Utformning och implementation av en interaktiv miljö för andraspråksinlärning

Stefan H Westlund
d98-swe@nada.kth.se

TRITA-NA-XXXX

Examensarbete i Datalogi om 20 poäng
vid Programmet för datateknik,
Kungliga Tekniska Högskolan, 15 juni 2004.
Uppdragsgivare: IPLab, KTH Nada
Handledare på Nada: Ola Knutsson & Teresa Cerratto Pargman
Examinator: Kerstin Severinson-Eklundh

Utformning och implementation av en interaktiv miljö för andraspråksinlärning

Sammanfattning

Syftet med det här examensarbetet är att utveckla en interaktiv inlärningsmiljö för andraspråksinlärare, men inlärningsmiljön ska dock inte utesluta andra tänkbara målgrupper. Det finns många olika typer av inlärningsmetoder som kan stödja andraspråksinlärare i deras språkliga utveckling, men i många fall har inlärarna fått anta en passiv roll i inlärningsprocessen. Med denna nya interaktiva miljö är det tänkt att inlärarna ska kunna få ta en aktivare roll.

För att genomföra detta projekt har det gjorts en grundlig genomgång av de tekniska förutsättningarna, samt vilka möjligheter som existerar. Den tekniska lösningen utgår ifrån befintliga tjänster på KTH Nadas servrar. De språkliga tekniker som servrarna tillhandahåller är resultatet av flera års forskning. Inlärningsmiljön sluter samman de olika forskningsinriktningarna till en gemensam plattform att utvärdera forskningen på.

Att från grunden konstruera ett förhållandevis komplext grafiskt gränssnitt tar lång tid och ställer utvecklaren inför många olika valmöjligheter. I stort sett har samtliga presenterade tekniker såväl för- som nackdelar. I slutänden blir det en kompromiss som främst måste utgå från användarens perspektiv och behov. Med hjälp av den tänkta målgruppen och deras önskningar har många beslut fattats för den fortsatta utvecklingen. Examensarbetet har lagt grunden till en modulär plattform och en vidareutveckling av detta projekt är inte bara möjligt utan påbjudes.

Design and implementation of an interactive environment for second language learning

Abstract

The purpose of this master project is to design and implement an interactive learning environment for second language learners, but the learning environment should not exclude other conceivable target groups. There is different learning methods that can support second language learners in their language learning, but in many cases the learners adapt passive roles in the learning process. With this new interactive environment the learners could get a more active role.

To complete this project there has been done a thorough review of the technical prerequisites and which possibilities may exist. The technical solution is based on available services at KTH Nada's servers. The language techniques that the servers provide is the result of several years of research. The learning environment brings together the different directions to a common platform to evaluate the research.

To construct a comparatively complex graphical user interface from the beginning takes a long time and will put the developer in front of different choices. Almost all of the presented technologies have advantages and disadvantages. At last there will be a compromise that mainly must consider the user's needs from the user's perspective. With help from the target group and their wishes many of the descisions have to take place for further development. The master project has created a base for a modular platform and further development of the project is not only possible but a decree.

Innehåll

Förord

Det här examensarbetet har jag gjort på uppdrag av Interaktions- och presentationslaboratoriet (IPLab) på Institutionen för numerisk analys och datalogi vid Kungliga Tekniska Högskolan i Stockholm under åren 2003–2004.

Mitt intresse för språk och språkteknologi har varit en underlättande faktor för att slutföra detta projekt. Väldigt mycket tid har gått åt till det arbete som inte syns efteråt, såsom kontakter med användare och administratörer om lingvistiska och funktionella detaljer.

Jag skulle vilja passa på att tacka mina handledare Ola Knutsson och Teresa Cerratto Pargman som hjälp mig framåt under projektarbetet. Även ett tack till de lärare och studenter vid Stockholms universitet som bidrog med idéer till inlärningsmiljön. Dessutom ett tack till alla som hjälp till med tekniska detaljer i projektarbetet, ingen nämnd – ingen glömd. Sist men inte minst ett stort tack till min flickvän och sambo Jennifer för förståelsen att så mycket tid gått åt till detta arbete.

Stefan H Westlund, 1 Juni 2004

Kapitel 1

Inledning

Den här rapporten är en del av det examensarbete som jag genomfört under åren 2003–2004 på Institutionen för numerisk analys och datalogi vid Kungliga Tekniska Högskolan i Stockholm, i fortsättningen kallat KTH Nada.

Då jag påbörjade mitt examensarbete våren 2003 hade jag redan fått en överblick av projektet och vad som förväntades av det. Ett par månader innan hade jag träffat min handledare Ola Knutsson på kursen i Språkteknologi. Tillsammans med min andra handledare Teresa Cerratto Pargman försökte vi finna realistiska alternativ till traditionell inlärning som kan gynna just andraspråksinlärare.

1.1 Historik

Språkgranskaren Granska, som har utvecklats på KTH Nada, har problem med att analysera de texter som andraspråksinlärare skriver. Därför har en ytparser kallad Granskas TextAnalysator utvecklats.

Till skillnad från Granska, som opererar på disambiguerad ordklasstaggad text och en flertydig partiell frasstrukturanalys, har Granskas TextAnalysator lyft analysen till en entydiggjord frasstrukturanalys [Knutsson et al., 2003]. Det gör att det är möjligt att skriva granskningsregler som ger färre falska alarm samtidigt som mindre komplexa regler kan upptäcka mer komplexa typer av fel[1].

Detta examensarbete bygger på forskningsresultat från projektet *The use of language tools for writers of Swedish as a second language* som är ett samarbete mellan KTH Nada och Institutionen för Data- och systemvetenskap vid Stockholms universitet. Projektet leds av Teresa Cerratto Pargman och finansieras under åren 2003–2005 av Vetenskapsrådet inom Utbildningsprogrammet.

Mycket av den språkteknologi som examensarbetet baseras på är utvecklat inom forskningsprojektet *Integrated language tools for writing and document handling* som leddes av Kerstin Severinson-Eklundh under åren 1998–2000. Denna språkteknologi vidareutvecklas nu i projektet *CrossCheck - svensk grammatikkontroll för andraspråksskribenter* som är ett samarbete mellan KTH Nada och Institutionen för

[1] http://www.nada.kth.se/theory/projects/xcheck/lagesrapport-030630.html

Lingvistik vid Stockholms universitet. CrossCheck leds av Viggo Kann och stöds av Vinnova inom Språkteknologiprogrammet.

1.2 Syfte

Den nya textanalysatorn Granskas TextAnalysator ska få ett interaktivt grafiskt gränssnitt som kan ta tillvara och visualisera informationen från textanalysen.

Dessutom bör olika typer av skrivstöd och inlärningsmoduler kunna läggas till även efter det att examensarbetet avslutats. Examensarbetet baseras på att utveckla en inlärningscentrerad miljö som på olika sätt kan utnyttja tjänster som Granska och Granskas TextAnalysator för att stödja användaren där skrivandet är en del av språkinlärningsprocessen.

Syftet är att låta inlärarna få ta en aktivare roll i inlärningsprocessen. Istället för att passivt bli serverade uppgifter ska inlärarna få utforska språket med sina egna eller andras texter.

1.3 Uppgift

Min uppgift var att utveckla en inlärningsmiljö för det svenska språket där andraspråksinlärare ska kunna ta del av den information som genereras av befintliga tjänster som finns på KTH Nadas servrar. De tjänsterna är i dagsläget främst Granska och Granskas TextAnalysator. Inlärningsmiljön ska integrera dessa tjänster på ett sådant sätt att inläraren inte märker av dem förrän denne behöver använda dessa verktyg. Ytterligare tjänster som ordlexikonet Lexin och ordböjaren Inflector bör kunna integreras i inlärningsmiljön. Dessa är båda delvis utvecklade på KTH Nada och hittills är det bara Lexin som har fått ett webbgränssnitt.

Utöver dessa redan befintliga tjänster tillkom en önskan om lingvistisk redigering. Till exempel ska nu alltså ett trippelklick inte markera en rad, som är standardbeteendet, utan en fras eller en mening. Det fanns också önskan om en funktion för att kunna byta ut text ur en viss ordklass till något annat som exempelvis understreck. Lingvistisk sökning och ersättning skulle kunna vara ett praktiskt hjälpmedel för lärare inom andraspråksundervisning som enkelt skulle kunna skapa lucktexter av godtyckliga texter.

Det var inte specificerat på vilket sätt som inlärningsmiljön skulle fungera eller hur det borde utvecklas. Däremot lades en stor vikt på att inlärningsmiljön ska kännas interaktiv. En annan av de viktiga frågorna var hur miljön ska stödja inlärarna i skrivprocessen. En del användare vill kanske granska varje mening för sig, andra granska styckesvis och slutligen har vi dem som kanske vill granska hela texten i ett svep.

1.4 Förkunskaper

Som Borin påpekar bör den som utvecklar programvara för språkinlärning ha en grundläggande kunskap inom datorlingvistik eller likvärdigt [Borin, 2002]. Dessutom bör utvecklaren ha en förståelse för inlärares och lärares speciella behov, om denne inte ska skapa lösningar till ickeexisterande problem. Följden kan bli att utvecklaren misslyckas med att finna lösningar till de existerande.

Han tror heller inte att det är realistiskt att begära att en datorkunnig språklärare ska konstruera tekniska lösningar på programmeringsproblem, eftersom de saknar den grundläggande kunskapen om algoritmer och komplexitet. Inte helt oväntat förordar han datorlingvistik som den ultimata modellen för utveckling av språkinlärningsapplikationer. Själv håller jag inte riktigt med eftersom en datorlingvist omöjligen kan ha lika hög nivå på datorvetenskap som en datoringenjör. Felmarginalerna är oerhört små inom just programutveckling. En produkt kan vara helt oanvändbar om den inte är optimerad och från grunden rätt konstruerad. Däremot behöver ingenjören mycket hjälp med lingvistiska termer och bruk. På denna punkt har som tur är mina handledare hjälpt mig att överbrygga dessa hinder.

Mina förkunskaper inom datorstödd språkinlärning var mycket begränsade och jag fick rådet av mina handledare att gå grundkursen i Människa-datorinteraktion under HT 2002. Eftersom mycket av detta examensarbete handlar om grafiska gränssnitt och interaktion kunde jag inte ha gått en lämpligare kurs. Jag hade redan då en ganska klar bild över problemet och ville visualisera syntaktisk information direkt i dokumenttexten, men jag hade inga speciella kunskaper inom detta område.

Kapitel 2

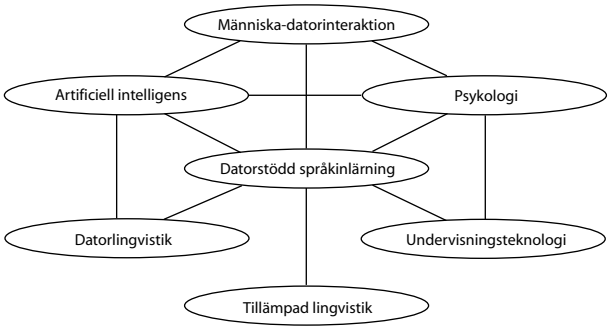
Bakgrund

I mitt examensarbete har jag kommit i kontakt med många tekniker för att utveckla applikationer på Internet. En del har varit otillräckliga medan andra utmärkta. Jag har också fått inblick i teorier när det gäller andraspråksinlärning. Här sker en kort genomgång av de viktigaste teknikerna som jag refererar till i rapporten.

2.1 Datorstödd språkinlärning

Datorstödd språkinlärning, eller *Computer Assisted Language Learning*, är en relativt ny term och togs i bruk i början av 80-talet. Det kan kortfattat beskrivas som utveckling av och studier inom datorbaserade applikationer för språkundervisning och språkinlärning. Datorstödd språkinlärning är en tvärvetenskaplig disciplin och har förekommit som teori inom den akademiska världen i över trettio år. Enligt Levy [Levy, 1997] kombinerar datorstödd språkinlärning flertalet olika forskningsområden, se figur 2.1.

Det finns en mängd olika tillämpningsområden av datorstödd språkinlärning där andraspråksinlärning på skolorna är bland de mest uppenbara. Ytterligare områ-



Figur 2.1. Datorstödd språkinlärning och dess samband med relaterade forskningsområden [Levy, 1997].

den kan vara för självstudier såsom Linguaphone och korrespondensstudier där man lätt kan se hur datorn direkt skulle kunna öka kvalitén på lärandet, precis som när kassettbanden ersatte grammofonskivorna. Då ökade antalet potentiella självstudiesituationer genast drastiskt. Ett annat område där datorstödd språkinlärning förväntas revolutionera är att upprätthålla och återuppliva små språk, huvudsakligen minoritetsspråk och regionala språk. När det gäller sådana språk finns ofta ingen undervisningsorganisation med utbildade lärare och heller inga medel för att skapa någon sådan organisation. Ett annat mycket tydligt exempel på tänkta målgrupper är historiker och arkeologer som vill lära sig att tyda utdöda språk och deras skrifter.

Det är viktigt att påpeka att målet med att införa datorstöd i den vanliga lärarledda undervisningen inte är att ersätta utan att understödja läraren och eleven. Datorer som hjälpmedel i undervisningen ska inte användas bara för att det är tekniskt möjligt [Borin, 2003]. Datorstödd språkinlärning är inte en hammare och inte alla undervisningsproblem är spikar[1].

2.2 Andraspråksinlärning

Inom lingvistiken karaktäriseras språkinlärning med den språkform det handlar om. Förstaspråk (eller modersmål), andraspråk och främmande språk är de tre områden som nämns. Skillnaden mellan andraspråksinlärning och främmandespråksinlärning skulle vara huvudsakligen den, att andraspråket används i den miljö där inlärningen sker. Detta är inte fallet med främmandespråksinlärningen. Svenska i Sverige är ett typiskt exempel på andraspråk, medan ryska i Sverige är ett typiskt exempel på främmande språk.

Sedan länge har språkinlärning svängt mellan två fundamentala inriktningar. Den ena inriktningen tar fasta på regel- och grammatikinlärning, en så kallad *formalism*. Den andra inriktningen tar fasta på praktisk språkinlärning och praktiskt språkande, en så kallad *aktivism*. Från att ha haft en stark tilltro till formalismen har dagens språkpedagoger antagit aktivismen som den bättre modellen för inlärning. Formalismen har dock inte helt övergetts.

Språkinlärning är mycket individuellt och några genvägar kan inte säkerställas. Vid såväl första- som andraspråksinlärning har man antagit att det finns en naturlig inlärningsföljd som alla språkinlärare måste genomgå, oavsett deras bakgrund och modermål. Inläraren kan alltså inte lära sig ett visst språkdrag innan denne har lärt sig de föregående. Detta antagande har dock fått visst motstånd från forskning som tyder på att även om man tränar på en högre nivå för tidigt så underlättar träningen den senare inlärningen av denna nivå.

Generellt sett påverkas inlärningen av flera yttre faktorer som till exempel i vilken miljö inlärningen sker och inlärarens känslotillstånd. En inlärare som är stressad eller frustrerad antas inte lära sig lika effektivt som en glad, rofull och harmonisk inlärare. En konsekvens av detta är att försöka skapa en god inlärningsmiljö, vilket inte alltid är så lätt i en skolmiljö. Det kan också finnas en ovilja mot språket, dess talare

[1] ”CALL is not a hammer and not every teaching problem is a nail!” - Chen, 1996

och kultur. Dessutom kan ekonomiska och sociala problem vara ett stort hinder i språkinlärningen.

Att ordinlärning har en central roll inom andraspråksinlärning anser de flesta forskare. Andraspråksinlärare, såväl som infödda talare, utökar ständigt sitt ordförråd i den meningen att de lär sig fler ord. Men också för att de lär sig mer om betydelsen av ord som de redan har stött på, när de på nytt stöter på dem i främmande sammanhang [Borin, 2003].

2.3 Java

Java är ett programmeringspråk som vuxit till sig rejält allteftersom utvecklare av applikationer upptäcker den rika samling av verktyg som ingår i grundutförandet av Java 2 Software Development Kit (J2SDK). Språket är plattformsoberoende och objektorienterat. Givetvis stöds inkapsling, arv och polymorfism som är standard för objektorienterade språk.

2.3.1 Java Swing

Java Swing är en del av Java Foundation Classes (JFC) och utgör själva grunden i Java för att bygga robusta och interaktiva gränssnitt. Komponenterna i Java Swing är lättviktskomponenter, vilket innebär att de inte har någon motsvarighet i det aktuella operativsystemet och alltså är oberoende av det.

Föregångaren till Java Swing är Abstract Window Toolkit (AWT), som baseras på tungviktskomponenter och beror på operativsystemet. AWT är alltså plattformsberoende vilket inte Java Swing är. En av de största skillnaden mellan AWT och Java Swing är att AWT inte stödjer arkitekturen Model-View-Controller (MVC), vilket Java Swing gör till fullo.

När utvecklare konstruerar gränssnitt i Java bör de inte blanda lättvikts- och tungviktskomponenter. Anledningen är att det finns många grafikproblem som kan inträffa när de två olika komponentbaserade typerna blandas. Det finns heller inga skäl att använda tungviktskomponenter förutom vid specialfall som vid utskriftshantering. Denna typ av hantering är inte oberoende av operativsystem.

Även om komponenter från AWT bör undvikas tillsammans med Java Swing så utnyttjar Java Swing mycket funktionalitet från AWT. Bland annat används layout- och händelsehanterare. Detta är dock ingenting som påverkar grafikrutinerna. Förutom några få toppnivåkomponenter är nu de flesta av komponenterna i Java Swing skrivna i 100 procent Java. Det innebär att ett fel i gränssnittet på en plattform förmodligen är ett fel i ett gränssnitt på en annan plattform. För utvecklarna av Java Swing är det en stor skillnad att korrigera källkoden på ett ställe istället för på varje individuell plattform.

2.3.2 Java Web Start

Kortfattat kan man säga att Java Web Start är en teknologi som möjliggör distribuering av applikationer (eller Java Applets) genom vanliga webbservrar. Applikationen hämtas av webbläsaren och körs sedan som en vanlig fristående applikation. När applikationen sedan är installerad, givetvis automatiskt, så behövs den inte hämtas igen då den lagras lokalt, förutsatt att det inte finns en senare version på webbservern.

Eftersom Java Web Start är en applikation som i sig själv är skriven i Java så är mjukvaran plattformsoberoende och stöds av alla system som stödjer den ytterst robusta Java-plattformen. Denna plattform har också en omfattande säkerhetsarkitektur. En stor fördel med Java Web Start är att den medger multipla Java-plattformar. Applikationen kan också efterfråga en specifik version. Om en äldre version är installerad på klienten kan Java Web Start automatiskt hämta och installera en senare.

Då en applikation körs utanför webbläsaren så kan det vara lätt att tro att applikationen har full åtkomst till datorns olika resurser. Men så är inte fallet utan applikationen körs inuti en begränsad miljö på samma sätt som Java Applets i en vanlig webbläsare. Användare kan alltså köra applikationer från okända källor, som kanske inte är tillförlitliga, utan att behöva tänka på olika säkerhetsrisker.

Med Java Web Start så uteblir det stora problemet att få Java att fungera hos olika webbläsare. Java Applets har inte ett gott rykte bland utvecklare och användare, eftersom Microsofts webbläsare Internet Explorer inte inkluderar en fullgod version av Java. Syftet är förstås att inte låta Java ta en allt större bit av gångbara webbtekniker och därmed konkurrera ut egna.

Det är enkelt att installera applikationer med Java Web Start och samtidigt kan användaren välja att integrera applikationen med dess egna ikoner på skrivbordet. För den genomsnittlige användaren så är applikationen ett program bland alla andra och uppför sig som sådana.

En applikation kan begära obegränsad åtkomst till användarens system. I så fall visar Java Web Start en säkerhetsvarning när applikationen startas för första gången. Säkerhetsvarningen visar information om tillverkaren som utvecklade programmet. Om användaren väljer att lita på tillverkaren kan programmet startas. Informationen om programmets ursprung baseras på digitala certifikat.

Java Web Start har en egen programhanterare där användaren enkelt kan starta program som tidigare startats med Java Web Start. Med hjälp av denna programhanterare kan användaren få ytterligare information om program och hitta länkar till respektive programs hemsida. Inifrån denna programhanterare kan användaren granska och ändra olika inställningar för Java Web Start.

Kapitel 3

Problem

Att konstruera en interaktiv inlärningsmiljö för andraspråksinlärare var huvudproblemet i mitt examensarbete. Det problemet innefattade flertalet delproblem varav det första var att transformera information från eXtended Markup Language (XML) till motsvarande representation av objekt i Java. Eftersom den redan befintliga servern för Granskas TextAnalysator kommunicerar via XML måste en säker och tillförlitlig transformering av denna kunna ske. Nackdelen med denna teknik är att ytterst lite information är känd om den interna strukturen på dokumenten i XML.

Det andra problemet var att leta efter en lämplig plattform för användning av grafiska gränssnitt. Det är betydligt större problem än det låter som eftersom hänsyn till interaktivitet och tillgänglighet ska prioriteras. Dessutom måste inlärningsmiljön uppfattas förtroendeingivande eftersom ett dåligt program sänker helhetsintrycket.

Ytterligare ett problem var att externa språkrelaterade webbtjänster ska kunna integreras i applikationen och samtidigt försöka leda användarna framåt i inlärningsprocessen. De externa tjänsterna måste följa samma mönster som de interna, det vill säga att de måste upplevas vara integrerade med webbapplikationen och kunna interagera med denna.

Det fjärde problemet var att applikationen inte får bli ett hinder för inlärarna och deras språkliga utveckling. Eftersom både datorkunskaper och den språkliga nivån på användare kan skilja rejält så måste inlärningströskeln vara så låg som möjligt för att alla ska kunna ta del av informationen. På samma sätt som det måste finnas tydliga funktioner för att slå på och av olika regler i applikationen [Knutsson et al., 2002a] måste det också gå att slå på och av syntaktisk information.

Slutligen, applikationen ska inte förtrycka inläraren med att försöka kategorisera den språkliga nivån, utan istället hjälpa denne att resonera kring och reflektera över språket [Knutsson et al., 2002b]. Detta är en avvikelse mot de övriga program som finns tillgängliga inom andraspråkinlärning.

Kapitel 4

Lösning

För att konstruera en miljö för andraspråksinlärning finns det ingen tänkbar steg-för-steg-metod. Mycket av utvecklingen måste istället ske på känsla och med hjälp av lärare och andraspråksinlärarna själva. Vad är den mest ultimata modellen för andraspråksinlärning? Ja, det finns många versioner om vad som borde göras men få konkreta resultat. För min egen del så tror jag att mycket ligger i att få utforska språket på ett valfritt sätt, helst i kombination med traditionell undervisning. Samtidigt vilar då ett större ansvar på inläraren för att denne ska lyckas.

Mycket av andraspråksinlärningen består idag av lucktexter och grammatiklära där texterna är utvalda av språklärare och kunniga författare. Inlärarna själva har inget annat val än att följa en snitslad bana dikterad av dessa. Å andra sidan är just lärarna ohyggligt pressade av sitt arbete, inte bara i rollen som lärare utan också som administratörer. Lärarna får allt större klasser där vissa av eleverna kräver mer tid än andra och till slut leder detta till försämrad undervisning för samtliga. Den kompetens som finns hos lärarna måste bättre kunna tas tillvara.

Ett rätt utformat språkinlärningsprogram skulle kunna avlasta inlärare i deras skrivarbete genom att hjälpa dem mer effektivt upptäcka lokala och formella problem i sina texter. Det skulle också kunna hjälpa ovana skribenter att uppmärksamma och hantera nya problem genom att bistå med vägledning. Med rätt instruktivt stöd skulle det kanske också kunna uppmuntra till ett mer genomgripande revisionsarbete, om inte annat så i ett pedagogiskt sammanhang där programmet skulle kunna avlasta och assistera läraren i dennes roll med elevernas texter [Domeij, 2003].

Om man alltså konstruerar ett program för språkinlärning som skulle kunna hjälpa lärarna med enklare typer av grammatiska fel kan lärarnas dyrbara tid läggas på svårare grammatiska konstruktioner som grammatikprogrammen inte klarar av ännu. Visserligen bygger det på att eleverna har allmäna datorkunskaper och inte känner sig främmande inför en ny teknik som informationsteknik. Idag har dock flertalet elever i skolan tillgång till datorer men för vuxenstuderande kan det vara sämre ställt. Dessutom måste de verktyg som utför den språkliga analysen implementeras effektivt och robust för att skribenterna inte ska sluta att använda applikationen för att programmet är för långsamt eller instabilt [Knutsson, 2001].

4.1 Förutsättningar

4.1.1 Begränsningar

Utanför storstadsområdena är datorer i hemmen ofta uppkopplade mot Internet med långsamma telefonmodem. Tyvärr är detta ett problem som inte kan åtgärdas i detta projekt eftersom bland annat den automatiska språkgranskningen kräver kommunikation mot befintliga servrar på KTH Nada.

En fullständig integration av servermjukvaran för språkgranskning ihop med det grafiska gränssnittet är inte planerad och kanske orealistiskt att genomföra. Analysen kräver förhållandevis kraftfulla datorer. För att även användare med lite äldre datorer ska kunna använda sig av programmet är det inte ett bra alternativ att integrera den komplexa logiken. Dessutom är koden för språkgranskningen oskyddad och exponerad om denna skulle medfölja programmet. Ytterligare ett argument för att detta vore orealistiskt är att algoritmerna för språkanalysen i sådana fall måste konstrueras till samtliga plattformar, med exempelvis Java. För närvarande är denna källkod skriven i C++ och alltså inte plattformsoberoende. Att översätta denna källkod är dock ett alldeles för stort arbete för detta examensarbete.

4.1.2 Möjligheter

Min lösning måste ta hänsyn till både tillgänglighet och inlärarnas generella förkunskaper. Att programmet är gratis för alla intresserade är dels ett sätt att offentliggöra resultat av språkteknologisk forskning för allmänheten, men dels en förutsättning för att kunna få en stor spridning. Och just spridningen är viktig om projektet ska kunna vidareutvecklas. Ju fler intresserade användare, desto större chans att även berörda myndigheter lyssnar och förhoppningsvis bidrar till vidare utveckling.

Relativt tidigt i projektet bestämde jag mig för att konstruera en neutral plattform för skrivande eftersom skrivandet har en central roll i språkinlärningen. Microsoft Word är en välkänd och kompetent ordbehandlare [McGee and Ericsson, 2002] som under flertalet år har använts på skolor och en sådan miljö skulle kunna ge positiva resultat. Användarna kan då utnyttja den kunskap de redan har från exempelvis Word. Men till skillnad från Word, som i stora drag fokuserar på form och funktionalitet, ska detta projekt lyfta fram språkinlärningen och de språkliga verktygen. Om dessa verktyg kan integreras på ett naturligt sätt för inlärarna kan chansen förbättras till ökad inlärning.

4.2 Teknisk förundersökning

I början av projektet hade jag ingen speciell djupare kunskap inom något enskilt programmeringsspråk utan kände mig måttligt kunnig inom flertalet. Eftersom en stor del av den befintliga tekniken av Granska och Granskas TextAnalysator bygger på XML krympte mängden realistiskt genomförbara programmeringsspråk till C++

och Java eftersom det finns gott om parsers för dessa språk. Men ett grafiskt gränssnitt som bygger på C++ skulle inte vara portabelt och ha stora tekniska brister vad gäller versionshantering och uppdatering.

Valet föll på Java som har fått ett rejält uppsving sedan Internet blev populärt i mitten av 90-talet. Sedan dess har mycket kraft lagts på serversidan och det har lett till ett gott rykte bland utvecklare runt om i världen. På de senaste åren har som tur är en stor fokusering på klientsidan fått Sun[1] att utveckla nya tekniker för grafiska gränssnitt.

Ett flertal kommersiella program på marknaden är konstruerade i Java även om de flesta är baserade på öppen källkod, däribland den oerhört komplexa utvecklingsmiljön Eclipse. The Eclipse Project[2] är ett projekt som baseras på öppen källkod men stödjs bland annat av IBM. Eclipse används för att utveckla applikationer, Applets och andra tekniker skrivna i Java.

4.2.1 Befintlig XML

Det dokument[3] som beskriver Granskas TextAnalysators XML är idag mycket stort och har tagit en lång tid att utveckla. Anledningen ligger i att den XML som genereras av Granskas TextAnalysator mer kan ses som en sorts pseudo-XML i stil med Hypertext Markup Language (HTML). Varningar och noteringar som är svåra att förutse förekommer också allt som oftast i denna XML. Dessutom är vissa objekt inbäddade tillsammans med primitiva dataobjekt som strängar, vilket gör att ordningen mellan dessa går förlorad om inte ramverket stödjer detta. Den XML som genereras var från början inte tänkt att kommunicera med yttre moduler såsom grafiska gränssnitt.

Det kan säkert anses som onödigt att skapa ett komplett beskrivande dokument eftersom det kan genereras dynamiskt av ramverket under parsning, men då blir det också en helt annan komplexitet. Jag föredrog en exakt och robust parsning för att användaren skulle uppleva gränssnittet så interaktivt som möjligt. Alltför mycket arbete av processorn skulle ha gått åt för att dynamiskt generera Java-klasser eftersom XML-dokumenten är förhållandevis komplexa. Stora delar av dokumenten är dessutom gemensamma. Det beskrivande dokumentet deklareras med XML Scheme Language, som är det modernaste formatet.

4.2.2 JAXB

Det är Sun som utvecklar Java Architecture for XML Binding (JAXB) och produkten har en hög kvalité och genomtänkt design. Prestandamässigt har den en mycket bra parser som kallas Xerces2 Java Parser[4]. Parsern Xerces finns i flertalet olika språk och bygger på öppen källkod. Men tyvärr kunde jag inte få JAXB att fungera så

[1] http://www.sun.com
[2] http://www.eclipse.org
[3] http://www.nada.kth.se/~d98-swe/primer.xsd
[4] http://xml.apache.org

bra ihop med Java Applets eftersom JAXB gör anrop av metoder i Java som är förbjudna inom kontrollerade miljöer såsom Java Applets.

Eftersom osignerade Applets körs i en så kallad sandlåda, vilket innebär att vissa metoder och klasser inte är tillåtna, så misslyckades en av mina första önskningar som jag ställt upp inför projektet. Användare ska inte behöva vara tvungna att godkänna certifikat som medföljer signerade Java Applets. En önskan var att användare ska kunna utforska och använda miljön innan de godkänner certifikatet för att på så sätt få förtroende för programmet.

Det förvånar mig verkligen mycket att inte Sun har tänkt på att det bör gå att använda JAXB ihop med osignerade Java Applets eftersom det är Sun som utvecklat båda dessa produkter. JAXB är visserligen en teknik som fokuserar på serversidan men utvecklarna måste ju ändå ha klientsidan i åtanke när de utvecklar en produkt.

Problemet finns fortfarande kvar i Suns senaste version av JWSDP och allt fler desperata mjukvaruutvecklare har upptäckt detta problem. Svaren jag har fått från Suns utvecklingsavdelning har varit allt ifrån att se efter i någon äldre tråd i deras webbforum som inte existerat till att det bör fungera om en konfigurationsfil hos webbläsaren ändras. Detta är naturligtvis oacceptabelt för en användare av webbapplikationen eftersom de kanske inte ens har någon datorvana. Det är dock oklart om detta är medvetet gjort av Sun eller om det kan anses vara en orättad bugg. Det är synd att detta problem inte gick att komma tillrätta med men det var bara att leta efter någon annan likvärdig produkt.

4.2.3 Castor

Efter att jag letat runt ett tag på Internet efter en likvärdig produkt till JAXB så fann jag projektet Castor[5] som bygger på öppen källkod. Detta projekt har flera olika inriktningar men med en stor fokusering på transformering mellan Java och XML. Castor som ramverk har fått fina recensioner av bland annat IBM:s utvecklingsteam [Sosnoski, 2002]. Castor använder sig liksom JAXB av parsern Xerces, som i stort sett är standard bland de olika ramverk som hanterar XML.

Castor hade inte den allra snabbaste parsern i de tester som IBM har genomfört men jag fokuserade istället på minnesallokeringen. Prestandamässigt är minnesallokering en viktigare aspekt än olika algoritmers lösningar (till en viss del). Orsaken är att Java har ett inbyggt system som hanterar inaktuellt minnesområde, en sorts sophämtare. Castor ligger i särklass på toppen med en minimal allokering. En anledning är att objekten i Java inte behöver skapas dynamiskt ifrån XML, vilket skulle kräva såväl tidsåtgång som extra allokering av minne. Istället genereras samtliga objekt i förväg med speciella medföljande verktyg och sedan distribueras objekten ihop med övriga beståndsdelar i projektet. Även JAXB använder sig av denna metod men har en sämre teknik för minnesallokeringen.

Det var överlag enkelt att konfigurera Castor till att generera passande objekt i Java och utvecklingsteamet bakom Castor är mycket aktivt i de webbforum som finns tillgängliga. De frågor jag ställt till dem har besvarats inom 24 timmar vilket

[5] http://castor.exolab.org

måste anses vara oerhört hjälpsamt. Den hjälp som presenteras på deras webbplats är kortfattad men konstruktiv för den som redan har kännedom om Java och XML.

Det enda jag har att klaga på med Castor är att parsningen ibland misslyckas utan någon tydlig anledning. Trots modifieringar av det beskrivande dokumentet i XML Scheme så fallerar någon procent av de parsningar som genomförs. Felet ligger i det omslutande ramverket och inte i parsern men jag anser ändå att Castor är ett bättre alternativ än JAXB i dagsläget. Då en ny parsning utförs på XML-dokumentet så fungerar det sannolikt. Dessutom är nya och buggrättade versioner tillgängliga allt som oftast så det är mycket möjligt att detta kommer att korrigeras innan projektet är färdigt.

4.3 Gränssnittsutveckling

4.3.1 Inledande kommunikation

I början av projektet var jag fast besluten om att distribuera applikationen som en Java Applet. En enkel webbsida refererade till en Applet som sedan kördes inuti webbläsaren hos klienten. Webbsidan publicerade jag på en lokal webbserver som var publikt åtkomlig. Jag började med hjälp av Borland JBuilder att konstruera ett generellt gränssnitt som förutom meny och knappar hade ett inmatningsfält där användaren kunde skriva in sin text.

I det första steget var jag tvungen att verifiera om det gick att få någon kommunikation mellan Appleten och servern för Granskas TextAnalysator. Kommunikationen implementerades med hjälp av vanlig socketprogrammering. När kommunikationen väl fungerade mellan Appleten och servern för Granskas TextAnalysator så prövade jag med att analysera den text som användaren skrivit in i inmatningsfältet. Texten kom som väntat tillbaka analyserad i form av XML.

I det här tidiga stadiet kom mitt första stora beslut för den fortsatta utvecklingen. Hur skulle jag visualisera den information som analysen producerade? Skulle det accepteras av användare om ett fält var till för inmatning och ett annat för utmatning? Jag beslöt mig här för att studera exempel jag sett på webben lite närmare. Orsaken var att jag väldigt gärna ville ha en dubbelriktad kommunikation mellan dokumenttext och analysresultat. Att de tydliga gränserna mellan dessa skulle suddas ut.

4.3.2 Signering och distribuering

Under den först tiden då jag letade efter relevant information på webben angående visualisering hade jag parallellt studerat verktyg för att signera Applets och applikationer. I J2SDK ingår verktyget Jarsigner för att signera dessa. Först var jag tvungen att generera ett nyckelpar som utgjorde mina egna privata uppgifter.

Nyckelparet är nödvändigt för att signera Applets och är en sorts certifikat enligt en välkänd standard. Med verktyget Keytool som också ingår i J2SDK skapades detta certifikat. Dessa två verktyg är mycket enkla att använda och orsakade inget

problem i mitt projekt. Båda två är kommandoradsbaserade och har alltså inga grafiska gränssnitt.

För att distribuera en applikation eller Applet över webben bör samtliga filer ingå i ett arkiv som Java Archive Files (JAR). I Borland JBuilder finns möjligheter att konstruera olika sorters arkiv av projektfiler och genom en automatisk generering av JAR-arkiv vid varje kompilering sparas mycket tid. Efter varje kompilering kunde jag direkt se om ändringen var tillfredsställande genom att besöka den uppdaterade lokala webbplatsen.

För att signera Applets används den privata nyckeln i nyckelparet. Sedan infogas den publika nyckeln i projektets JAR-arkiv innehållande bland annat mina uppgifter. När JAR-arkivet är signerat har varje enskild fil i arkivet en kontrollsumma och den summan verifieras av klienten för att garantera att ingenting ändrats sedan signering. Jag kunde dock inte få Borland JBuilder att fungera med automatisk signering av JAR-arkiv utan fick manuellt genomföra signering efter varje kompilering.

När Appleten distribueras signerad är det första som syns att webbläsaren öppnar en säkerhetsdialog och frågar användaren efter ett godkännande av medföljande certifikat. Ett godkännande ger Appleten åtkomst till lokala systemresurser som den annars inte skulle ha. Det är webbläsaren som hanterar alla certifikat och fördelar dessa rättigheter individuellt.

4.3.3 De första utvecklingsproblemen

Efter några dagars sökande hade jag fått upp ögonen på webbplatsen[6] för bokförlaget Manning Publications. De har publicerat ett kapitel [Robinson and Vorobie, 2003] från en mycket bra bok på webben om Java Swing. Kapitlet ger en god inblick i hur man konstruerar en enkel och flexibel ordbehandlare.

Ordbehandlaren som beskrivs detaljerat är en applikation och inte en Applet men många tips går ändå att ta vara på eftersom den programmeringsmässiga skillnaden är minimal mellan applikationer och Applets. Jag insåg redan från början att en Java Applet inbäddad i webbläsaren kraftigt skulle påverka och begränsa användarvänligheten. Appletens eget menysystem skulle lätt blandas ihop med webbläsarens eget och användarna skulle riskera förlora sitt arbete så fort de, sannolikt oavsiktligt, lämnar webbsidan. Dessutom är Applets bundna till en fixerad storlek som inte kan ändras dynamiskt under exekvering utan problem.

Min första tvekan att utveckla Java Applets gjorde att jag tog ett beslut om att låta inlärningsmiljön vara en fristående applikation som startas med en knapp i Appletens gränssnitt. Appleten borde alltså få ett eget fönster, `javax.swing.JFrame`, fristående från webbläsaren även om detta inte rekommenderas av professionella gränssnittsutvecklare. Java Applets bör enligt dem alltid placeras inuti webbläsaren för att inte förväxlas med övriga applikationer på skrivbordet. Risken är att användaren stänger webbläsaren utan att tänka på att applikationen då automatiskt avslutas. Om nya fönster öppnas inifrån Appleten så följer visserligen alltid en extra

[6] http://www.manning.com

informationstext med fönstret som talar om att applikationen är en Java Applet, men denna text visas inte om användaren har godkänt en signerad Applet.

Applets har initialt deklarerade metoder som automatiskt exekveras utifrån webbläsares tillstånd. Detta gör det alltså möjligt att skriva egen programkod som exekveras när webbsidor öppnas eller lämnas. Det bör till exempel finnas stöd i Appleten för att automatiskt spara användarens öppna dokument innan denne surfar vidare. Tyvärr är det upp till varje enskild webbläsare att implementera ett fullständigt stöd för Java Applets och naturligtvis stöds inte detta till fullo i den mest förekommande webbläsaren Microsoft Internet Explorer.

Klassen `javax.swing.text.Highlighters` är speciellt konstruerad för att fungera som färgmarkörer i textdokument. Utseendet kan förhållandevis enkelt modifieras för den som till exempel vill använda understrykningar eller överstrykningar i färg. Det var ett bra alternativ för min applikation och jag studerade detta objekt mer ingående tillsammans med relaterade klasser [Loy et al., 2002]. Genom att konstruera egna klasser som ärver och utökar egenskaperna från `Highlighters` samt addera metoder för ändring av storlek, färg och form fick jag till slut rätt utseende.

4.3.4 Spara inställningar med Cookies

Den första integreringen av Lexin baserades på LiveConnect. Med hjälp av LiveConnect kunde Appleten öppna och styra webbläsaren mot Lexin och dess tjänster. Denna teknik är mycket praktiskt och användbar. Dessutom kunde Appleten nu också spara inställningar i gränssnittet med hjälp av Cookies hos användaren. Denna funktion är mycket viktig för en god inlärningsmiljö [Borin, 2003]. Dock är det viktigt att komma ihåg att användningen av Cookies klart ska framgå på webbplatsen, vad de används till och hur man kan stänga av användningen av dessa. Detta enligt lagen om elektronisk kommunikation[7] som tillträdde i kraft den 25 juli 2003.

4.3.5 Användarcentrerad utveckling

En lärare i Svenska från Stockholms universitet var inbjuden till att ge sina åsikter om Appleten i detta stadium. Några av de synpunkter som gavs var att det bör gå skriva ut kopior på skrivare ifrån Appleten. En annan att dokumentet i gränssnittet borde stödja fler teckensnitt. Den dokumentmodell som användes var helt enkelt inte konstruerad för multipla teckensnitt.

Bearbetningen av dessa synpunkter tog flera veckor eftersom stora delar av koden fick skrivas om. Att kunna skriva ut från Applets och applikationer i Java är mycket avancerat och det krävs stor kunskap om Java Swing för att resultatet ska bli bra.

När den första prototypen blivit redo att testas på lämpliga användargrupper så inbjöds lärare och studenter från Stockholms universitet till en demonstration av inlärningsmiljön. Tyvärr var det inte så många av dem som dök upp men de närvarandes åsikter var i överlag positiva.

[7] http://www.notisum.se/mp/sls/lag/20030389.htm

```
STAT 100
REPL Granska har granskat texten.
TXT< <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<Root>
  ...
  ...  // XML som representerar den analyserade texten
  ...
</Root>
```

Figur 4.1. Ett exempel på analysen från servern för Granskas TextAnalysator.

Studenterna såg genast flera möjliga tillämpningar av inlärningsmiljön. Bland annat ville de ha grammatikkontrollanten Granska implementerad. Den webbläsarbaserade lösningen av Granska ger inte så stora möjligheter till textproduktion. Den interaktiva dialogen går förlorad eftersom webbläsare inte har några mer avancerade metoder för att presentera text. Interaktionen med texten är mycket viktig och om Granska kunde implementeras skulle de kunna använda miljön till att vara en ersättare för olika ordbehandlare med språkliga stöd.

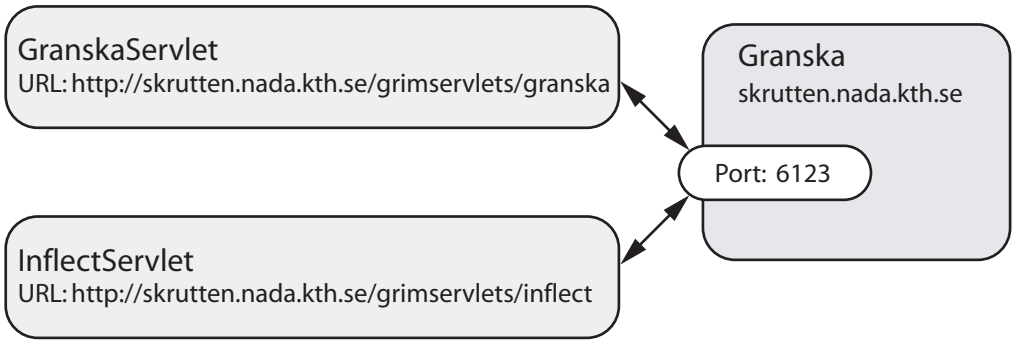
Studenternas önskan var att kunna utnyttja några externa språkrelaterade tjänster som idag finns på webben. Om dessa verktyg gick att implementera i Appleten skulle de slippa växla fram och tillbaka mellan olika programfönster. Lexikonet Lexin finns idag som en webbaserad tjänst men är aningen besvärlig att använda ihop med textproduktion. Användaren måste hela tiden manuellt skriva eller klistra in ord i ett formulär på webbsidan för att genomföra en översättning. I inlärningsmiljön skulle det däremot gå bra att använda Lexin integrerat ihop med texten då Java Swing medger en stor frihet att utveckla avancerade och interaktiva applikationer.

4.4 En robustare kommunikation

Ett av problemen med applikationen har hela tiden varit kommunikationen mot servern som sköter textanalysen. Servern kraschade ibland utan möjlighet för mig att ta del av stackdumpning för felsökning. Dessutom var kommunikationen baserad på sockets vilket i sin tur håller anslutande serverport stängd medan analys pågår. Andra klienter kan under denna tid inte ansluta eller ens ställa sig i kö. Det kan leda till stora problem för de anslutande klienterna.

Andra problem som kan uppstå är säkerhetsrelaterade. Det går stänga av servern utifrån vilket inte är acceptabelt annat än i ren testmiljö. Visserligen startas servern om automatiskt igen av ett skript men under tiden kan klienter inte ansluta. Detta problem kan lätt åtgärdas genom att integrera en brandvägg som endast släpper igenom trafik på speciella portar, förutsatt att en Java Servlet fungerar som förmedlare.

Jag började med att skapa Java Servlets som förmedlar den XML som servern för Granskas TextAnalysator genererar. Tyvärr var denna form av XML inte strikt XML



Figur 4.2. En konflikt hos Granska-servern som bör åtgärdas snarast.

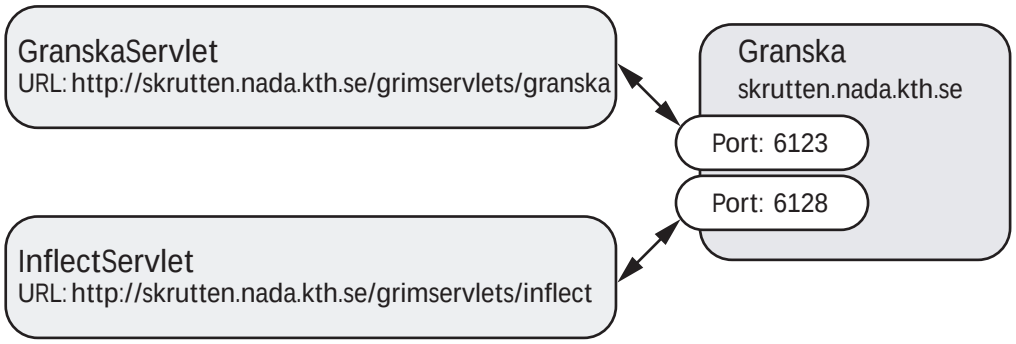
utan text kombinerat med XML, se figur 4.1. Den befintliga implementationen av XML på serversidan tvingade mig att extrahera bort den information som inte tillhör XML. Generellt kan det sägas vara de två inledande raderna plus början på den tredje som måste filtreras bort. Denna filtrering tar dock inga större systemresurser i anspråk och filtreringen sker med hjälp av reguljära uttryck.

Tanken med Java Servlets som förmedlare och filter är att ingen annan anslutning mot servern för Granskas TextAnalysator ska kunna vara möjlig. All trafik bör gå via en Servlet eftersom denna även kan fungera som passage genom brandvägg. Tyvärr är inte det övriga systemet konfigurerat så att detta är möjligt men en vidareutveckling av denna inlärningsmiljö bör absolut ta hänsyn till detta.

Den befintliga webbservern bör även integreras ihop med applikationsservern så att all kommunikation går via port 80, som är standardporten för protokollet Hypertext Transfer Protocol (HTTP). Tills vidare sker all yttre kommunikation mot alla Servlets på en speciellt avsedd port. Nackdelen med systemet såsom det är konstruerat är att brandväggen på klientens sida måste tillåta utgående trafik på den avsedda porten. På ett hemmanätverk är det inte så stora svårigheter att konfigurera routern eller brandväggen, men på skolor och företag kan detta helt säkert ställa till stora problem för presumtiva användare då systemadministratörer ogärna öppnar portar utåt. Om kommunikationen istället skulle ske genom standardporten, så behövs ingen konfiguration utföras. Tills dess den slutliga versionen av webbapplikationen presenteras ska detta problem åtgärdas.

För att åtgärda problemen med samtida anslutningar mot servern så behövs ingen större förändring göras av programkoden. I Java finns det synkroniserade metoder som garanterar tillträde för endast en anslutning i taget. Detta görs på de Servlets som ansluter mot Granska-servern med sockets. Kösystemet som buffrar upp samtliga anrop utifrån ligger under ytan och är ingenting utvecklare behöver ta del av.

Till systemet har det konstruerats nio stycken Java Servlets som de flesta kan ses som leverantörer av språkrelaterade tjänster. Servleten som sköter förmedlingen av analysen från Granskas TextAnalysator, GTAServlet, är den som sköter om



Figur 4.3. En korrekt implementation av anslutningar mot Granska-servern.

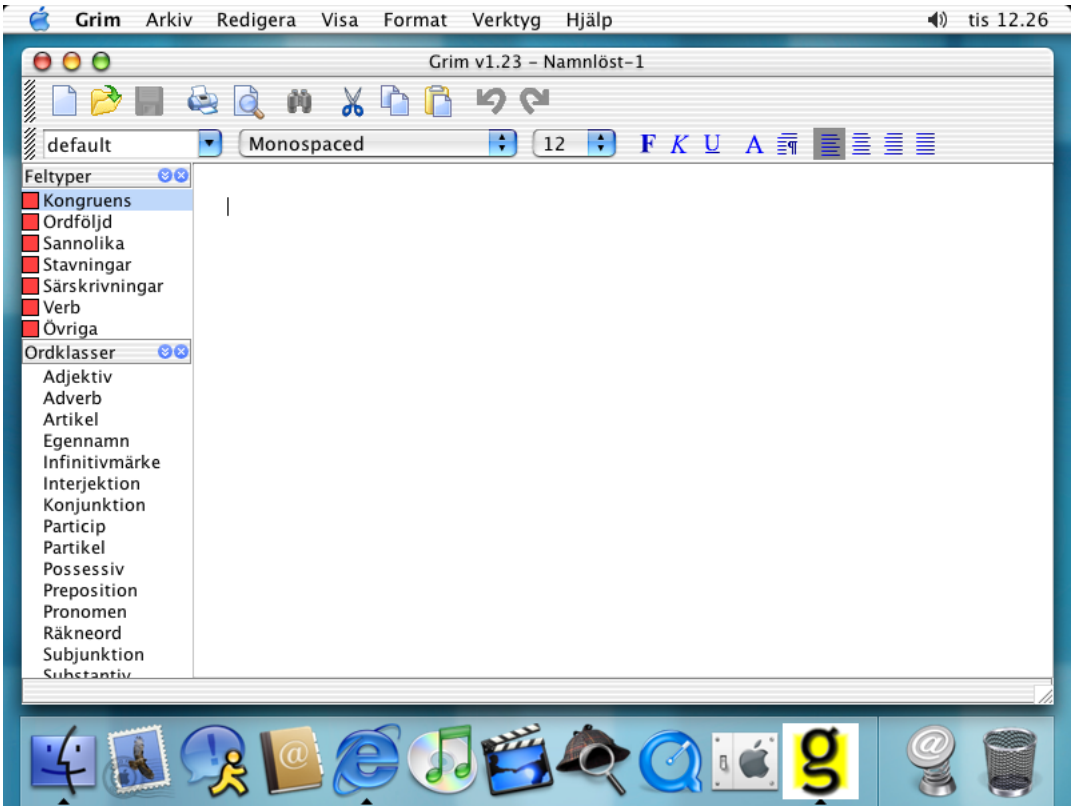
det mest grundläggande i applikationen. Denna Servlet är mycket viktig för att applikationen ska fungera. Utan den kan ingen av de övriga språktjänsterna fungera överhuvudtaget eftersom denna tar hand om positionering av enskilda ord. En annan viktig Servlet är den som sköter förmedlingen av analysen från Granska, GranskaServlet. Den är inte lika viktig för stabiliteten i systemet men är ett krav för att grammatikfunktionerna i applikationen ska fungera.

Det bör påpekas att den Servlet som ansvarar för ordböjning, InflectServlet, gör en socketanslutning mot samma port på Granska-servern som GranskaServlet, se figur 4.2. Detta kan mycket väl leda till problem när antalet samtidiga användare ökar och bör åtgärdas i samband med en vidareutveckling. Lösningen är att reservera en port för varje Servlet, se figur 4.3. Om tid finns över på slutet av detta utvecklingsarbete kommer det att åtgärdas.

4.5 Alternativ till Java Applets

Det har genom projektets gång visat sig att Java Applets inte håller måttet för mer komplexa applikationer. Egentligen är det inte något fel på Applets i sig, men de är alltid beroende av webbläsare. Som alla vet så är en kedja inte starkare än sin svagaste länk och för det mesta är det webbläsaren som kraschar innan Applets gör det. Oavsett val av webbläsare så var stabiliteten i dessa så dåliga att jag inte såg Java Applets som ett försvarbart alternativ. Det är absolut inte roligt att förlora sitt arbete framför datorn.

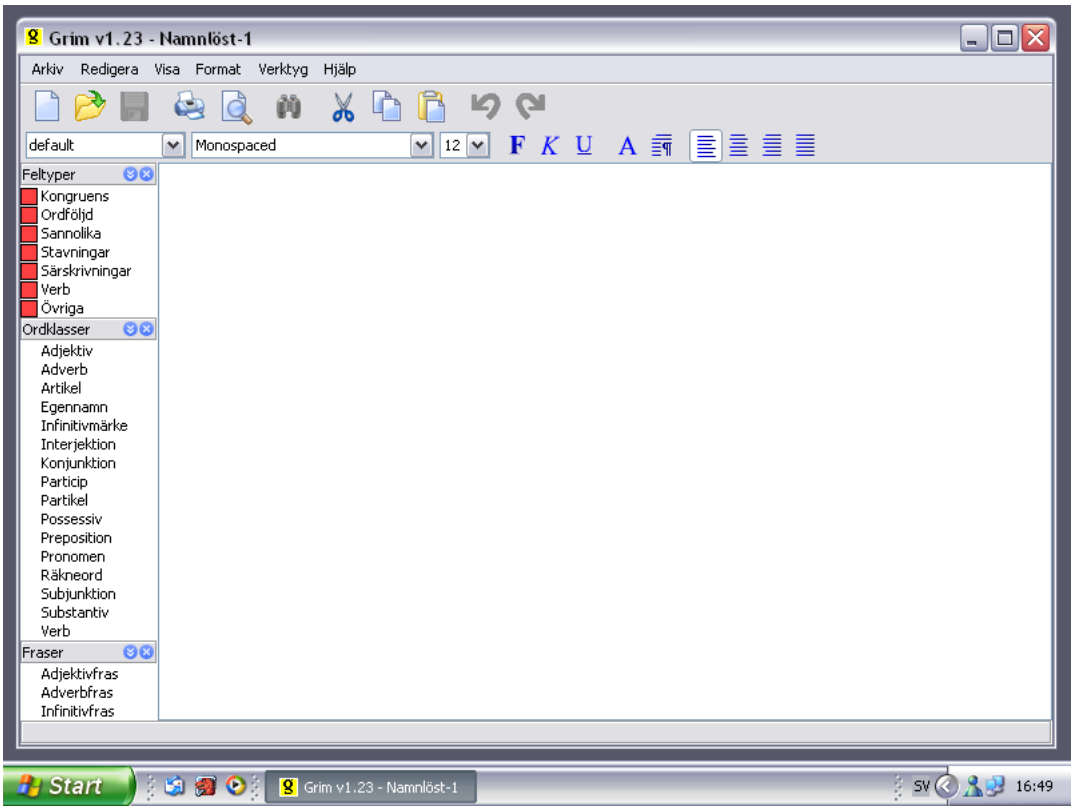
Jag letade efter andra Java-relaterade tekniker för webben och hittade Java Web Start. Den mängd kod jag behövde modifiera för att fungera ihop med Java Web Start var överkomlig och arbetet inleddes omedelbart. Resultatet gav en entydig bild. Med Java Web Start har stabiliteten i systemets klart förbättrats. Fördelen är att applikationen exekveras fristående från webbläsare men har forfarande möjligheten att kommunicera med dessa. Applikationen kan alltså startas från webbläsaren men körs i en egen miljö vilket minskar risken för krasch. Det går också bra att öppna



Figur 4.4. Standardplacering av programmenyer på operativsystemet Mac OS X är i överkant av skrivbordet.

standardwebbläsaren ifrån webbapplikationen med hjälp av Java Network Language Protocol (JNLP).

Det negativa när det gäller Java Web Start är att denna teknik inte behandlar certifikat på samma sätt som webbläsare gör. Det finns inget alternativ till att neka åtkomst till datorns resurser men fortfarande starta applikationen. Alltså blir användaren tvungen att lita på utvecklaren och godkänna certifikatet. Det var en annan av de önskningar som jag ansåg viktiga i projektet. Tyvärr kan det inte uppfyllas med nuvarande teknik. Beteendet är inbyggt i Java Web Start och kan inte påverkas av mjukvaran. Att det förhåller sig på detta sätt är säkerligen att Java Web Start har möjlighet att via JNLP få tillgång till systemresurser utan att applikationen behöver vara signerad. Tyvärr har detta system inte en tillräckligt intuitiv dialog med användaren. Det finns exempelvis ingen möjlighet att öppna filer med direktlänkar. Inte heller någon möjlighet att spara filer utan filväljare. Java Web Start är en förhållandevis ny teknik så det är inte omöjligt att utvecklingen av JNLP blir mer användarcentrerad.



Figur 4.5. Standardplacering av programmenyer på bland annat operativsystemet Microsoft Windows XP är inuti det egna programfönstret.

4.6 Modifieringar för Mac OS X

Det visade sig att applikationen måste anpassas till olika operativsystem, i synnerhet Mac OS X. Utseendet och beteendet hos Mac OS X är annorlunda är hos exempelvis det vanligast förekommande operativsystemet Microsoft Windows XP. Mac OS X har alltid sin huvudmeny i överkant av skrivbordet, se figur 4.4, och inte inuti programfönstret som hos Windows XP, se figur 4.5. För jämförelsens skull har de båda skärmdumparna en upplösning på $800 * 600$ pixlar.

Den kod som modifierar gränssnittet för Mac OS X måste kompileras ihop med övriga delar i applikationen. Men de klasser och gränssnitt som används av Mac OS X finns inte tillgängliga för övriga operativsystem. Därför måste temporära klasser konstrueras som efterliknar dessa klasser och gränssnitt för att fungera ihop vid kompileringen. Det är bara en av de kompilerade klasserna, `SpecialMacHandler`, som sedan följer med den färdiga webbapplikationen.

Med hjälp av Java Reflection kan existerande originalklasser laddas dynamiskt under programexekvering om webbgränssnittet används på Mac OS X. Något anrop av klasserna sker inte på övriga operativsystem.

Det bör påpekas att metoden för att ladda klasser dynamiskt med Java Reflection i det här fallet möjligen inte är befogad eftersom Mac OS X ska garantera att Macs originalklasser har högre prioritet än de i projektet medföljande klasserna. Med dessa fakta skulle det alltså gå att distribuera samtliga av dessa klasser i projektet. Det har dock inte gått att verifiera dessa uppgifter i tekniska rapporter från Apple.

Det har med tiden visat sig att det är komplicerat att få webbapplikationer att utseende- och känslomässigt fungera på så sätt att samtliga användare finner sig tillrätta. När webbapplikationen startar upp kontrolleras vilket operativsystem som används och extra funktioner tar hand om den dynamiska klassladdningen när Mac OS X används. För övriga operativsystem behövs ingen modifiering av källkod.

4.7 Paketering och distribuering av källkod

Verktyget Ant, Javas motsvarighet till `make` för C, implementerades i projektet och det gör att det går att bygga bytekoden och tillhörande dokument direkt från källkoden med ett enda kommando. För eventuell vidareutvecklare och inte minst mig själv är det till stor nytta och lättnad att detta har adderats till projektet. Alltför mycket tid har gått åt till att skapa rätt miljövariabler som till exempel `CLASSPATH` och `JAVA_HOME`. Det har också visat sig genom projektets gång att Ant är enkelt att konfigurera och installera.

Med hjälp av Ant har genereringen av de klasser som beskriver XML-strukturen av analysen från Granska- och GTA-servern kunnat ske automatiskt. De genererade klasserna kompileras sedan ihop med övriga klasser i projektet. Dock har en uppdelning skett mellan webbapplikationens klasser och de genererade klasserna. Detta på grund av att vid uppdatering av en klass ur det ena paketet inte ska kräva nedladdning av de i sådana fall sammanfogade paketen.

Totalt sett distribueras webbapplikationen i sex stycken olika JAR-arkiv. Två av dessa är till för XML, `castor.jar` och `xercesImpl.jar`, och två andra för webbapplikationens gränssnitt, `grim.jar`, och kommunikation, `grimx.jar`. Ytterligare två paket tar hand om utseende och känsla, så kallad *Look&Feel*, samt layouthanterare. Båda dessa är produkter från JGoodies[8] och bygger på öppen källkod.

[8] http://www.jgoodies.com

Kapitel 5

Resultat

5.1 Installation

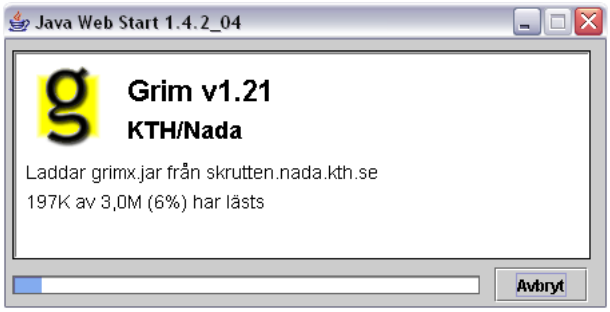
För installationen av webbapplikationen krävs inte mer än två-tre musklick eftersom Java Web Start sköter om denna procedur automatiskt. Java Web Start sköter även om uppdateringar av applikationen om en ny version finns tillgänglig på webbservern.

När användaren besöker projektets hemsida på webben[1] finns en länk till en installationsanvisning där samtliga steg i processen redovisas. Då användaren klickar på länken till JNLP-dokumentet startas Java Web Start, se figur 5.1, om webbläsaren är rätt konfigurerad. Java Web Start i sin tur läser innehållet i denna fil och hämtar bland annat JAR-arkiven, se figur 5.2, som utgör själva applikationen. Webbapplikationens olika delar består tillsammans av runt 3,5 megabyte programkod. Beroende på uppkoppling tar tiden för att ladda hem delarna allt från ett par sekunder till några minuter.

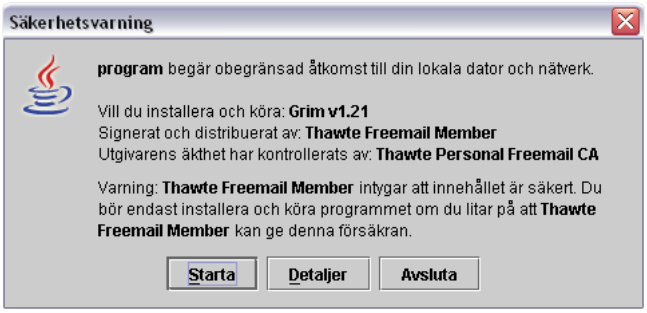
Figur 5.1. Java Web Start startar automatiskt när användaren klickar på länken till JNLP-dokumentet på webbplatsen.

Då hämtningsprocessen är avslutad och webbapplikationen installerad frågar det inbyggda säkerhetssystemet i Java Web Start om användaren vill godkänna applikationens åtkomst till datorns systemresurser, se figur 5.3. Detta eftersom JAR-arkiven är signerade och JNLP-dokumentet uttryckligen efterfrågar full åtkomst. Det går tyvärr inte att neka webbapplikationen åtkomst till systemresurserna och ändå starta applikationen. Det går dock bra att studera det medföljande certifikatet som utgör själva signeringen och själv avgöra om certifikatet kan anses tillförlitligt.

[1] http://skrutten.nada.kth.se/grim



Figur 5.2. Java Web Start hämtar och installerar webbapplikationen.

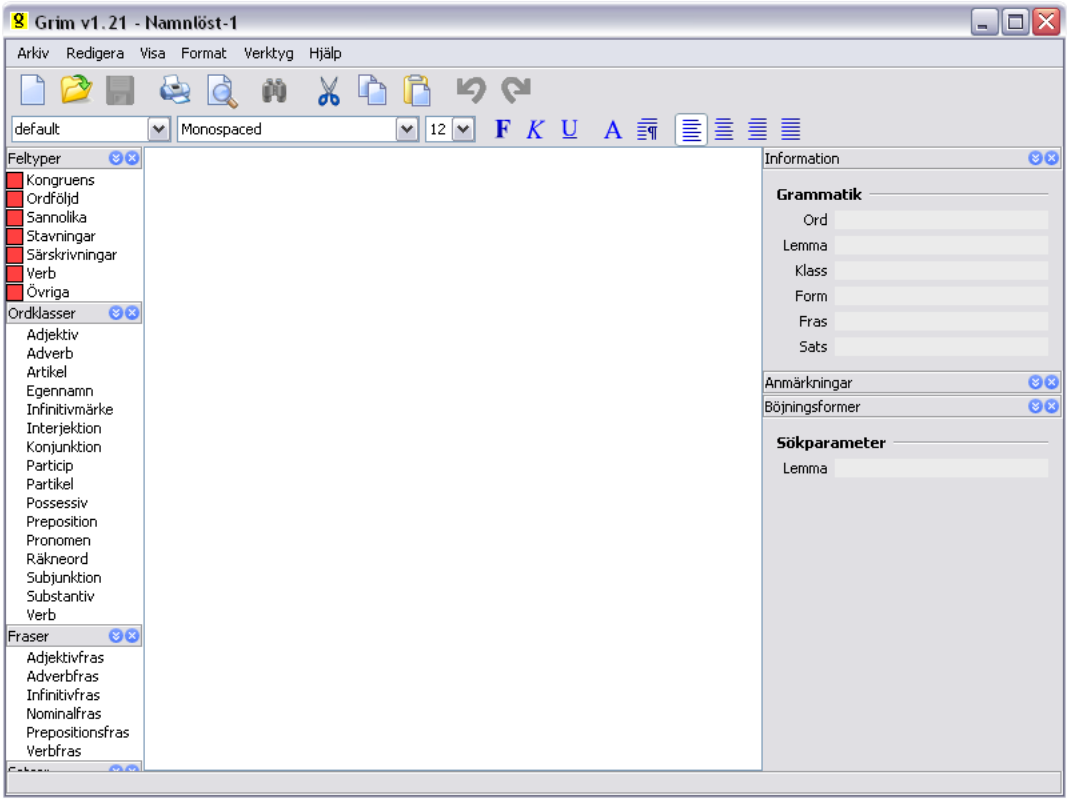


Figur 5.3. En i Java Web Start inbyggd säkerhetsdialog gör att användaren måste godkänna webbapplikationens åtkomst till systemresurser.

Figur 5.4. Java Web Start frågar användaren om denne vill installera en snabblänk till webbapplikationen (endast på Microsoft Windows och Mac OS X).

När användaren godkänt certifikatet och applikationen fått tillgång till systemresurserna frågar Java Web Start om användaren vill ha en snabblänk till webbapplikationen på skrivbordet, se figur 5.4. Observera att detta endast sker på Mac OS X och Windows än så länge. På Mac brukar dock frågan komma andra gången applikationen startas. Snabblänken gör det enkelt för användaren att starta applikationen utan att behöva gå omvägen genom webbläsaren.

Det bör tilläggas att det går att starta webbapplikationen direkt från programhanteraren för Java Web Start som bör finnas på skrivbordet efter installationen



Figur 5.5. Webbapplikationen startar med alla verktyg öppna som standard.

av Java. Däremot kan det vara nödvändigt att skapa denna snabblänk manuellt på operativsystemen Linux och Solaris. På framtida versioner av Java Web Start är detta något som kommer att förbättras då många utvecklare har efterfrågat detta.

Efter detta startar applikationen nu i sitt utgångsläge där verktygen är påslagna, se figur 5.5. De vanliga verktygen som finns i de flesta ordbehandlare är placerade där användaren förväntar sig dem. På detta förfaringssätt slipper användaren koncentrera sig på att lära sig det generella gränssnittet och kan istället direkt försöka lära sig de inbyggda språkverktygen.

5.2 Gränssnittet

5.2.1 Ordbehandlingsmiljö

För att användaren snabbt ska kunna ta till sig information från webbapplikationen har alla verktyg bäddats in en ordbehandlingsmiljö. Allt för att göra tröskeln för inlärning så låg som möjligt. Gränssnittet är till viss del adaptivt då användarens personliga inställningar lagras i användarens hemkatalog och används till följande gång. Dessutom lagras även de senast använda filerna och applikationens unika

identifikationskod som används av servrarna på KTH Nada för att registrera viss användarinteraktion. Det bör tilläggas att identifikationskoden inte kan användas för att identifiera den unika användaren, vilket skulle kunna vara oetiskt. Syftet är att kunna avläsa vilka verktyg som till exempel är mest populära och vilka som det eventuellt skulle vara värt att vidareutveckla.

5.3 Språkverktyg

Språkverktygen ska inte ses ur tekniskt perspektiv utan ur användarens perspektiv. Granska och Granskas TextAnalysator är helt eller delvis implementerade i applikationens språkverktyg. Anledningen till denna uppdelning är att verktygen har olika roller. Några av verktygen har passiva roller som exempelvis vill uppmärksamma inläraren att något är fel i texten, andra verktyg har aktivare roller som till exempel att komma med information angående felet samt eventuella rättningsförslag.

Webbapplikationens språkverktyg kan delas upp i tre kategorier. I den första kategorin finns de passiva verktyg som markerar fel och klasser med hjälp av färger. De befinner sig på vänsterkanten av gränssnittet. I andra kategorin de verktyg som är externa som inte är integrerade med applikationens huvudgränssnitt och som följdaktligen har egna fönster. Den tredje kategorin består till största delen av aktiva språkverktyg som är integrerade med applikationens huvudgränssnitt i högerkant.

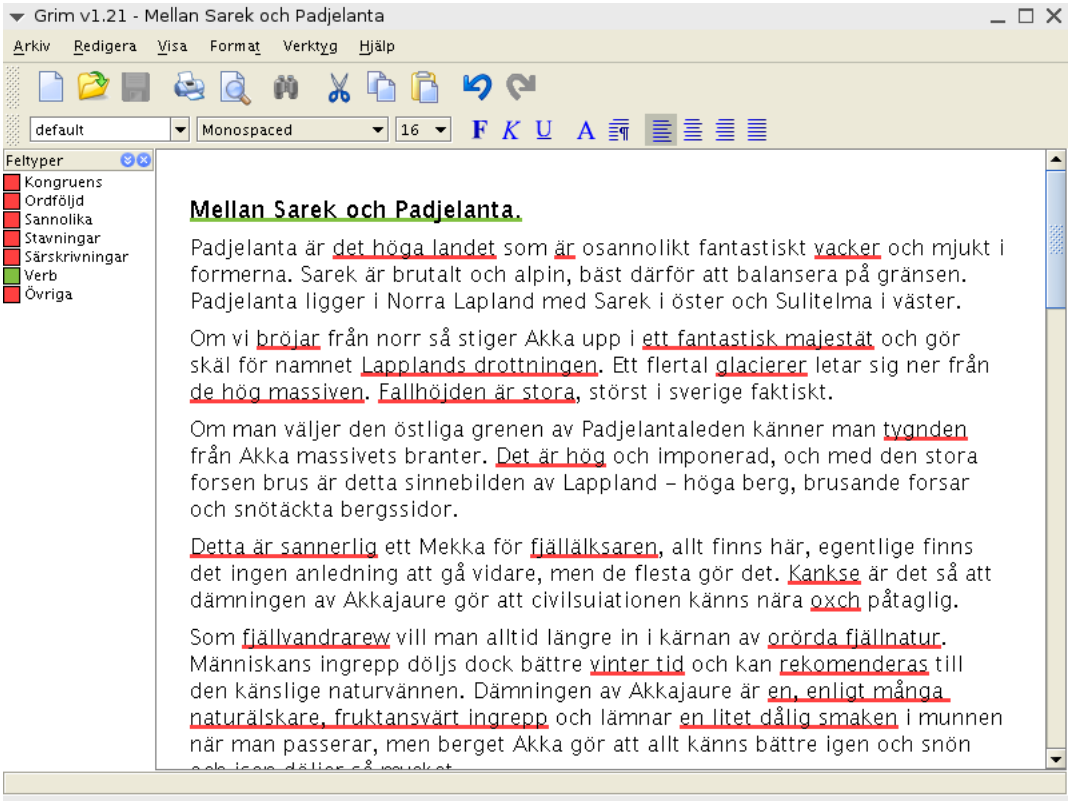
5.3.1 Färgmarkeringsverktygen

För att markera feltyper, ordklasser, frasklasser och satser används listor bestående av de olika kategorier dessa kan delas in i. Varje element i listorna visar kategorinamn samt tillhörande färg, se figur 5.6, som används för att markera ord i dokumentet. Med ett höger- eller dubbelklick aktiveras en färgväljardialog som enkelt låter användaren byta eller ta bort färg. Det går också alldeles utmärkt att samtidigt markera flera kategorier för färgändring.

Ändringar av färg avspeglar sig direkt i dokumentet. Verktygen kan minimeras och maximeras genom att fälla upp respektive fälla ner listan från den interna ramen. Det går också bra att stänga av verktyget från ramen. Detta är min egen design för att spara utrymme på skrivbordet då många språkverktyg används parallellt. Alternativet hade varit traditionella vertikala rullningslistor, men dessa tar enligt min mening för mycket utrymme i horisontell riktning.

5.3.2 Information

Den första och kanske mest grundläggande av språkverktygen är verktyget Information. Till varje analyserat ord i dokumenttexten hör grammatisk information och verktyget visar i tur och ordning ord, lemma, klass, form, fras och sats. Det finns även en återkoppling mot dokumentet i form av att ursprungsordet (eller omslutande fraser och satser) i dokumenttexten markeras med bakgrundsfärg då muspekaren förs över värdena i verktygsfältet, se figur 5.7.



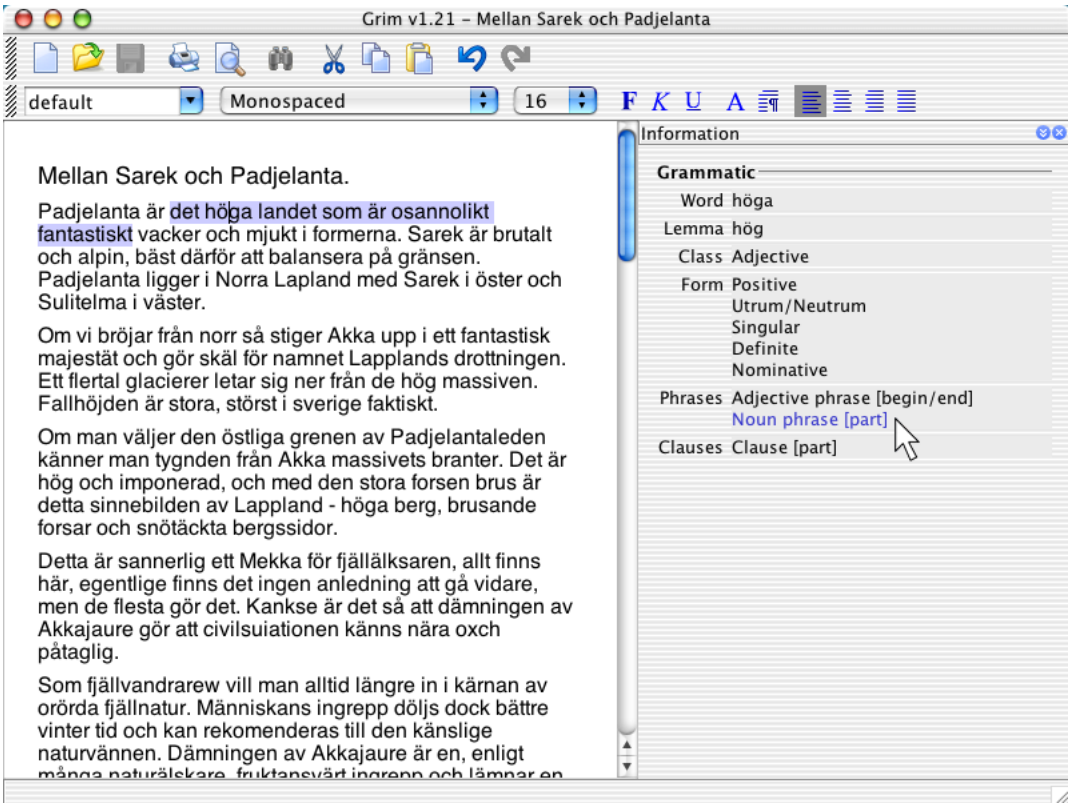
Figur 5.6. Verktyget ”Markera feltyper” har en direkt återkoppling mot texten i dokumentet. Här ett exempel på FreeBSD med Gnome som fönsterhanterare.

5.3.3 Anmärkningar

Det andra verktyget är Anmärkningar som är en direkt koppling mot språkgranskningsverktygen Granska och ProbGranska. Då användaren placerar markören över ett analyserat ord som ingår i ett språkfel visas vad som hittats, information om vad som är fel samt eventuella rättningsförslag. Inte allt för sällan ingår ordet i två eller flera språkfel och samtliga radas då upp för användaren. Användaren kan dubbelklicka på ett rättningsförslag för att ersätta den felaktiga texten i dokumentet. Det finns också en återkoppling mot språkfelet i dokumenttexten som markeras med bakgrundsfärg då muspekaren förs över värdena i verktygsfältet, se figur 5.8.

5.3.4 Böjningsformer

Det tredje och sist inbyggda språkverktyget är Böjningsformer och det är en koppling mot ordböjningsverktyget Inflector som är utvecklat vid KTH Nada. Då användaren placerar markören över ett analyserat ord i dokumenttexten visas de ordformer som ordets lemma kan böjas till. En återkoppling mot dokumentet finns i form av att ursprungsordet i dokumentet markeras med bakgrundsfärg när muspekaren förs över



Figur 5.7. Verktyget ”Information” har en återkoppling mot texten i dokumentet. Här ett exempel på Mac OS X med engelsk språkinställning.

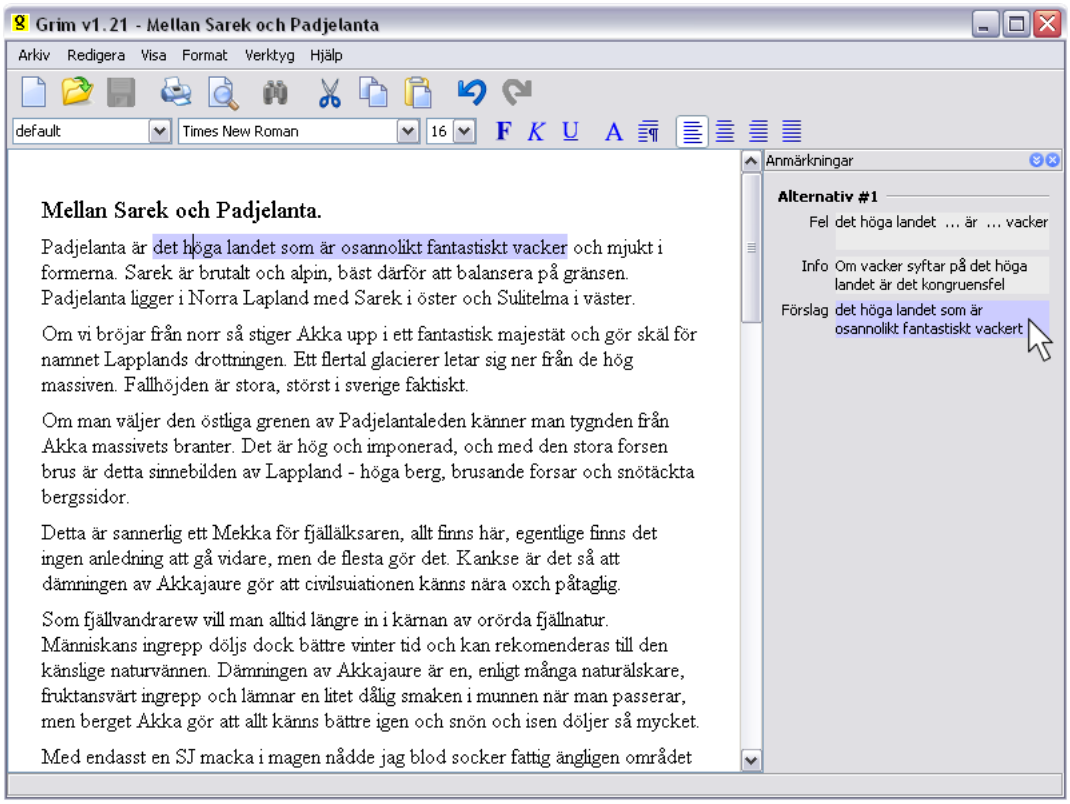
alternativen i verktygsfältet. Användaren kan också dubbelklicka på ett alternativ för att få böjningsformen insatt i dokumentet istället för ursprungsordet. I stort sett fungerar det på samma sätt som för verktyget Anmärkningar.

5.3.5 PAROLE

På Språkdata vid Göteborgs universitet har den ordklasstaggade korpusen PAROLE utvecklats. PAROLE är ett EU-projekt som avslutades år 1997 och inriktade sig på att ordklasstagga textmaterial med totalt 19 miljoner ord. Syftet var att bygga upp ett europeiskt nätverk av språkliga resurser. För att allmänheten skulle kunna ta del av informationen från denna databas konstruerades ett webbgränssnitt där användaren skulle kunna göra sökningar efter hur ett visst ord används i det svenska språket.

Tyvärr har webbgränssnittet[2] en del funktionella brister och det inbjuder inte till längre tids användning. Det går exempelvis inte att inkludera alla böjningsformer av ett visst ord. Verktyget ligger dessutom alldeles för långt ifrån skrivprocessen.

[2] http://spraakbanken.gu.se/lb/parole

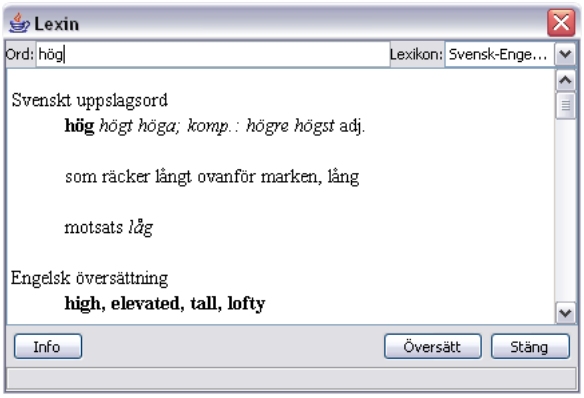


Figur 5.8. Verktyget ”Anmärkningar” har en direkt återkoppling mot texten i dokumentet. Användaren kan dubbelklicka på förslaget för att ersätta felet.

Den som skriver en egen text och vill veta hur ett visst ord används ska inte behöva lära sig på krångliga webbgränssnitt.

Verktyget PAROLE i webbapplikationen är ett enkelt utformat gränssnitt mot konkordansdatabasen PAROLE. Om användaren vill undersöka hur ett visst ord används i ett sammanhang i det svenska språket är detta verktyg utmärkt. Användaren inte göra mer än att klicka på en menypost, alternativt använda en snabblänk som tangentkombinationen *Ctrl-K*, vilket automatiskt aktiverar sökningen för det ord som markören befinner sig på.

Genom att ändra i inställningarna för PAROLE kan samtliga böjningsformer av ordet inkluderas. Inställningarna kan också användas för att matcha versaler och gemener. Precis som i webbgränssnittet kan också bland annat kontextlängd, träfflängd och sorteringsordning ställas in efter användarens önskemål.



Figur 5.9. Verktyget ”Lexin” översätter ord till och från åtta olika språk.

5.3.6 Lexin

Ett annat verktyg är det välkända verktyget Lexin som i ett flertal år har funnits tillgängligt på webben[3]. Lexin har idag ett webbgränssnitt som låter användaren fylla i ord för översättning efter val av lexikon. Denna process kan vara lite för omständig om användaren exempelvis försöker läsa och förstå en text, eftersom webbgränssnittet inte är speciellt interaktivt. Med integreringen av Lexin i webbapplikationen har verktyget nått lite närmare användaren då denne inte manuellt behöver fylla i sökparametrar.

Detta lexikonverktyg kommer nog till sin fulla rätt om det blir ett internt verktyg i webbapplikationen men på grund av tidsbrist har det endast blivit ett externt. Som internt skulle det kunna fungera mycket bra som stöd av läsning för exempelvis andraspråksinlärning då översättning av ord skulle kunna ske automatiskt då användaren klickar på ett ord i dokumentet. En förutsättning för detta är dock att en ny teknisk lösning installeras på serversidan. Den nuvarande servertekniken är på tok för långsam för att kunna vara ett bra översättningsverktyg vid läsning.

Verktyget Lexin i webbapplikationen är ett enkelt utformat gränssnitt, se figur 5.9, mot Lexins webbserver. Webbservern använder sig av ett skript skrivet som Common Gateway Interface (CGI). Användaren kan placera markören på ett ord i dokumenttexten och klicka på menyposten Lexin, alternativt använda en snabblänk som tangentkombinationen *Ctrl-L*, vilket automatiskt aktiverar översättningen. En relativt viktig detalj är att Lexin kan ge länkar till beskrivande bilder på webben om orden är okända för andraspråksinläraren. Ett klick på en sådan länk aktiverar standardwebbläsaren som visar denna bild.

[3] http://www-lexikon.nada.kth.se/skolverket/lexin.shtml

5.3.7 SweSum

Sist men inte minst finns det nyligen utvecklade verktyget SweSum som är en textsammanfattare. Användaren kan korta ner texten i sitt dokument automatiskt och ändå behålla det viktigaste innehållet. Med hjälp av nyckelord och olika lingvistiska, statistiska samt heuristiska metoder kan detta åstadkommas. Till exempel kan en rubrik vara mer värd en ett stycke och ett ord som förekommer tidigt i texten vara mer värt en ett annat ord som förekommer senare. På så sätt kan meningar och stycken utelämnas från texten. SweSum är ett intressant språkverktyg men lider för närvarande av att analyserna tar för lång tid för att kunna användas i en produktiv skrivmiljö. Anledningen att SweSum är integrerat i webbapplikationen är att se om användare finner verktyget intressant.

5.4 Servlets

På serversidan av applikationen finns nu nio stycken Java Servlets, se figur 5.10. Dessa Servlets körs på Servlet-motorn Apache Jakarta Tomcat. Denna är integrerad ihop med en vanlig webbserver från Apache och körs därför på standardporten för HTTP. Med denna teknik kan alla som kan få åtkomst till webbapplikationens hemsida, med exempelvis webbläsare, också använda sig av webbapplikationen eftersom all kommunikation går via denna standardport.

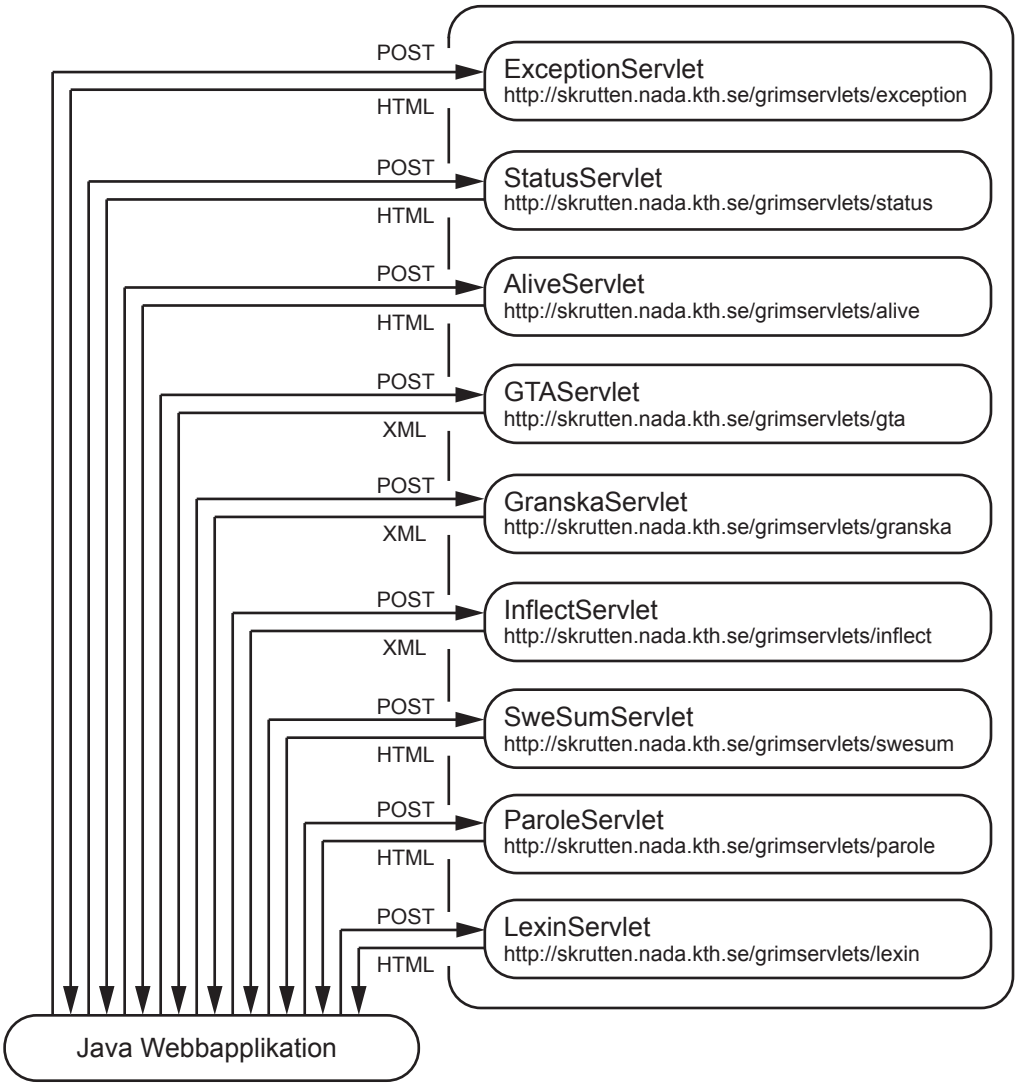
Senare versioner av Tomcat har en mycket trevlig egenskap och det är utnyttjandet av namngivningstjänsten Java Name and Directory Interface (JNDI). Med hjälp av JNDI kan kommunikationen mellan Servlets och databas konfigureras externt i speciellt avsedda filer. Det gör att programkoden inte behöver kompileras om då ändringar måste ske i exempelvis nätverket. När sedan varje Servlet startar upp så hämtas dessa konfigurerade värden med hjälp av klassen `javax.naming.InitialContext`.

För kommunikationen mot databasen MySQL använder sig dessa Servlets av en anslutningsförmedlare, en *ConnectionPool*, som håller kommunikationen öppen, se figur 5.11. Det gör att inloggning mot databasen bara sker en gång istället för vid varje förfrågan och det sparar massor av tid eftersom just autentiseringen tar det mesta av tiden. Anslutningsförmedlaren konfigureras med hjälp av JNDI utanför själva källkoden och kan alltså ändras utan omkompilering.

5.4.1 GranskaServlet

När applikationen anropar GranskaServlet sker en vidarebefodring av det modifierade anropet. Servleten fungerar alltså som ett gränssnitt mellan dessa två olika system. Granska-servern kommunicerar via en socket på en speciell port[4] som inte är trådad. Utan trådar kan dock bara en förfrågan i taget expedieras. Detta innebar att ett kösystem var tvunget att implementeras för att alla parallella anrop ska kunna besvara i tur och ordning, se figur 5.12.

[4]Port nummer 6123

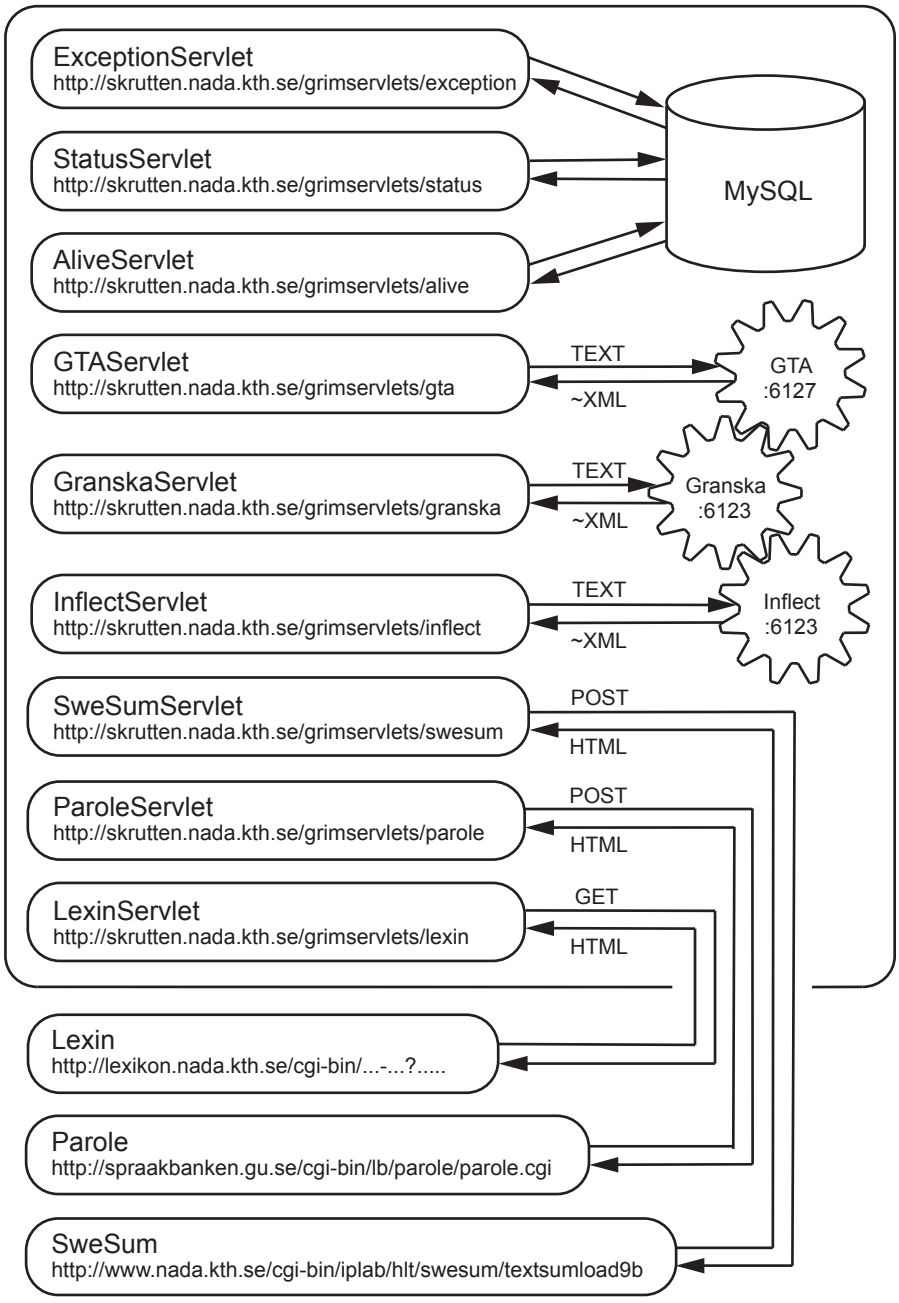


Figur 5.10. Webbapplikationens kommunikation mot alla Servlets.

När Granska-servern analyserat texten skickas resultatet tillbaka till Servleten som i sin tur klipper bort statusmeddelanden i början av analysen. Resterande del är XML som sänds tilbaka till webbapplikationen.

5.4.2 GTAServlet

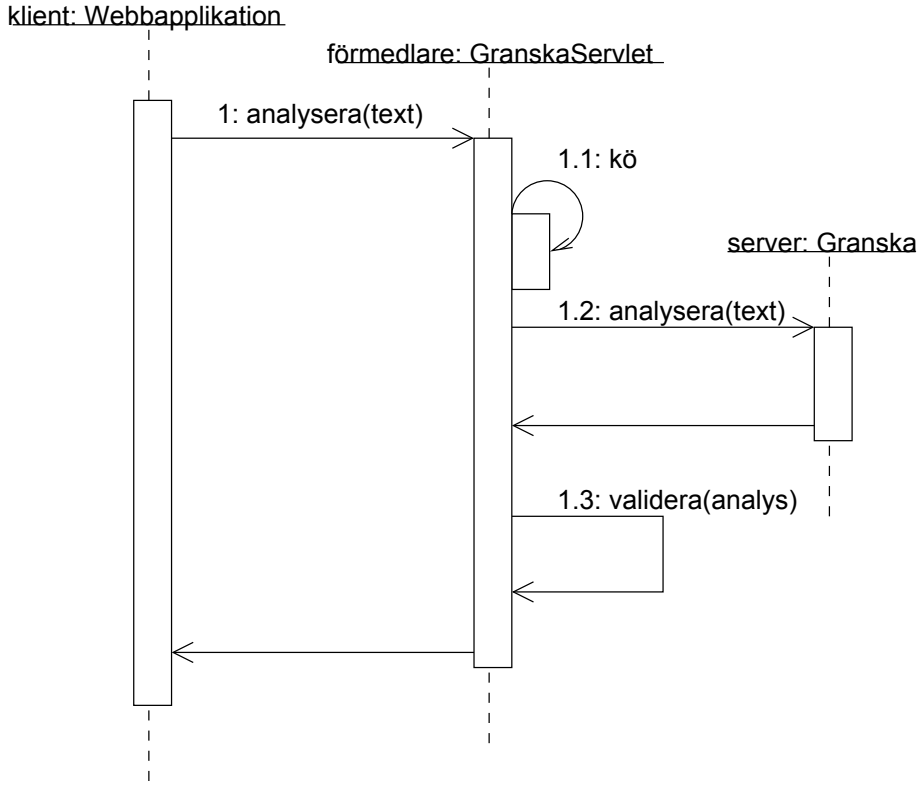
För att analysera texten syntaktiskt används GTAServlet som mellanhand på samma sätt som GranskaServlet förmedlar den grammatiska analysen av texten. Den enda skillnaden mellan dessa två Servlets är att den XML de genererar är olika, om än



Figur 5.11. Servletarnas kommunikation mot bland annat GTA och Granska.

med mycket liten skillnad, och att kommunikationen sker på en annan port[5].

[5]Port nummer 6127



Figur 5.12. Analyssekvensen mellan server och klient med inbyggt kösystem.

5.4.3 InflectServlet

För den integrerade ordböjaren i webbapplikationen används InflectServlet som i sin tur kommunicerar mot Granska-servern. Analysen från Granska-servern representeras i form av radbaserad samt oformaterad text och måste transformeras till XML för att smidigt passa ihop med webbapplikationen. Eftersom analysen sker med hjälp av Granska-servern kan en konflikt uppträda om både InflectServlet och GranskaServlet samtidigt försöker kommunicera med Granska-servern.

5.4.4 ParoleServlet

Denna Servlet imiterar Språkdatas webbgränssnitt för PAROLE och adderar eventuellt utelämnade parametrar från webbapplikationen. Webbapplikationen skickar med sökparametrar från det egna gränssnittet för PAROLE. Servleten kontaktar

alltså PAROLE:s gränssnitt på webben och filtrerar bort den information som är överflödig, det vill säga allt utom själva sökresultatet i HTML. Tillstånd till denna typ av användning har godkänts av Språkdata vid Göteborgs universitet.

5.4.5 LexinServlet

Denna Servlet kontaktar webbgränssnittet för Lexin med speciellt avsedda parametrar. Servleten agerar en mellanhand som förmedlar sökresultatet till webbapplikationen. Dessförinnan filtreras överflödig information bort så att endast det väsentliga når mottagaren. Den överflödiga informationen är allt utom de HTML-taggar som utgör själva sökresultatet.

5.4.6 SweSumServlet

SweSum är en textsammanfattare som utvecklats på KTH Nada och har ett enkelt gränssnitt som kan nås från en vanlig webbläsare. Denna servlet kommunicerar med det CGI-skript som finns på webbservern och tar emot förfrågningarna. Sedan skickas resultatet från analysen tillbaka till webbapplikationen via Servleten utan filtrering.

5.4.7 ExceptionServlet

För framtida utveckling och tillvaratagande av felinformation från webbapplikationen i de miljöer där den förekommer, alltså hos användarna, har en speciell Servlet konstruerats. Den är ständigt beredd att lagra felinformation, eller rättare sagt `java.lang.Exception`, från webbapplikationen i en databas.

För varje kritiskt undantag som kastas i Java Virtual Machine (JVM), samt fångas av webbapplikationen, kontaktas denna ExceptionServlet. Användaren får dock först en fråga om denne vill hjälpa KTH Nada att förbättra programkoden genom att dela med sig av felinformationen till KTH Nadas servrar. Felinformationen kompletteras med systemegenskaperna, `System.getProperties()`, som gör det möjligt att till exempel identifiera vilket operativsystem som använts då undantaget i programkoden inträffade. En variant av detta felrapporteringssystem används bland annat av Microsoft.

Det bör tilläggas att felinformationen är filtrerad från varje webbapplikation. Användarspecifik information som till exempel användarnamn är inte viktiga för korrigeringen av koden och därför har denna utelämnats. Dessutom kan det vara oetiskt att lagra sådan information i databasen.

5.4.8 AliveServlet och StatusServlet

Webbapplikationen har en inbyggd timer som varje minut skickar ett anrop, likt ett hjärtslag, till AliveServlet. Databasen MySQL används för lagring av denna information. Med ett webbgränssnitt kan sedan antalet aktiva webbapplikationer studeras närmare och tanken bakom detta är att få en bild över användandet av webbapplikationen.

AliveServlet och StatusServlet använder sig av en gemensam tabell i databasen där informationen uppdateras löpande. Om inget hjärtslag inkommer på fem minuter antas webbapplikationen ha stängts ner utan möjlighet för den att kontakta Servleten. Detta kan inträffa om användaren stänger av sitt modem eller om webbapplikationen mot förmodan hänger sig.

StatusServlet lyssnar på användarinteraktionen från webbapplikationens gränssnitt och loggar denna information i databasen. Interaktionen kan till exempel vara *Idle*, då applikationen inte har något dokument framme, och det kan också vara *Edit*, då användaren skriver i sitt dokument.

5.5 Kända buggar

Genom projektets gång har en prioritering av funktionaliteten i webbapplikationen gjorts och det mest väsentliga har åtgärdats allteftersom. Tyvärr finns det andra delar av webbapplikationen som inte fungerar lika bra på grund av tidsbrist.

5.5.1 Funktionerna Ångra och Gör om

Två funktioner som inte är fullt åtgärdade är funktionerna Ångra och Gör om. På Mac OS X fungerar dessa funktioner som standard på de sätt som användaren förutsätter. Men på övriga plattformar har den interna hanteringen av funktionerna ett annorlunda beteende.

Det är nämligen så att alla ändringar av attribut på texten såsom storlek, typsnitt och stil adderas till objektet `javax.swing.undo.UndoManager`, som gör att ändringar i dokumentet kan ångras respektive göras om. På alla andra plattformar än Mac OS X gäller tyvärr detta även de attribut som inte kan visualiseras.

I den modell jag använt för att lagra attribut som ordklasser och grammatikfel har jag adderat dessa attribut på både teckennivå och styckesnivå. Enkelt uttryckt har jag spännt ett nät av olika attribut över varje stycke. Tyvärr läggs denna ändring av dokumentet även till i `UndoManager`, vilket gör att funktionen ångra inte längre ändrar användarens sista hantering, utan applikationens. Detta beteende förvärras av att attributen som adderas är mycket stort till antalet och funktionerna Ångra och Gör om blir därigenom i stort sett oanvändbara. Egentligen är enda anledningen till att de existerar att det fungerar utmärkt på Mac OS X.

5.5.2 Funktionen Utjämnad

Att kunna få texten i sitt dokument i vänsterkant, centrerad, högerkant eller utjämnad är i stort sett standard på samtliga ordbehandlare på marknaden. Tyvärr finns det en bugg i funktionen för att få text utjämnad sedan fem år tillbaka i Java. Trots att över 250 utvecklare har denna bugg[6] som prioriterad i Suns BugParade har inget gjort åt saken ännu.

[6] http://developer.java.sun.com/developer/bugParade/bugs/4263904.html

Som tur är finns det en metod att komma förbi det här problemet, men det är alldeles för stort ingrepp för att hinnas med i examensarbetet. I en framtida version kan dock denna information[7] vara till stor hjälp.

5.5.3 XML

Den XML som idag genereras i Granska och Granskas TextAnalysator är felaktig och måste rättas till eftersom applikationen inte klarar av att parsa den genererade XML-strukturen i vissa fall. Ett tecken som skickas in till servern måste kodas om vid utmatning ifall det är ett av XML reserverat tecken. Det är ju inte speciellt ovanligt att en del av dessa tecken ingår i svensk text. Parsern klarar dock av de flesta av dem med hjälp av självkorrigering men absolut inte tecknena `&` och `<`. De reserverade tecknen redovisas i tabell 5.13.

Tecken	Förklaring	Entitetsreferens
<	Mindre än	`\<`
>	Större än	`\>`
&	Och	`\&`
"	Citation	`\&quot;`
’	Apostrof	`\&apos;`

Figur 5.13. Tabell som visar tecken och dess motsvarande entitetsreferenser.

5.6 Övrigt

5.6.1 Forum

För att stödja användarna har ett forum[8] skapats där frågor och funderingar kring inlärningsmiljön får plats. Detta forum kan användarna nå via en länk på projektets webbplats. Användarna måste registrera sig för att kunna skriva egna inlägg på grund av att risken är hög för att diskutionen annars spårar ur. Alla kan dock läsa samtliga inlägg utan att vara inloggade i systemet. Forumet är baserat på ett projekt[9] och använder sig av teknikerna PHP och MySQL. Den enda som behövdes var att konfigurera forumet och MySQL-servern.

5.6.2 Gästbok

En enkel gästbok har skapats baserad på PHP och MySQL som möjliggör kortare meddelanden från webbplatsens besökare. Ett av de stora problemen med att få användare att våga starta webbapplikationen är att det inbyggda säkerhetssystemet i Java Web Start varnar för att godkänna applikationens åtkomst till systemresurser.

[7] http://forum.java.sun.com/thread.jsp?forum=57&thread=4263904
[8] http://skrutten.nada.kth.se/grim/forum
[9] http://www.phpbb.com

En gästbok skulle kunna få användare att känna sig säkrare på att applikationen fungerar som avsett om de kan se att andra har använt den.

I gästboken kan vem som helst lämna sina meddelanden och den har inmatningsfält för de vanligaste frågorna som namn, rubrik och meddelande. Om gästboken missbrukas bör den dock tas bort eftersom lagarna är sådana att administratören är ansvarig för vad som visas på webbplatsen. Dock har en loggning av användarens IP samt visning av densamma i gästboken en avskräckande effekt på dylika tilltag.

5.6.3 Programstatus

En kontrollerad användning av webbapplikationen är nödvändig för driftsäkerhetens skull. Därför skapades en webbsida där de klienter som är anslutna till systemet visas. Det går alltså att se hur många användare som utnyttjar systemet parallellt och den totala tiden som varje klient har använts. Två av webbapplikationens tillhörande Java Servlets kommunicerar med databasen MySQL och de håller systemet uppdaterat.

Servleten AliveServlet lyssnar på anrop från webbapplikationer som ska komma varje minut. Om en inloggad klient inte hör av sig på fem minuter antas att den inte längre är i funktion. I sådana fall uppdateras informationen om klienten med ny status. AliveServlet uppdaterar också antalet minuter en viss klient varit ansluten.

Ytterligare en Servlet kallad StatusServlet lyssnar på anrop från webbapplikationer som kommer varje gång som användaren byter programområde i webbapplikationen. Tanken är att kunna logga hur många minuter en viss del i webbapplikationen varit aktiv. En del kan till exempel vara Lexin, Parole eller SweSum. Dessutom tillkommer vissa automatiskt aktiverade signaler som bland annat talar om att applikationen precis startat upp.

För att inte acceptera statusändringar från vilken webbapplikation som helst skickas alltid en speciell kod med till AliveServlet och StatusServlet som skapades första gången webbapplikationen startades upp. Koden är försedd med kontrollsiffror De två sista siffrorna i koden är de två sista siffrorna i summan på adderingen av varje teckens ASCII-värde i de sex första tecknen. Systemet är inte speciellt säkert för den som absolut vill förstöra systemet men minskar i alla fall risken.

Eftersom koden skapas slumpmässigt och antas vara unik för varje uppstartad applikation är det lätt att beräkna antalet installerade applikationer. Det kan vara viktigt av flera skäl. Bland annat för ansökningar av forskningsbidrag om eventuell vidareutveckling av webbapplikationen ska ske. Överhuvudtaget är en god kontroll över antalet distribuerade applikationer och deras användningssätt en god motivation för uppdatering och vidareutveckling av systemet.

5.7 Vidareutveckling

Den utvecklade inlärningsmiljön är en bra plattform för att implementera nya språkverktyg. Den tekniska lösningen för applikationen är modulär och har en väl genomtänkt design som baseras på generella designmönster. Genomgående har hänsyn tagits till användaren och dennes behov i skrivprocessen. Att förstå inlärningsmiljön

ska inte vara svårare än att förstå språkverktygen. Givetvis antas att användaren har grundläggande datorkunskaper för att få maximalt utbyte av applikationen men tröskelnivån är låg för att alla ska kunna ta del av den.

Nya språkverktyg som kan implementeras i miljön är till exempel stöd för synonymer. De flesta som använt en ordbehandlare har någon gång använt denna funktion för att att variera och berika språket. Med tanke på hur pass stor roll ordinlärning har på andraspråksinlärning är en sådan funktionalitet önskvärd.

Lexin bör flyttas ännu närmare inläraren genom att integrera Lexins gränssnitt i webbapplikationens huvudfönster. Att växla mellan olika fönster ska inte behöva vara ett tvång utan en möjlighet för de som absolut vill. Konkordansdatabasen PAROLE bör också integreras på ett bättre sätt eftersom ordförståelsen ökar om inläraren kan förstå ordet i olika sammanhang.

Andra tänkbara verktyg är ordböcker för citat och ordspråk som skulle kunna ge inläraren en djupare insyn i det svenska språket. Ordspråk är som regel mycket svåra att förstå för inlärare, som främst måste koncentrera sig på att förstå språket och inte den dolda betydelsen. Givetvis bör verktygen kunna känna igen ordspråk även om en felstavning eller en felaktig böjning förekommer i texten, samt erbjuda korrigerande förslag.

En annan typ av vidareutveckling kan vara att försöka tillvarata lärarnas kunskaper. Idag kan lärarna inte tillrättavisa applikationen om analysen har gett ett felaktigt resultat. Om lärarna gemensamt skulle kunna påverka nästkommande analyser så skulle säkert nivån på analyserna kunna höja ytterligare. En förutsättning av detta är att någon form av inloggning i systemet måste konstrueras.

Dessutom skulle nivån på analyserna kunna förbättras avsevärt med hjälp av en kombination av maskininlärning och de existerande teknikerna.

5.8 Avslutande ord

I stort sett har jag lyckats konstruera den inlärningsmiljö som jag föreställde mig innan projektet påbörjades. Om den däremot uppfyller sitt syfte är svårare att dra någon slutsats om. En omfattande och grundläggande undersökning borde kunna ge svaret på den frågan. Webbapplikationen har hittills haft väldigt många användare[10] och flertalet tillkommer varje dag så underlaget för en sådan studie finns säkert.

Trots att vissa delar inte gick att uppfylla på grund av tekniska förhinder är jag i stort sett nöjd över resultatet. Men hade jag påbörjat projektet idag så hade vägen till målet varit klart kortare. Tyvärr har min omväg förbi bland annat Java Applets och webbläsare försenat projektet betydligt.

För att visualisera abstrakt information på ett så optimalt sätt som möjligt, måste vi komma ihåg att nya teknologier gör det möjligt för oss att visualisera information som vi aldrig skulle kunna göra på ett vanligt papper. Därför bör vi förvänta oss att det kommer ta ytterligare en tid för att erhålla den nödvändiga förståelsen innan vi kan applicera teknologin så effektivt som vi önskar [Gershon and Page, 2001].

[10] 1759 stycken den 15 juni 2004

Referenser

[Borin, 2002] Borin, L. (2002). What Have You Done for me Lately? The Fickle Alignment of NLP and CALL. Paper accepted for presentation at the EuroCALL 2002 pre-conference workshop on NLP in CALL, Jyväskylä, Finland.

[Borin, 2003] Borin, L. (2003). *Datorstödd språkinlärning och språkteknologi*. Institutionen för svenska språket, Göteborgs universitet.

[Domeij, 2003] Domeij, R. (2003). *Datorstödd språkgranskning under skrivprocessen*. PhD thesis, Stockholms universitet.

[Gershon and Page, 2001] Gershon, N. and Page, W. (2001). What Storytelling Can Do for Information Visualization. *Communications of the ACM*, 44:31–37.

[Knutsson, 2001] Knutsson, O. (2001). *Automatisk språkgranskning av svensk text*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden. TRITA-NA-0105.

[Knutsson et al., 2003] Knutsson, O., Bigert, J., and Kann, V. (2003). A Robust Shallow Parser for Swedish. Nodalida '03 Proceedings from the 14th Nordiske datalingvistikkdager, Reykjavik, Iceland.

[Knutsson et al., 2002a] Knutsson, O., Cerratto Pargman, T., and Severinson-Eklundh, K. (2002a). Computer Support for Second Language Learners' Free Text Production - Initial Studies. Proceedings of ICL2002, 5th International Workshop on Interactive Computer Aided Learning, Villach, Austria.

[Knutsson et al., 2002b] Knutsson, O., Cerratto Pargman, T., and Severinson-Eklundh, K. (2002b). Transforming Grammar Checking Technologies into a Learning Environment for Second Language Writing. Institutionen för numerisk analys och datalogi, Kungliga Tekniska Högskolan.

[Levy, 1997] Levy, M. (1997). *Computer-Assisted Language Learning; Context and Conceptualization*. Oxford University Press.

[Loy et al., 2002] Loy, M., Eckstein, R., Wood, D., Elliot, J., and Cole, B. (2002). *Java Swing*. O'Reilly & Associates, Inc.

[McGee and Ericsson, 2002] McGee, T. and Ericsson, P. (2002). The politics of the program: MS WORD as the invisible grammarian. *Computers and Composition*, 19:453–470.

[Robinson and Vorobie, 2003] Robinson, M. and Vorobie, P. (2003). *Swing*, chapter 20. Manning Publications Company. http://www.manning.com/sbe/files/uts2/Chapter20html/Chapter20.htm.

[Sosnoski, 2002] Sosnoski, D. S. (2002). Xml and java technologies: Data binding with castor. http://www-106.ibm.com/developerworks/java/library/x-bindcastor/index.html.

Bilaga A

Mac-specifikt

```
// check if Mac OS X
if (System.getProperty("mrj.version") != null) {
  try {
    //use menubar at top of screen (default Mac-layout)
    System.setProperty(
      "com.apple.macos.useScreenMenuBar",
      "true");

    //set the application name
    System.setProperty(
      "com.apple.mrj.application.apple.menu.about.name",
      "ApplicationName");

    //instantiate a class dynamically
    Object[] args = { this};
    Class[] arglist = { MainFrame.class};
    Class mac_class = Class.forName("SpecialMacHandler");
    Constructor new_one = mac_class.getConstructor(arglist);
    new_one.newInstance(args);
  }
  catch (Exception e) {
  }
}
```

Figur A.1. Ett exempel på hur klasser kan laddas dynamiskt i Java.

```
package com.apple.mrj;

public class MRJApplicationUtils {
  public static final void registerAboutHandler( MRJAboutHandler _handler) {}
  public static final void registerPrefsHandler( MRJPrefsHandler _handler) {}
  public static final void registerQuitHandler( MRJQuitHandler _handler) {}
}
```

```
package com.apple.mrj;

public interface MRJAboutHandler {
  public void handleAbout();
}
```

```
package com.apple.mrj;

public interface MRJPrefsHandler {
  public void handlePrefs();
}
```

```
package com.apple.mrj;

public interface MRJQuitHandler {
  public void handleQuit();
}
```

Figur A.2. Klassen MRJApplicationUtils och gränssnitten MRJAboutHandler, MRJPrefsHandler och MRJQuitHandler som används för kompileringen av källkoden.

```
package se.kth.nada.grim;

import com.apple.mrj.*;

public class SpecialMacHandler implements MRJAboutHandler,
                                          MRJPrefsHandler,
                                          MRJQuitHandler {
  private MainFrame frame;

  public SpecialMacHandler( MainFrame _frame) {
    this.frame = _frame;
    MRJApplicationUtils.registerAboutHandler( this);
    MRJApplicationUtils.registerPrefsHandler( this);
    MRJApplicationUtils.registerQuitHandler( this);
  }

  public void handleAbout() {
    this.frame.about();
  }

  public void handlePrefs() {
    this.frame.prefs();
  }

  public void handleQuit() {
    this.frame.quit();
  }
}
```

Figur A.3. Klassen SpecialMacHandler följer med webbapplikationen.

Bilaga B

Databastabeller

```
CREATE TABLE exceptions (
  id smallint(5) unsigned NOT NULL auto_increment,
  time timestamp(14) NOT NULL,
  exception blob,
  properties blob,
  PRIMARY KEY  (id)
) TYPE=MyISAM;
```

Figur B.1. Databasens tabell som lagrar felinformation.

```
CREATE TABLE status (
  id int(11) NOT NULL auto_increment,
  cid varchar(14) NOT NULL default '',
  state varchar(10) NOT NULL default '',
  alive timestamp(14) NOT NULL,
  status varchar(20) NOT NULL default '',
  time timestamp(14) NOT NULL,
  ip varchar(16) NOT NULL default '',
  min int(11) NOT NULL default '0',
  PRIMARY KEY  (cid)
) TYPE=MyISAM;
```

Figur B.2. Databasens tabell som lagrar aktuell status.

```
CREATE TABLE guestbook (
  time timestamp(14) NOT NULL,
  id int(11) NOT NULL auto_increment,
  name varchar(60) default NULL,
  city varchar(60) default NULL,
  email varchar(60) default NULL,
  url varchar(60) default NULL,
  subject varchar(100) default NULL,
  message mediumtext,
  ip varchar(16) default NULL,
  PRIMARY KEY  (id)
) TYPE=MyISAM;
```

Figur B.3. Databasens tabell som lagrar gästbokens information.

Bilaga C

Filstrukturen

```
-ROOT
    -examples
        Padjelanta.rtf
    -images
        about.gif
        logo.gif
    -lib
        castor.jar
        forms-1.0.4.jar
        grim.jar
        grimx.jar
        looks-1.2.1.jar
        xercesImpl.jar
    -properties
        grim.properties
        text.properties
        text_sv_SE.properties
        text_en_US.properties
    grim.jnlp
    index.html
    install.html
    press.html
```

Figur C.1. Strukturen från roten på projektets vitala delar som körs på webbservern Apache.

```
-ROOT
    -META-INF
        MANIFEST.MF
    -WEB-INF
        -classes
            -se
                -kth
                    -nada
                        -grimservlets
                            AliveServlet.class
                            ExceptionServlet.class
                            GTAServlet.class
                            GranskaServlet.class
                            InflectServlet.class
                            LexinServlet.class
                            ParoleServlet.class
                            StatusServlet.class
                            SweSumServlet.class
        web.xml
    exception.jsp
    exceptions.jsp
    index.jsp
```

Figur C.2. Strukturen från roten på projektets delar som körs på Servlet-motorn Tomcat.

Bilaga D

Förklaring

D.1 Abstract Window Toolkit

Abstract Window Toolkit (AWT) är en föregångare till Java Swing och består liksom den senare av en samling komponenter avsedda för att bygga interaktiva grafiska gränssnitt. AWT är till skillnad från Swing beroende av operativsystem och har inget stöd för MVC-arkitekturen.

D.2 Apache Ant

Ant[1] är ett projekt från The Apache Software Foundation[2] med öppen källkod vars syfte är att automatisera processer som behövs göras ofta av mjukvaruutvecklare. Apache Ant är ett byggverktyg främst för projekt i Java men det går också att göra vanliga procedurer som att kopiera filer och skapa installationskataloger. Verktyget Ant är i sig självt skrivet i Java och sålunda plattformsoberoende.

D.3 Apache HTTP Server

Den i särklass mest använda webbservern för HTTP är idag Apache som har hela 64 procent[3] av marknaden. Projektet som bygger på öppen källkod är den mest populära webbservern på Internet sedan April 1996 och har en mycket stor grupp som utvecklar samt förfinar funktionaliteten av servern.

D.4 Application Program Interface

Application Program Interface (API) är gränssnitt som gör det möjligt att i program och insticksmoduler utnyttja funktioner för vissa tjänster som finns tillgängliga i andra applikationer eller i en funktionssamling.

[1] http://ant.apache.org
[2] http://www.apache.org
[3] Netcraft i Oktober 2003

D.5 Beans

Beans är små enkla klasser, se figur D.1, i Java som främst har syftet att lagra information temporärt. Det kan till exempel röra sig om resultat från sökningar i objekt- eller relationsdatabaser. Gemensamt för dem alla är att de brukar ha metoderna `getX()`, `setX()` och `isX()` för de variabler som förekommer i klassen. Beans ska absolut inte förväxlas med JavaBeans eller PageBeans som är utökningar av denna teknik.

```
public class MyBean {
  private boolean bValue;
  private int iValue;

  public void setBoolean( boolean _b) {
    this.bValue = _b;
  }

  public void setInt( int _i) {
    this.iValue = _i;
  }

  public boolean isBoolean() {
    return this.bValue;
  }

  public int getInt() {
    return this.iValue;
  }
}
```

Figur D.1. Ett exempel på Beans.

D.6 Castor

Castor är ett projekt som bygger på öppen källkod från organisationen Exolab[4] och de inriktar sig bland annat på att automatiskt transformering mellan XML-dokument och Java-objekt. Till skillnad från Document Object Model (DOM) och Simple API for XML (SAX), som båda fokuserar på den interna strukturen i ett XML-dokument, så fokuserar Castor på informationen i XML-dokumentet genom en objektmodell som representerar denna information.

[4] http://castor.exolab.org

Castor kan transformera nästan samtliga Java-objekt som har en struktur liknande Beans till och från XML-dokument. I stort sett har Castor och JAXB liknande funktionalitet och är fullt jämförbara tekniker.

D.7 Certifikat

Ett certifikat, en så kallad publik nyckel, är ett digitalt signerat dokument från en utfärdare. Utfärdaren kan sedan bevisa att den publika nyckeln och dess eventuella värden verkligen hör ihop med den privata nyckeln. Certifikat används i mycket stor utsträckning inom säker kommunikation på webben.

D.8 Common Gateway Interface

Common Gateway Interface (CGI) är en förhållandevis gammal teknik för att webbläsare ska kunna kommunicera med skripts på servern. Skripten är oftast skrivna i programspråket Perl men även C++ förekommer. Användningsområde för dessa skripts är kommunikation mot databaser, gästböcker och dylikt.

D.9 Cookies

Cookies är korta informationsdokument som kan lagras i webbläsaren för att kunna återanvändas. De flesta webbservrar använder Cookies för att genomföra transaktioner och spåra användarprofiler. Rent teknisk är innehållet i Cookies inget annat än ren text.

En cookie, eller kaka, kan inte innehålla virus eller förstöra information hos användaren. En kaka kan heller aldrig läsas av någon annan än den som skapat den. Cookies har en livslängd som bestäms av webbservern och kontrolleras av webbläsaren.

Cookies popularitet bland webbutvecklare har fått EU att reagera och Sverige antog lagen om elektronisk kommunikation som trädde i kraft den 25 juli 2003. Denna lag säger att alla som besöker en webbplats med Cookies ska få information om att webbplatsen innehåller Cookies, vad dessa används till samt hur dessa Cookies går att undvika.

D.10 Jarsigner

Verktyget för att signera JAR-filer heter Jarsigner och är som standard inkluderat i J2SDK från och med version 1.2. För att kunna signera ett arkiv måste en privat nyckel användas. Privata nycklar och dess motsvarighet publika nycklar är lagrade hos utvecklaren i en sorts databas. Denna kan innehålla en mängd nycklar för olika utvecklare som har varsitt lösenord.

Vid signeringen kontrolleras varje enskild fil i arkivet och en kontrollsumma appliceras med hjälp av den privata nyckeln. Den publika nyckeln bifogas arkivet

och sedan verifierar klienten varje dokument i arkivet så att användaren kan vara helt säker på att ingenting ändrats sedan signeringen utfördes. Oftast använder sig jarsigner av standardiserade certifikat enligt specifikationen X509, men andra val är möjliga.

D.11 Java Applet

Java Applets är applikationer, skrivna i Java, som kan inbäddas i dokument formaterade i HTML på samma sätt som bilder och ljud kan inbäddas. När en webbläsare med stöd för Java Applets interpreterar HTML med en inbäddad Java Applet hämtar webbläsaren denna Java Applet och exekverar den i webbläsarens JVM.

D.12 Java Architecture for XML Binding

Java Architechture for XML Binding (JAXB) tillhandahåller ett API och kraftfulla verktyg för automatisk transformering mellan XML-dokument och Java-objekt. JAXB var tidigare en fristående produkt men numera ingår JAXB i det större paketet Java Web Services Development Pack (JWSDP).

JAXB gör XML enkelt att använda genom att kompilera ett XML-dokument till ett eller flera Java-objekt. En kombination av de framställda Java-klasserna och ramverket JAXB ger utvecklare möjlighet att utföra avancerade operationer på ett XML-dokument.

- Uppackning och transformering av XML till motsvarande representation av Java-objekt i hierarki.
- Uppdatering och validering av representationen i Java gentemot fördefinierade schema.
- Packning och transformering av Java-objekt i hierarki till motsvarande representation i XML.

JAXB ger Java-utvecklare ett effektivt och standardiserat sätt att transformera mellan XML och Java. Med JAXB behöver inte Java-utvecklarna vara experter i XML. Det har visat sig att XML och Java är utmärkt i kombination med applikationer för webben och utvecklingsmodellen har ett starkt stöd av utvecklare runt om i världen. Flera uppdateringar har gjorts medan examensarbetet har pågått och de forum som Sun har dedikerat till JWSDP är livliga och innehåller mycket tips från utvecklare runt om i världen.

D.13 Java Archive Files

Java Archive Files (JAR) är ett filformat som möjliggör paketering av multipla filer till enkla filarkiv. För det mesta innehåller ett sådant arkiv klassfiler och andra

resurser som är relaterade med Java Applets och Java-applikationer. JAR-formatet har många fördelar varav de främsta är;

- Säkerhet. Innehållet i ett JAR-arkiv går att signera digitalt. Signaturen består bland annat av kontrollsummor beräknade på varje enskilt dokument i arkivet. Användare som känner igen signaturen kan frivilligt bevilja mjukvaran säkerhetsprivilegier som den normalt inte skulle ha.

- Minskad nedladdningstid. Det behövs bara en förfrågan till webbservern för att hämta hela applikationen istället för en förfrågan för varje enskilt dokument eller klass som tillhör applikationen.

- Komprimering. För att ytterligare minska nedladdningstiden kan JAR-arkivet komprimeras för en effektivare lagring. JAR-arkiv komprimeras med samma sorts algoritmer som används för det mer välkända ZIP-arkivet. JVM är väl integrerad med denna teknik och kan ladda klassfiler direkt från JAR-arkiv.

D.14 Java Foundation Classes

Java Foundation Classes (JFC) består av en uppsättning bibliotek för att utveckla grafiska gränssnitt och grafisk funktionalitet för klientapplikationer. JFC ingår som standard i J2SDK.

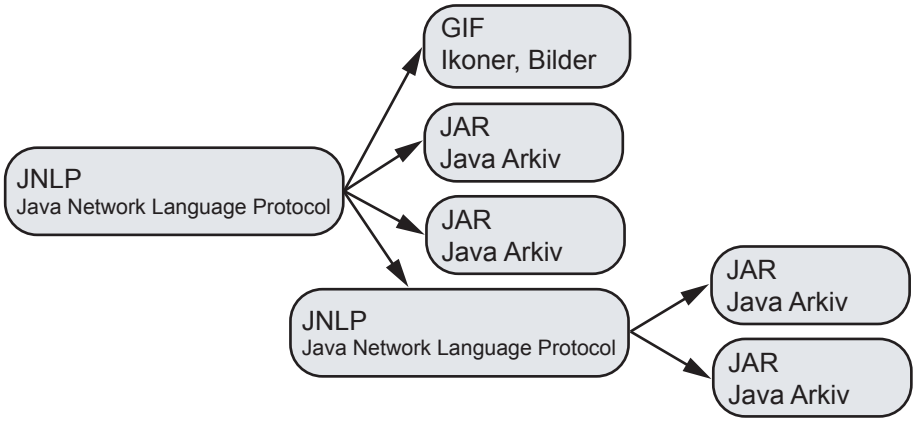
D.15 Java Name and Directory Interface

Java Name and Directory Interface (JNDI) är en standardiserad utökning till Java och tillhandahåller ett enat gränssnitt för ett flertal namn- och katalogtjänster. För att distribuerade komponenter ska kunna hitta varandra behövs nämligen en tjänst som hjälper till med detta, en så kallad namngivningstjänst. JNDI mappar namn mot objekt på samma sätt som DNS mappar namn mot IP.

D.16 Java Network Language Protocol

Java Network Language Protocol (JNLP) är en del av Java Web Start och denna specifikation beskriver en webbaserad applikation. Kärnan i denna teknik är applikationens tillhörande JNLP-dokument som är XML-formaterat. JNLP-dokumentet innehåller inte någon binär information. Istället innehåller den sökvägar som hänvisar till binära data såsom ikoner, klasser och paket samt allmän information om applikationen. Det går också att referera till externa JNLP-dokument.

JNLP API som medföljer Java Web Start är kraftfullt nog att kunna utföra avancerade operationer på ett säkert sätt ur användarens perspektiv. Bland annat så finns det möjlighet att kunna läsa filer på användarens dator och läsa information från operativsystemets klippbok. Den stora skillnaden mot vanliga applikationers



Figur D.2. Ett exempel på ett JNLP-dokument och dess externa resurser.

standardåtkomst av systemresurser är att JNLP API för varje resurs kräver aktivt godkännande med hjälp av säkerhetsdialoger. JNLP API kräver alltså inte att applikationen ska vara signerad för att åtkomst till systemresurser ska vara möjlig.

D.17 Java Plug-in

Java Plug-in är mjukvara som låter användare exekvera Java Applets i webbläsare och är fullt jämförbar med övriga Java-tekniker. Java Plug-in är inkluderat i J2SE som standard och integreras i respektive webbläsare i samband med installation av J2SE. Suns JRE tillhandahåller en Java-kompatibel exekveringsmiljö för de flesta av dagens webbläsare. Detta garanterar följdriktighet och tillförlitlighet vid exekvering av godtyckliga Java Applets.

D.18 Java Reflection

Java Reflection API representerar, eller reflekterar, klasser, gränssnitt och objekt i den aktuella virtuella maskinen. Detta API används oftast för att utveckla debuggers, klassläsare och gränssnittsbyggare. Denna teknik gör att den exekverade programkoden kan undersöka och/eller modifiera egenskaperna hos objekt under programkörning.

D.19 Java Servlet

Java Servlet-teknologin erbjuder webbutvecklare en enkel och konsekvent mekanism för att utöka funktionaliteten av en webbserver och för att göra befintliga affärs-

system tillgängliga. Java Servlet är en Java-teknologi för att förhöja kvalitén och utöka funktionaliteten på webbservrar.

Servlets tillhandahåller komponentbaserade metoder för att konstruera webbaserade applikationer utan begränsningar i prestanda som till exempel hos interpreterande CGI-skript. Till skillnad från en patentskyddad serverteknik som Netscape Server API och olika Apache-moduler är Java Servlets plattformsoberoende. Detta låter utvecklaren välja Best of Breed-strategin för sina servrar, plattformar och verktyg.

En Java Servlet kan betraktas vara en Java Applet men exekveras på serversidan, utan något eget gränssnitt. Denna Java Servlet-teknik har gjort många webbapplikationer med höga krav möjliga. I dag är Servlets ett mycket populärt val för att bygga interaktiva webbapplikationer.

D.20 Java Virtual Machine

Java Virtual Machine (JVM) är själva grunden för att exekvera applikationer i Java. Den virtuella maskinen är en abstrakt dator som likt en riktig dator har en uppsättning instruktioner och den kan manipulera minnesområden under exekvering.

Javas virtuella maskin kan ingenting om programmeringsspråket Java utan bara tolka ett specifikt binärformat, en så kallad bytekod. Det lättaste är att se JVM som ett gränssnitt mellan bytekod och operativsystem.

D.21 Java Web Services Developer Pack

Java Web Services Developer Pack (JWSDP) består av en samling verktyg som tillåter utvecklare att bygga och testa XML-applikationer, webbtjänster och webbapplikationer. JWSDP innehåller bland annat verktyget JAXB men även Servlet-motorn Tomcat.

D.22 Java 2 Software Development Kit

Java 2 Software Development Kit (J2SDK) är en utvecklingsmiljö för Java och inkluderar hela Java 2 Standard Edition (J2SE) samt en uppsättning verktyg. Denna utvecklingsmiljö är standard för dem som vill utveckla egna applikationer i Java. Innehållet täcker det mesta inom klientprogrammering.

Det finns även något som heter Java 2 Runtime Edition (J2RE), som normalt är en del av J2SDK och som kan användas av dem som vill använda men inte utveckla applikationer. Denna nedbantade version går att installera separat och är det alternativ en vanlig användare behöver.

D.23 Java 2 Standard Edition

Java 2 Standard Edition är själva kärnan i Java-teknologin. Det inkluderar flera olika fristående och kraftfulla API, i det här fallet Java-paket, som är standardklasser och kan användas i alla Java-applikationer.

D.24 JavaScript

JavaScript är ett kompakt script-språk för utveckling av klient- och serverapplikationer på Internet. JavaScript stöds av de flesta webbläsare med varierad framgång och används först och främst för händelser på webbsidor. En händelse kan vara att användaren klickar på en knapp på webbsidan. Denna händelse kan sedan exempelvis aktivera en ny sida i webbläsaren. JavaScript är snarare objektbaserat än objektorienterat och trots att det påminner om Java så har det inga större likheter. Framförallt kompileras Java till exekverbar bytekod medan JavaScript interpreteras direkt av webbläsaren. Java är också starkt typat med statiska bindningar medan JavaScript är löst typat med dynamiska bindningar.

JavaScript används ibland på serversidan med en teknik som kallas LiveWire JavaScript. LiveWire JavaScript tillåter utvecklaren skriva serverbaserade applikationer i stil med CGI-skript, Common Gateway Interface. Till skillnad från vanlig JavaScript så kompileras LiveWire JavaScript till exekverbar bytekod. Men funktionaliteten är begränsad och LiveWire JavaScript som serverteknik har i stort ersatts av Java Servlets.

För att lagra Cookies i webbläsaren använder nästan alla webbservrar JavaScript, så tekniken är väl spridd. Men de flesta webbläsare har möjlighet att inaktivera JavaScript eftersom JavaScript har kommit att missbrukas på Internet.

D.25 Keytool

Keytool är ett verktyg för att skapa och hantera nycklar och certifikat och är en del av J2SDK. För att lagra nycklar och certifikat använder keytool en så kallad Keystore. Denna Keystore skyddar den privata nyckeln med ett lösenord.

För närvarande så hanterar Keytool certifikat enligt specifikationen X509. De genererade nycklarna har en längd från 512 till 1024 bitar, men längden måste vara en multipel av 64. Oftast är nyckellängden maximal, det vill säga 1024 bitar.

D.26 LiveConnect

LiveConnect[5] är en teknik som utvecklats av Netscape och möjliggör kommunikation mellan JavaScript och Java Applets. Det går bland annat att använda JavaScript för direkt åtkomst av variabler, metoder, klasser och paket i Java. Det omvända går

[5] http://wp.netscape.com/navigator/v3.0/liveconnect.html

också bra, det vill säga att använda Java för direkt åtkomst av JavaScripts metoder och egenskaper.

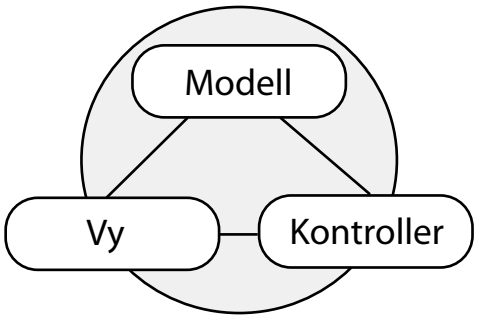
Med LiveConnect så ger Netscape Java-utvecklare möjligheter att ge sina Java Applets en utökad funktionalitet. Bland annat kan Java Applets inte lagra Cookies i webbläsaren, men med LiveConnect får utvecklare den möjligheten.

Till LiveConnect hör ett API i Java som möjliggör denna kommunikation. Detta API finns implementerat i de flesta webbläsare på marknaden men en del av dem har bristande stöd för denna teknik.

Eftersom Netscape numera har upphört med utveckling av sin egen webbläsare Netscape Navigator så är det risk för att även denna teknik försvinner. Förmodligen är enda möjligheten att Mozilla-projektet tar över utvecklingen.

D.27 Model-View-Controller

Model-View-Controller (MVC) är en mycket vanlig och kraftfull arkitektur för att konstruera grafiska gränssnitt och är inte bundet till något speciellt programmeringsspråk. Det finns tre delar i detta designmönster, se figur D.3. Kontrollern kan modifiera modellen som i sin tur ger en uppdaterad vy. I MVC är modellen unik medan det kan finnas olika sätt att visualisera den och olika sätt att modifiera den.



Figur D.3. Det välkända designmönstret MVC.

D.28 PHP: Hypertext Preprocessor

PHP: Hypertext Preprocessor (PHP) är ett skriptspråk som används på webbservrar med stöd för moduler och dessa servrar exekverar dokument innan de skickas från dem. Vid exekveringen är det bland annat möjligt att applicera data från databaser. Stödet för olika databaser är stort i språket PHP, vilket gjort det mycket populärt bland nybörjare i databasprogrammering. PHP[6] är en del av de olika högkvalitativa projekt från The Apache Software Foundation, som är en decentraliserad sammanslutning av utvecklare över hela världen.

[6] http://www.php.net

Nackdelen med PHP är främst att det är ett skriptspråk som måste interpreteras direkt av webbservern. För större webbplatser med många besökare är det ett dåligt alternativ men för mindre organisationer och företag är det bra eftersom inlärningströskeln av PHP är låg. PHP kan i stort liknas med lösningen från Microsoft som kallas Active Server Pages men med den skillnaden att PHP bygger på öppen källkod. Däremot kan PHP inte jämföras med Java Server Pages från Sun eftersom dokument i JSP kompileras till bytekod innan det modifierade dokumentet sänds iväg.

D.29 Servlet-motor

En Servlet-motor används för att exekvera Java Servlets och göra dem tillgängliga för webben. En av den mest kända Servlet-motorerna är Tomcat, som är en del av Jakarta-projektet[7] från The Apache Software Foundation. Den bygger på öppen källkod och anses mycket robust av webbutvecklare över hela världen.

D.30 Structured Query Language

Structured Query Language (SQL) är ett standardiserat sätt att ställa frågor om och modifiera data i en relationsdatabas. Det finns många olika relationsdatabaser där vissa är kommersiella och andra med öppen källkod. Den mest kända och använda databasen baserad på SQL är idag den svenskutvecklade MySQL som lämpar sig för både små och medelstora företag och organisationer. MySQL är licensierad under GNU General Public Licence och finns idag till de flesta förekommande plattformar.

D.31 XML Schema

XML Schema är en specifikation för att beskriva strukturen i XML-dokument. Det har ett mycket bra stöd för olika datatyper och även för viss objektorientering såsom arv och abstrakta element. En föregångare till XML Schema var DTD, Document Type Definition, som tyvärr var oanvändbart i större och mer komplicerade definitioner av XML-dokument. Med XML Schema introducerades en mer flexibel och kraftfull teknik för att beskriva dessa dokument.

Ett XML Schema, eller bara schema, identifierar element som kan förekomma i ett XML-dokument, i vilken ordning de förekommer, vilka attribut de kan ha och vilka element som är underordnade. Ett XML-dokument behöver inte ha ett schema, men om det har så måste dokumentet uppfylla schemats specifikation. JAXB och andra relaterade tekniker utgår ifrån att XML-dokument har beskrivande scheman och att dessa scheman uppfyller W3C XML Schema Language[8].

[7] http://jakarta.apache.org
[8] http://www.w3.org/XML/Schema