det enda som behövs för att kunna utveckla AJAX funktionalitet men det är viktigt att påpeka att man inte får något mer än JavaScript biblioteken, man måste fortfarande själv se till att göra sina asynkrona anrop och uppdatera klientens gränssnitt via DOM men man gör det via ASP.NET AJAX vilket förenklar programmeringen avsevärt.

4.2.6.4 UpdatePanel

Tidigare nämnde jag om att det enda som behövs för att utveckla AJAX funktionalitet är att använda sig att ScriptManager klassen på serversidan. Detta innebär dock att man har väldigt mycket programmering på klientsidan framför sig. All logik som hanterar uppdateringen av klientens webbläsare måste skrivas manuellt, visserligen så har man en väldigt stor fördel av alla de API:er som finns tillgängliga i ASP.NET AJAX Library. Ett annat sätt att arbeta är att flytta all logik till serversidan där man har bättre stöd för programmering i och med att man har hela .NET Framework att arbeta mot istället för webbläsarens DOM. För att använda sig av denna metod så utnyttjar man klassen UpdatePanel. Med hjälp av denna kan man definiera områden på sina sidor där man vill utnyttja asynkron kommunikation.

4.3 Verktyg

För att kunna utvärdera och konstruera prototyper för den nya asynkrona funktionaliteten så har jag satt mig in i samma utvecklingsmiljöer som WIS projektet har använt. Dessa miljöer är Microsofts officiella utvecklingsmiljöer för .NET Framework.

4.3.1 Visual Studio .NET 2003

Utveckling för .NET Framework 1.1, som WIS 1.6 är baserat på, så använde jag mig av Visual Studio .NET 2003. Det är en avancerad IDE där man arbetar i solutions som i sin tur kan innehålla en mängd olika projekttyper. WIS 1.6 är en solution som innehåller en mängd olika klassbibliotek och ett gränssnitt mot användarna i form av ett webbgränsnitt byggt med ASP.NET.

4.3.2 Visual Studio .NET 2005

WIS 2.0 kommer till största sannolikhet att byggas på .NET Framework 2.0 och då kan man inte längre använda sig av den äldre versionen av Visual Studio 2003 utan måste gå över till den nya utvecklingsmiljön Visual Studio .NET 2005. De båda miljöerna kan utan problem köras samtidigt. Skillnaderna är inte så stora utan har man lärt sig Visual Studio 2003 så kan man med lätthet lära sig det nya systemet och ta del av förbättringarna som definitivt finns. För att kunna bygga prototyper som bygger på Atlas så måste man använda sig av denna miljö eftersom Atlas kräver .NET Framework 2.0.

4.3.3 ASP.NET AJAX Toolkit

De konceptuella prototyperna som jag planerar att göra till WIS 2.0 kommer att använda sig av ASP.NET AJAX och i och med det så kommer jag att sätta mig in i ett fristående *open source* bibliotek som hanterar ett flertal grafiska effekter som man kan använda sig av för att göra upplevelsen för användarna bättre. Biblioteket ämnar till att i första hand förenkla hanteringen av JavaScript. Hur mycket av detta som kommer att realiseras i prototyperna får jag och min handledare komma fram till allt eftersom arbetet fortgår. Det vikiga är att få en överblick i hur man kan använda sig av detta bibliotek och mer specifikt hur AJAX funktionalitet hanteras.

4.4 Prototyper

Under examensarbetets gång har följande funktioner som är intressanta för WIS funnits.

1. Listor
2. Dra-och-släpp
3. Notifiering
4. Moduler
5. Automatisk-spara-funktion

4.5 Val av funktioner

Utifrån ovanstående fem funktioner valdes tre ut som ska vara grunden till prototyperna. Funktionen som avser punkt ett ”Listor” noterades tidigt som ett prioriterat område på grund av det omfattande användandet av olika typer av listor i applikationen. Dra-och-släpp valdes främst för att undersöka hur man skulle effektivisera tillvägagångssättet vid manipulering av data i applikationen. Till sist så valdes funktionen som hanterar notifieringar av olika slag. Främsta orsaken till detta var att utröna om det ens skulle vara möjligt med synkron kommunikation. Funktionen som kallas moduler i listan ovan var en idé som handlar om att göra en startsida som användarna själva kan välja vilka moduler som ska vara synliga. Detta föll bort på grund av att det inte direkt fanns några sådana önskemål från användarna. En annan funktion som blev bortgallrad var den automatiska spara-funktionen. Användarna skriver sina anteckningar i ett webbgränssnitt och där fanns det önskemål om en sådan funktion. Denna funktion hade varit väldigt intressant att undersöka men tyvärr så föll den bort på grund att det inte fanns utrymme i detta examensarbete att genomföra det.

4.4.1 Listor

En stor del av informationen visas i form av listor av olika slag och en del av dessa listor har efter önskemål från användarna utrustats med en ganska omfattande informationstext som syns medan användaren håller musen över en specifik rad i listan. Denna informationstext visas och döljs med hjälp av JavaScript vilket innebär att informationen som skall visas respektive döljas redan måste finnas i klientens webbläsare. Eftersom användarna har utryckt att de tycker denna funktionalitet hos listor är väldigt bra så har det efterfrågats mer och mer innehåll i denna informationstext. Detta leder till att när det börjar handla om relativt stora listor så måste klienten ladda ner mycket information som dom kanske aldrig utnyttjar. Detta är ju ett problem eftersom man eftersträvar att skicka så lite information som möjligt mellan klient och server. Helst av allt vill man bara skicka exakt det som klienten frågar efter som i det här fallet är bara den aktuella informationstexten för den specifika raden. Det är här som AJAX skulle kunna vara en

lösning på detta problem. En av funktionerna som dök upp under detta examensarbete var hur man skulle kunna minimera informationen som laddas in vid klientanrop av listor.

En annan funktion som gäller listor är hur man hanterar bläddring i listor. Stora listor delas ofta upp i delar som man sedan kan bläddra mellan. För att minimera mängden data som skickas mellan klient och server så genereras listan på servern och sedan skickar man de aktuella raderna till klientens webbläsare. Detta innebär att för varje gång en användare vill växla mellan olika sidor i en lista så laddas hela sidan om och de nya raderna visas. Detta kan uppfattas som statsikt och svårarbetat, speciellt under hög belastning. Här kan man kunna förbättra upplevelsen med hjälp av AJAX. Istället för att göra en omladdning av sidan varje gång servern genererar nästa del av listan så använder man sig av asynkrona anrop som sker i bakgrunden.

4.4.1.1 Krav

När en lista med objekt visas och den totala mängden sträcker sig över flera sidor och användaren väljer en ny sida ska bara den del av sidan som visar objekten uppdateras.

4.4.1.2 Design

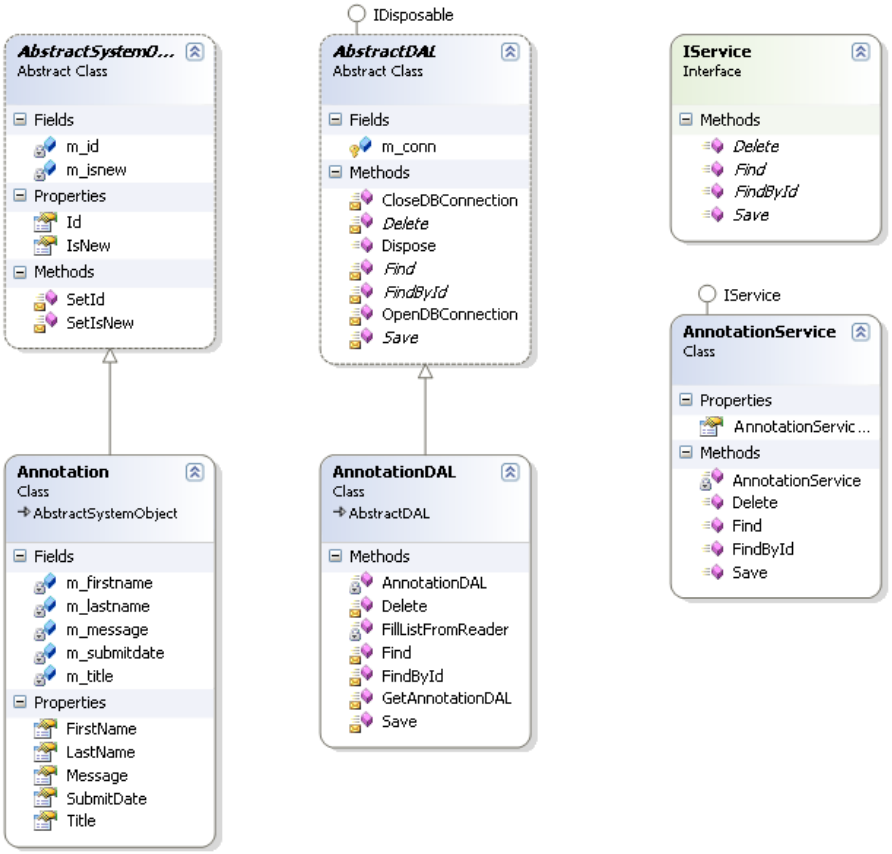
Designen av prototypen har följt rekommendationer från Ajax Design Patterns (11).

4.4.1.3 Predictive Fetch Pattern

Detta design mönster innebär att man försöker minska antalet anrop mellan klient och server med hjälp av att försöka anta vad klienten vill göra i nästa steg. När det gäller att försöka implementera detta mönster generellt över en hel applikation som visar det sig ganska snabbt att det kommer att bli helt omöjligt att försöka förutse användarnas interaktions mönster. Om man däremot tittar på en isolerad del av en applikation, som till exempel, listor av olika slag där innehållet är genererat från en server så blir det mer applicerbart. Exempelvis om man har en lista på 100 poster som man sedan delar upp i tio delar med tio poster varje. Dessa delar kan man sedan bläddra mellan via ett navigeringssystem. Om en användare står på sida ett så är det ganska troligt att nästa sida han vill titta på är sida två. Där kommer Predictive Fetch Pattern in i bilden. Exempelvis kan man ladda sidan efter och sidan före den aktuella sidan som användaren tittar på. Detta skulle innebära att man laddar nästa steg ögonblickligen, utan ett nytt anrop görs till servern. Detta leder till färre anrop men till kostnaden av ett behov av att *cacha* data hos klienten.

4.4.1.3 Implementation

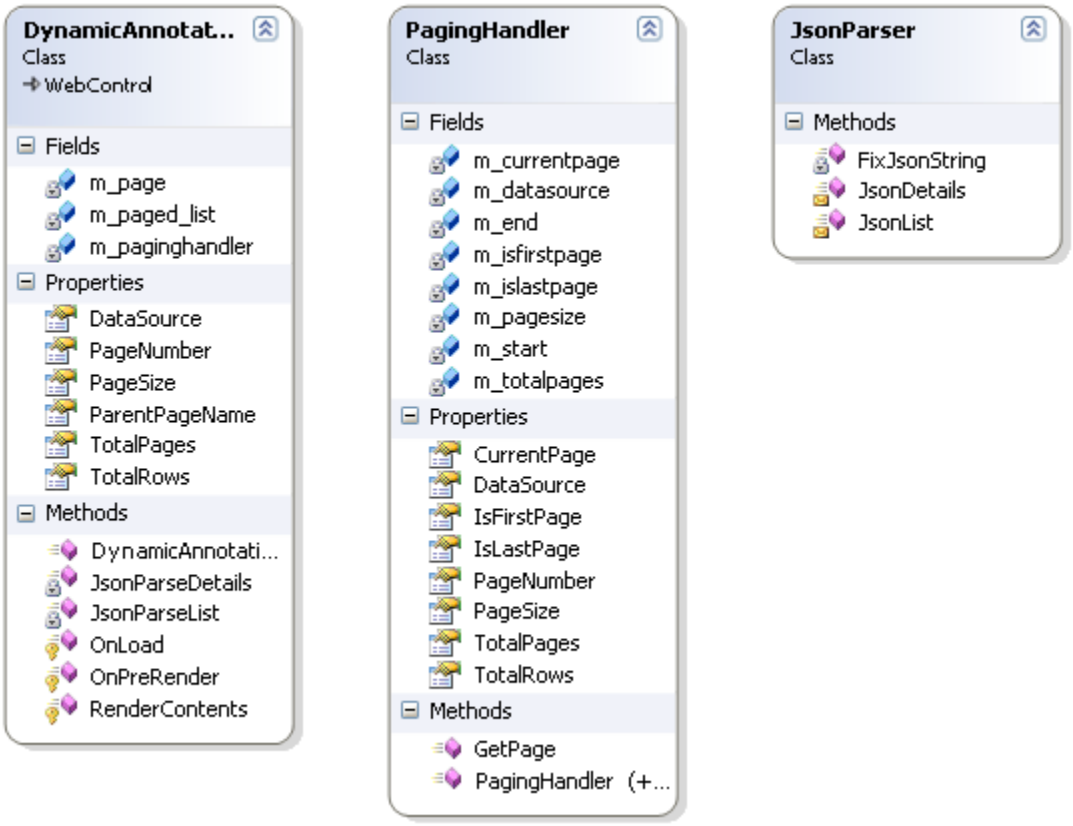
Listorna hanterar anteckningar som tänkta användare har lagt in i systemet. Dessa listor används flitigt på massor av ställen och det är inte alltid listorna innehåller anteckningar men denna prototyp har tagit anteckningar som ett exempel. En enkel objektmodell (se figur 5) har skapats där en anteckning representeras av klassen Annotation. Själva hämtningen av anteckningar sker med hjälp av klassen AnnotationService som är en publik klass som hanterar anteckningar. Den interna klassen AnnotationDAL är inte tillgänglig utifrån för att försöka efterlikna ett verkligt scenario. För att komma åt anteckningarna i databasen måste man gå via den publika klassen AnnotationService. AnnotationService returnerar en generisk lista innehållande Annotationobjekt.



Figur 6 Klassdiagram över datalagret och objektmodellen

För att sedan visa denna lista av anteckningar som man fått från datalagret så har jag använt mig av en kontroll (se figur 6). En kontroll kan ses som en egen modul som sedan en utvecklare sedan bara kan använda på sin sida utan att behöva påverka något annat. Kontrollen använder sig av klassen PagingHandler som hanterar listans uppdelning i olika

sidor samt av en JsonParser klass som serialiserar listan med anteckningar från databasen till en användbart format för JavaScript. Anledningen till detta är att om man bortser från första gången kontrollen laddas då man får listans html från servern så är det i fortsättningen upp till klienten, det vill säga webbläsaren, att rendera ut nästa sida i listan. Allt detta på grund att kommunikationen mellan klient och server sker bakom kulisserna med hjälp av Ajax. För att kunna göra asynkrona anrop mellan klient och server så innehåller kontrollen att JavaScript som ligger automatiskt laddas ner till klienten när kontrollen anropas. Detta JavaScript innehåller metoder för att skapa en referens till XmlHttpRequest objektet samt metoder för att rendera html från json. Av utrymmesskäl så tar jag inte med denna del i rapporten.



Figur 7 Klassdiagram över Listkontrollen

Allt detta tillsammans leder till ett enkelt sätt att använda sig av en lista över anteckningar var som helst i en applikation. För den slutliga webbutvecklaren som vill använda sig av en sådan lista någonstans så ser det ut på följande sätt på gränssnittslagret (se exempel 7).

```
<p class="header">1. Listor - Asynkron kommunikation</p>

<div id="list">
<cc1:DynamicAnnotationList
      ID="DynamicAnnotationList1" runat="server" />
</div>
```

Exempel 7 ASP.NET implementation av lista

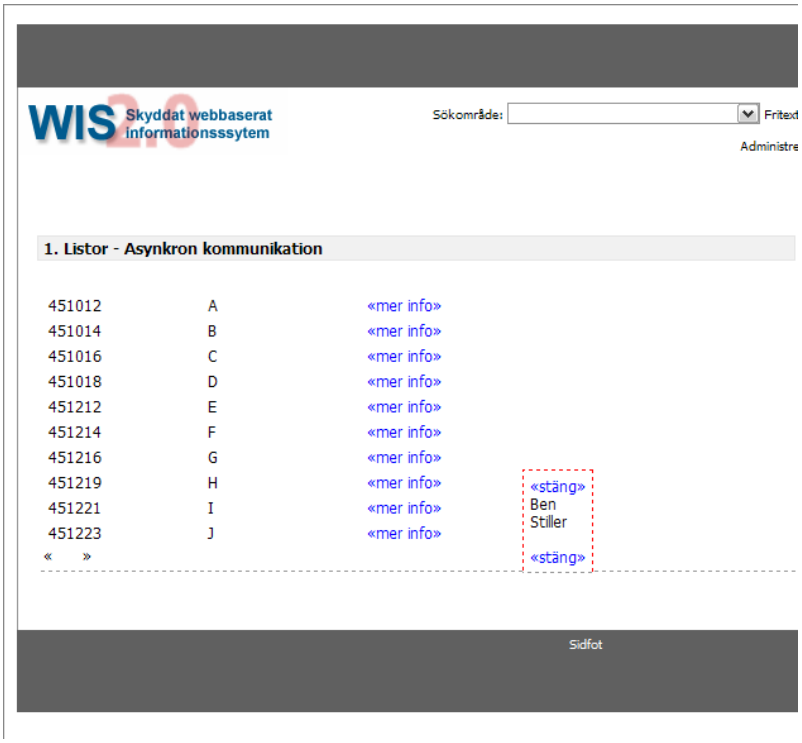
Förutom denna kod så måste man koppla denna kontroll till en datakälla innehållande anteckningar. Detta gör man med hjälp av några egenskaper på kontrollen (se exempel 8) .

```
AnnotationService serv = AnnotationService.AnnotationServiceManager;

DynamicAnnotationList1.DataSource = serv.Find();
DynamicAnnotationList1.PageNumber = 1;
DynamicAnnotationList1.PageSize = 10;
DynamicAnnotationList1.ParentPageName = "AsyncListor.aspx";
```

Exempel 8 ASP.NET listans datakälla

Det slutliga resultatet kan se ut på följande sätt när man tittar på listan i en webbläsare (se figur 7).



Figur 8 Exempel på hur listkontrollen renderas i en webbläsare

4.4.2 Dra-och-släpp

Applikationens hjärta är anteckningarna som användarna skriver och lämnar in till redaktören för publicering. Redaktören har en inkorg där denne har överblick över alla inkomna anteckningar som användarna vill publicera. Från inkorgen bestäms om anteckningen ska publiceras eller nekas publicering. All denna logik hanteras på ett klassiskt sätt genom att klicka på knappar för att komma till olika vyer där man antingen kan skriva in sin anteckning eller titta på listor över anteckningar. Denna hantering innebär en hel del klickande och nya vyer som laddas in. Här skulle man med hjälp av AJAX kunna göra ett mer flexibelt gränssnitt som bygger på dra-och-släpp tekniken. Användarna är vana att arbeta på det sättet från sitt operativsystem som bygger på att använda musen till just dra och släpp.

4.4.2.1 Krav

Funktionaliteten måste efterlikna den dra-och-släpp teknik som användarna är vana vid från traditionella datorgränssnitt.

4.4.2.2 Design

I denna prototyp har jag använt ett redan existerade JavaScript bibliotek för att åstadkomma själva logiken kring dra-och-släpp. Detta bibliotek är ett gratis bibliotek som kan användas fritt av vem som helst. Biblioteket heter script.aculo (12) och bygger på ett annat JavaScript ramverk kallat *Prototype* (13) som för övrigt har stora likheter med Microsofts klientdel i ASP.NET AJAX.

4.4.2.3 Implementation

Denna prototyp använder sig inte av kontroller utan enbart av script.acula biblioteket samt av mina egna klasser (se figur 5) för att hämta data från en databas. Implementationen är gjord på så sätt att jag hämtar en lista av anteckningar från databasen och med hjälp av JavaScript definierar jag dessa som rader i en tabell på vilka man kan utöva dra-och-släpp teknik. På sidan finns sedan en papperskorg för att simulera ett scenario där användare drar objekt från en lista av anteckningar till en papperskorg, liknande det man kan göra i de flesta grafiska operativsystem. Med hjälp av script.aculo så specificerar man vilka objekt man vill kunna dra och släppa. Sedan när man släpper något i papperskorgen så anropar man servern med hjälp av XmlHttpRequest objektet och markerar anteckningen som borttagen.

4.4.3 Notifiering

WIS är en applikation som hanterar mängder av information som kommer från massor med olika användare och organisationer. Informationen är indelad i dagböcker som sedan kan delas ut så att andra kan ta del av den information som man skriver. Om man prenumererar på en viss dagbok så vill man få reda på när det har hänt något i den aktuella dagboken, det vill säga när en anteckning har blivit uppdaterad eller om det har tillkommit någon ny anteckning. WIS 1.6 kontrollerar detta idag men för att en användare ska kunna bli notifierad om att det har hänt någonting i en dagbok så krävs det att användaren själv aktivt uppdaterar sin webbläsare. Då körs kontrollen över senaste uppdaterade dagböcker. Detta är på inget sätt optimalt eftersom det hänger på användarens uppdateringsfrekvens. Man skulle kunna skapa en timer som tvingar klientens webbläsare att uppdatera sidan och men det finns risk att applikationen uppfattas som att den tar kontrollen över användaren, och det är inte en bra lösning. Här finns en möjlighet för AJAX att förbättra upplevelsen för användaren. Denna prototyp kommer att använda sig av en timer som uppdaterar en del av sidan med hjälp av asynkrona anrop. Om någonting har hänt i en dagbok så visar man detta för användaren på ett lämpligt sätt.

4.4.3.1 Krav

Prestanda får inte bli lidande på grund av för frekventa uppdateringar.

4.4.3.2 Design

Denna är uppbyggd som en kontroll som man kan använda på olika ställen i sin webbapplikation. Uppdateringsfrekvensen för notifieringarna är en egenskap på kontrollen och beror även på hur många inloggade användare det är i systemet. Desto mer inloggade användare desto färre uppdateringar. Detta för att minska serverbelastningen vid många inloggade användare.

4.4.3.3 Implementation

Denna prototyp använder sig återigen av kontroller där man kapslar in logiken i en separat komponent som man sedan använder sig av på de ställen där man vill ha tillgång till notifiering av vad som har hänt i olika dagboksflikar.

4.5 Krav

Samtliga prototyper ska fungera i IE5.5+ samt Mozilla Firefox 2.0. Dessa krav ska ses som minimikrav för att prototyperna ska kunna utvärderas på ett tillfredställande sätt.

4.6 Implementering

4.6.1 WIS 1.6

Under examensarbetets gång så valde vi att inte implementera något i produktionsversionen av WIS 1.6. Detta på grund av att kunden fryste projektet i december 2006 vilket ledde till att implementationen i WIS 1.6 är endast gjord i min egen utvecklingsmiljö. Detta innebär att fokuseringen vid implementation har gällt den framtida versionen av WIS.

4.6.2 WIS 2.0

Prototyperna är gjorda i Visual Studio 2005 inriktat på att kunna användas i WIS 2.0. All implementering skall vara modulbaserad och lätt kunna användas fristående på något annat ställe i applikationen. För att en prototyp skall anses som användbar måste det finnas en märkbar förbättring av gränssnittet för användaren utan att på något sätt ha förändrat förutsättningarna för andra delar av applikationen.

5 Slutsats

5.1 Problem

I första hand så har jag stött på problem när det gäller att implementera egna JavaScript lösningar som hanterar dynamiska uppdateringar av webbläsaren hos klienten. Inkonsekvent implementering av DOM beroende på webbläsare gör arbetet väldigt tidskrävande och frustrerande. Framtidens implementeringar av JavaScript kommer med största sannolikhet att byggas på olika typer av ramverk som hanterar skillnader mellan olika webbläsare i generella moduler. Rapporten tar upp Microsofts implementation i ASP.NET AJAX men det finns flera andra stabila ramverk. Till exempel så använder jag script.aculo i prototypen för dra-och-släpp. Jag utnyttjar bara själva logiken kring hur man drar och släpper olika element på en sida men ramverket, tillsammans med Prototype (13), har en mängd funktioner som hanterar det mesta men behöver för att göra dynamiska webbapplikationer, inklusive Ajax funktionalitet. Den stora fördelen med att använda ASP.NET AJAX är att man kan arbeta som ett konventionellt webbprojekt utan att behöva lägga ner en massa tid på att anpassa till asynkron kommunikation. Det förenklar utvecklingen och projektet kan använda sig av vanliga serverkomponenter. Ytterligare problem jag stötte på under utvärderingen av prototyperna var hur man skulle kunna bedöma om det blev någon förbättring för användarna. Speciellt prototypern dra-och-släpp kanske inte var den bästa funktionen att undersöka om man har perspektivet att minska datamängden mellan klient och server. Den funktionen påverkar i första hand själva interaktionen mellan användaren och applikationen och en sådan undersökning är alldeles för omfattande för att ta med i detta examensarbete. För att verkligen kunna utvärdera om asynkron kommunikation är applicerbart så bör man lägga ner mer tid på att värdera vilka funktioner som ska undersökas.

5.2 Resultat

Resultatet av prototyperna är väldigt blandat. Prototypen för Dra-och-släpp påverkade inte mängden data som skickas utan gav mest ett annorlunda interaktionsmönster för användarna. Detta är inte är helt lätt att utvärdera inom ramarna för detta examensarbete. Resultatet som man kan diskutera angående Dra-och-släpp är att datamängden som skickas är mindre till kostnad av ett ökat antal serveranrop. Den toala mängde data som skickas är i stort sett densamma. Prototypen som hanterar notifieringar blir också svårt att jämföra med synkron kommunikation. En notifiering gjord på ett synkront sätt skulle helt enkelt inte ha varit användbar. Att uppdatera hela webbläsaren vid jämna intervall skulle, för användaren, kunna kännas som om denne inte har kontroll över vad som händer. Resultatet från den prototypen är dock att den skickar väldigt lite data varje gång men att uppdateringsfrekvensen blir viktig när det gäller belastningen av webbservern. Ett bättre sätt att implementera notifiering skulle vara med hjälp *server-push*(se kapitel 5.3) teknik. På så sätt effektiviserar du kommunikationen så att det bara skickas data när något

verkligen har hänt. Den prototypen som gav det tydligaste resultatet när det gäller att minska datamängden var Listor. Av denna anledning så kommer resultatdelen främst att fokusera på resultatet jag fick av listorna. Alla prototyper har implementerats med hjälp av Microsoft ASP.NET AJAX samt även med hjälp av egna JavaScript moduler som hanterar den asynkrona kommunikationen. Det enda undantaget är Dra-och-släpp prototypen där jag har använt mig av script.aculos ramverk för manipulering av webbläsarens DOM. Däremot så sker all kommunikation med hjälp av egna AJAX implementationer.

För att undersöka hur mycket mindre data man skickar genom att använda sig av Ajax så använde jag mig av Fiddler (14), ett program som analyserar HTTP trafik. Här nedan följer resultat (se tabell 2) från en jämförelse av en lista som hämtar alla poster med tillhörande information som döljs bakom ”mer info” länken och en min prototyp som enbart hämtar data i form av json som sedan renderas av webbläsaren.

	Synkron kommunikation	Asynkron kommunikation
Lista (utan ”mer info”)	~7752 bytes	~7262 bytes
Tooltip (informationen bakom – ”mer info”)	~181217 bytes	-
Totalt	~188969 bytes	~7262 bytes

Tabell 2 Storleken på http meddelandena i synkron respektive asynkron kommunikation.

Dessa siffror anger hur många bytes som http meddelandet innehåller när man begär en ny sida med tio poster från databasen. Båda anropen gäller exakt samma poster från databasen och resultatet visar att http meddelandet är cirka 25 gånger mindre när man bara hämtar själva listan och struntar i all information som ska visas när man klickar på ”mer info” länken. Om man bara tittar på själva listan så gjorde jag ett test genom att ta bort all extra information som laddas i en standard lista och då ser man att vinsten att använda Ajax sjunker betydligt. I det enkla fallet så är storleken på meddelandena nästan lika stora. Detta beror mest troligt på att mitt enkla fall så är själva listan extremet liten. Det har bara två kolumner med lite data från databasen och då märks inte skillnaden mellan att skicka html och Json på samma sätt. Sedan så innehåller det asynkrona svaret från servern data för tre sidor, enligt Predictive Fetch Pattern, och det försämrar siffrorna för den asynkrona kommunikationen. En fråga man kan ställa sig är varför jag har implementerat ett mönster som ökar mängden data som skickas. Svaret på den frågan är att jag märkte tidigt hur enormt stor skillnad det blev mellan asynkron och synkron kommunikation så jag valde att även försöka göra gränssnittet mer reaktivt. Jämförelsen mellan asynkron och synkron kommunikation kanske hade blivit tydligare om jag inte hade gjort det men jag tycker att vinsten är mycket tydlig trots detta val.

Om man tittar på upplevelsen för användarna så visar Ajax implementationen en annan vinst. Istället för att hela tiden ladda om hela sidan föra att visa ny data från servern så kan man få ett mycket smidigare gränssnitt genom att hämta data bakom kulisserna med Ajax och sedan rendera ut det till klienten via JavaScript, man slipper alla omladdningar av sidan där det upplevs som om sidan ”blinkar till”.

Nackdelarna med AJAX är att antalet anrop mellan klient och server mångdubblas och kan ske utan att användaren egentligen vet om att webbläsaren hämtar data. Hur man utformar gränssnittet mot användarna får en större betydelse och olika typer av status indikatorer bör användas för att kommunicera till användaren att något ska uppdateras. Microsoft ASP.NET AJAX har en egen klass för detta ändamål, men även script.aculo har tagit detta i beaktande när man konstruerat sitt JavaScript ramverk.

5.3 Reflektioner

En sak som jag har märkt under mina fördjupningar är mängden av så kallade AJAX bibliotek. I många fall så handlar det enbart om DHTML ingen som helst koppling till att

kommunicera mellan klient och server. Så det råder en viss förvirring om vad som AJAX egentligen står för. När det gäller mina prototyper så har jag inte använt mig av Xml för att representera strukturerad data som skickar mellan klient och server utan jag har valt att använda Json istället. Anledningen till detta är i första hand för att jag ville ta reda på vad Json egentligen var men det är inte det enda skälet. Det verkar nämligen som om Json har blivit lite av standard sättet att skicka data mellan klient och server när man arbetar med Ajax. Främst tack vare kopplingen mellan JavaScript objekt och Json. Det är en enkel match att skapa objekt av en Json sträng med hjälp av JavaScript på klientsidan. Xml skapar ett mycket större avtryck när det gäller att tolka Xml på klientsidan. Detta leder till att betydelsen av Ajax, Asynchronous JavaScript and XML, ter sig lite märklig men det beror på att när XmlHttpRequest objektet skapades så var det Xml som användes till att skicka objekt mellan klient och server Om konceptet hade dykt upp idag så hade nog det varit troligare med en förkortning i stil med Ajaj (Asynchronous JavaScript and Json) istället.

5.4 Säkerhet

Denna rapport har inte tagit upp någonting om AJAX och säkerhet. Detta har fått uppmärksamhet av olika grupper på Internet. Anledningen till detta är att AJAX möjliggör för webbutvecklare att på ett enkelt sätt göra anrop till en server utan att användarens vetskap. Detta i kombination med så kallad XSS (Cross-Site Scripting) (15), som är en teknik att utnyttja brister i webbapplikationerna och skicka skadlig kod in i systemet som ekekveras i användarnas webbläsare. De allra flesta anrop är högst oskyldiga men det skulle kunna gå att utnyttja på ett skadligt sätt. Detta har lett till att många AJAX-bibliotek, inklusive ASP.NET AJAX, har gjort det omöjligt att anropa andra domäner än den som själva webbapplikationen ligger på. Man har även lagt till skydd mot XSS genom att kontrollera innehållet som kommer från en UpdatePanel. Det går till exempel inte att skicka in JavaScript utan att det genereras ett fel.

5.5 Design mönster

När det gäller AJAX och designmönster så finns det massor av material i det ämnet men jag ställer mig något frågande till om man egentligen kan kalla det för ”riktiga” design mönster. För att ett mönster ska just bli ett generellt mönster så innebär det att du har ett specifikt problem som kan lösas med hjälp av ett specifikt mönster. Materialet jag ahr hittat när det gäller AJAX och designmönster handlar mer om tips om hur man kan göra en enskild sak. Många så kallade mönster tar upp hur man använder sig av ett specifikt bibliotek istället för att prata om generella metoder. Den bästa boken i ämnet som jag hittade var AJAX Design Patterns (11). Prototypen över listorna följer ett mönster som tas upp i den boken.

5.6 AJAX och skalbarhet

När man tittar på AJAX kommunikation generellt så är det eller kommer snart att vara standard på en webbsida. Redan idag så utnyttjas det på ett flertal webbplatser som till exempel *www.prisjakt.nu*, där laddas sidan med produkter in i webbläsaren i takt med att du *scrollar* ner på sidan. Även all info runt en specifik vara i en lista laddas med liknande tekniker som jag har använt i mina prototyper, självklart är de väldigt optimerade, men det blir väldigt många anrop mellan klient och server när man använder sig av AJAX i den utsträckningen. Personligen så kan jag tycka att man inte behöver använda sig av AJAX bara för att man kan. Det bör ju vara skalbart och inte påfresta webbservern alltför mycket. Om man bygger en större webbapplikationen med massor av användare så bör man nog undersöka vilka funktioner som verkligen blir bättre av att införa asynkron kommunikation. Eftersom att antalet serveranrop mångdubblas jämfört med traditionella webbapplikationer så kommer brister på webbservern att visa sig ganska snart. Applikationer som fungeras alldeles utmärkt idag kan helt enkelt krascha för att webbservern som applikationen ligger på inte klarar den ökande lasten. Att minska mängden data som skickas kan göra webbservern långsam på grund av det ökande antalet asynkrona anrop. Resonemanget leder till att det kommer kanske att dröja ett tag innan själva tekniken har ”landat”, den är bra när den utnyttjas på rätt ställen.

6 Vidare forskning

6.1 AJAX

När det gäller AJAX så har det kommit för att stanna, sen i vilken form man kommer att använda det är nog lite mer oviss. Adobe har stora planer med sin utvecklingsplattform Apollo (15), Microsoft arbetar för högtryck på sin Silverlight (16). Den senare är en plugin till webbläsaren i likhet med Adobes *Flashplugin*. Resultatet båda är ute efter är att gå runt begränsningarna i själva webbläsaren och åstadkomma Ajax-applikationer med hjälp av en plugin istället. Redan idag så är *Flash* något av en webbstandard för interaktiva webbapplikationer. Ett intressant forskningsområde skulle vara att undersöka dessa plugins närmare för att se hur den asynkrona webbkommunikationen är implementerad.

6.2 Web 2.0

Idag pratas det mycket om nästa generation webbapplikationer. Ett intressant område skulle kunna vara att se vilken roll AJAX har i det mycket större begreppet Web 2.0. Kommer det överhuvudtaget vara via webbläsare som man tar del av information från Internet.

Referenser

1. **Krisberedskapsmyndigheten.** WIS, Webbaserat Informationssystem. [Online] 2006. http://www.krisberedskapsmyndigheten.se/templates/EntryPage____5722.aspx.

2. **Moroney, Lawrence.** *Foundations of Atlas, Rapid Ajax Development with ASP.NET 2.0.* u.o. : APRESS, 2006. ISBN:1590596471.

3. **Esposito, Dino.** *Microsoft Internet Developer.* [Online] 1998. [Citat: den 5 Februari 2007.] http://www.microsoft.com/mind/0498/cutting0498.asp.

4. **Wikipedia.** XmlHttpRequest. *Wikipedia.* [Online] 2007. [Citat: den 5 Februari 2007.] http://en.wikipedia.org/wiki/Xmlhttprequest.

5. **O'Reilly, Tim.** What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *O'Reilly.* [Online] 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

6. **Wikipedia.** [Online] http://en.wikipedia.org/wiki/Image:Overview_of_a_three-tier_application.png.

7. **Garrett, Jesse James.** [Online] http://www.adaptivepath.com/publications/essays/archives/000385.php.

8. **Krisberedskapsmyndigheten.** Om Krisberedskapsmyndigheten. [Online] http://www.krisberedskapsmyndigheten.se/templates/Page____4.aspx.

9. **Krisberedskapsmyndigheten**. [Online] http://www.krisberedskapsmyndigheten.se/templates/Page____5.aspx.

10. **Microsoft Corporation.** ASP.NET AJAX Documentation. [Online] http://ajax.asp.net/docs/overview/default.aspx.

11. **Mahemoff, Michael.** *Ajax Design Patterns.* u.o. : O´Reilly, 2006.

12. **script.aculo.us.** web 2.0 javascript framework. [Online] http://script.aculo.us/.

13. **Prototype.** [Online] http://www.prototypejs.org/.

14. **2007 Microsoft Corporation.** Fiddler HTTP Debugger. [Online] http://www.fiddlertool.com/fiddler/.

15. **Wikipedia.** XSS, Cross Site Scripting. [Online] http://en.wikipedia.org/wiki/Cross_site_scripting.

16. **Adobe.** Adobe Labs - Apollo. [Online] http://labs.adobe.com/wiki/index.php/Apollo.

17. **Microsoft Corporation.** Silverlight. [Online] http://silverlight.net/.

18. **Asleson, Ryan och T. Schutta, Nathaniel.** *Foundations of Ajax.* u.o. : APRESS, 2005. ISBN:1590595823.