NANYANG TECHNOLOGICAL UNIVERSITY

SCE00-083
Customizing Kirrkirr for the Microsoft browser

Submitted in Partial Fulfilment of the Requirements
for the Degree of Bachelor of Applied Science
of the Nanyang Technological University

Supervisor: Dr. Nitin Indurkhya
Student: Lim Hong Lee

ABSTRACT

This project was initiated to exploit the capabilities of the web browser (Internet Explorer 5) to improve the performance of Kirrkirr (an electronic dictionary of English and Warlpiri, an Australian Aboriginal language). The main features exploited are the use of Stylesheets to handle the dictionary entries, which are all in XML files and also the use of Macromedia Flash to enhance the audio and visual experience of the user. Another feature exploited is the use of JavaScript to implement some of the functions of the original Java Applet so as to reduce the program size and startup speed. Comparisons are also made between the different ways of implementing these features so as to evaluate their performances. As a result of the improvements made and features added, Kirrkirr is now more enjoyable.

Table of Contents

CHAPTER 1 INTRODUCTION **1**

1.1 BACKGROUND 1
1.2 WED-BASED KIRRKIRR 1
1.3 PROBLEMS WITH THE EXISTING VERSION 3
1.3.1 Under-utilization of the web browser *3*
1.3.2 Slow startup process *3*
1.4 SCOPE AND OBJECTIVE 3
1.5 MAIN ACCOMPLISHMENT 4

CHAPTER 2 DESIGN ISSUES **5**

2.1 ENVIRONMENT 5
2.1.1 Cross Platform Consideration *5*
2.1.2 Browser Consideration *5*
2.2 DISPLAY 6
2.2.1 Main Components *6*
2.2.2 Space Constraints *6*
2.2.3 Application Window *7*
2.2.4 Browser Window *8*
2.2.5 Web-Page *9*
2.2.6 Working with frames *10*

CHAPTER 3 STYLESHEETS **12**

3.1 STYLESHEETS 12
3.1.1 How Stylesheets Work *12*
3.1.2 HTML styling Vs Stylesheets styling *13*
3.1.3 Connecting Stylesheets to Documents *14*
3.2 CASCADING STYLE SHEETS (CSS) 15
3.2.1 Introduction to CSS *15*
3.2.2 Features of CSS *16*
3.2.3 Using CSS in Kirrkirr *18*
3.3 EXTENSIBLE STYLING LANGUAGE (XSL) 20
3.3.1 Introduction to XSL *20*
3.3.2 Features of XSL *20*
3.3.3 Using XSL in Kirrkirr *25*
3.4 CSS VS XSL 27

CHAPTER 4 PROGRAMMING LANGUAGE 28

4.1 JAVASCRIPT 28

4.1.1 Introduction to JavaScript 28

4.1.2 Features of JavaScript 29

4.1.3 Java vs JavaScript 33

4.1.4 Using JavaScript in Kirrkirr 35

CHAPTER 5 WEB ANIMATION 38

5.1 MACROMEDIA FLASH 38

5.1.1 Introduction to Flash 38

5.1.2 Features of Flash 38

5.1.3 Using Flash in Kirrkirr 44

CHAPTER 6 EVALUATION 49

6.1 PROGRAM PERFORMANCE EVALUATION 49

6.1.1 Startup Speed and Program Size 49

6.1.2 Other Improvements 49

6.2 STYLESHEET PERFORMANCE EVALUATION 50

6.3 MACROMEDIA FLASH PERFORMANCE EVALUATION 51

CHAPTER 7 FUTURE WORK AND CONCLUSION 52

7.1 FUTURE WORK 52

7.2 CONCLUSION 53

CHAPTER 8 REFERENCES 54

LIST OF ILLUSTRATIONS

Figures

Figure 1. Original version of Kirrkirr. 1
Figure 2. Application window of the new Kirrkirr. 7
Figure 3. Application window of the new Kirrkirr. (Panel clicked) 8
Figure 4. Browser window of the new Kirrkirr. 8
Figure 5. Full-screen view of the new Kirrkirr. 11
Figure 6. Transformation and Formatting. 13
Figure 7. A typical word entry displayed using CSS 18
Figure 8. A typical word entry displayed using XSL. 26
Figure 9. The JavaScript language 28
Figure 10. Previewing a movie with the Bandwidth Profiler. 43
Figure 11. A typical word entry displayed using Flash. 44

Tables

Table 1. JavaScript compared to Java 34

Table 2. Comparison between the original and new versions of Kirrkirr 49

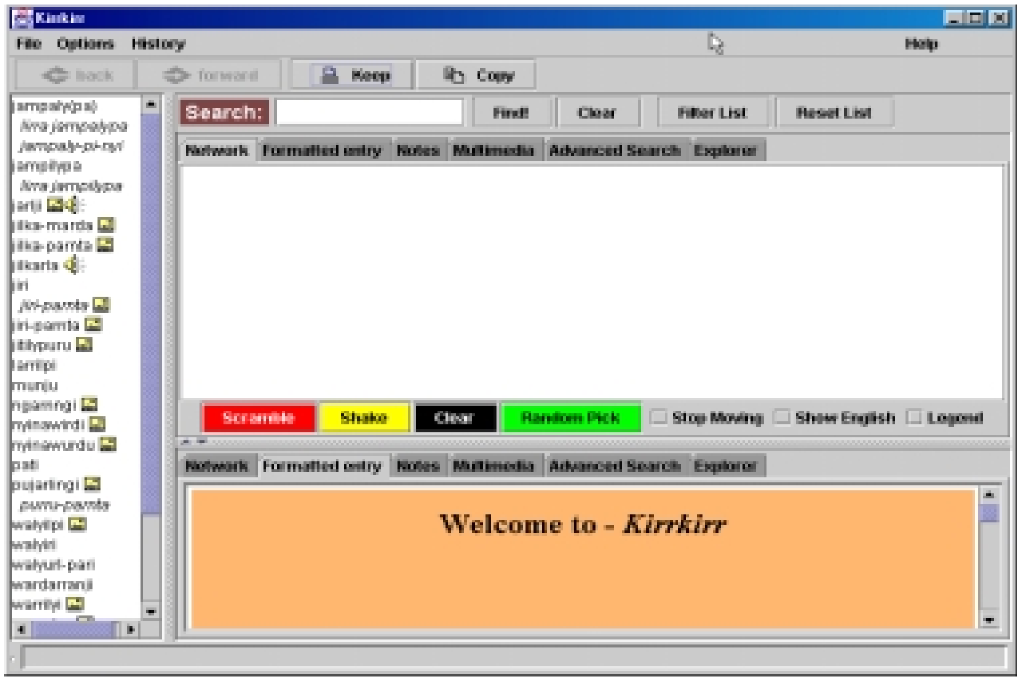
Chapter 1 Introduction

1.1 Background

Kirrkirr is a bilingual electronic dictionary of English and Warlpiri (an Australian Aboriginal language) and was originally developed by Kevin Jansz [Jansz1998]. It is not just an electronic dictionary that only searches and retrieves dictionary data and finally displays them in plain format. Instead, the retrieved data will be displayed in a number of ways including a graphical format. It can be started both as an application and as a Java Applet.

1.2 Wed-based Kirrkirr

In this report, we are mainly dealing with the web-based applications of Kirrkirr, paying particular attention to Microsoft Internet Explorer 5 (IE5) and above. In order to allow users to be able to make use of Kirrkirr anywhere and anytime, it is a must to put Kirrkirr on the web. Currently, Kirrkirr can be run on the Internet by using any web browsers that can support Java Plugins or Applets. It is started by a Java Applet called *Demo.class*, which in turn starts the Kirrkirr application on the client computer.



Kirrkirr
File Options History
Help
back forward Keep Copy
Search:
Find! Clear Filter List Reset List
Network Formatted entry Notes Multimedia Advanced Search Explorer
Scramble Shake Clear Random Pick
Stop Moving Show English Legend
Welcome to - Kirrkirr

Figure 1. Original version of Kirrkirr.

The existing version of Kirrkirr uses the following tabbed panels to help users understand the language better:

Network Panel:

This is the visual animated representation of the words. It mainly shows words which are related to the current one. This greatly enhances the learning experience as users are introduced to other words of interest. Finally it also takes away the trouble of the conventional way of flipping through pages to search for related words.

Formatted Entries Panel:

After the words have been found, they are extracted from the dictionary (which is an XML file) and converted into HTML format. This panel displays these HTML files to show the meanings of the words in English.

Notes Panel:

Users are allowed to make their own notes for every word and all these notes can be saved in a user profile. This is very useful as it helps users remember certain information or storing questions about the particular word.

Multimedia Panel:

Pictures and sounds (for pronunciation) are available for many words to help users understand and speak the language easier and better. The Multimedia Panel implements this feature.

Advanced Search Panel:

The Advanced Search options let the users query the database using regular expressions, fuzzy spelling or just plain spelling. The algorithm is in such a way that simple spelling mistakes are tolerated and corrected automatically.

It is exactly the same as the non web-based version in terms of looks and functions. This, however, also brings us the following problems.

1.3 Problems with the Existing Version

1.3.1 Under-utilization of the web browser

As the Applet *Demo.class* simply opens a new window to start Kirrkirr, the original web browser window is then totally disregarded by the application and not used at all after that. Much of the browsers capabilities are not used and exploited, in IE5's case, we are talking about the ability to display XML files directly and the usage of JavaScript. Since Kirrkirr's dictionary file is in XML format and then converted to HTML format for display, we can actually take away this extra step and display more information onto the browser window easier and faster. The JavaScript can also implement some of the functions of the Panels in Kirrkirr thus again increasing the speed of execution of the program. The final program size will also be reduced.

1.3.2 Slow startup process

Currently, the startup process is very slow, as with all applets. Since applets are started by downloading the appropriate class files to the client computer for the Java Virtual Machine (JVM) to process, if we can make the program size smaller, downloading will then take less time and thus making the whole process faster.

1.4 Scope and Objective

Having defined the various problems that currently exist in the web-based Kirrkirr application today, the objective of this project is to investigate these problems further to reduce them to a minimum. To achieve this, there is one clear solution available – exploiting IE5's browser capabilities and in turn reducing the program size.

Although by doing so, this may make Kirrkirr available only to IE5 users, but it is something that we need to give up in order to optimize the running of Kirrkirr on the web. We could, however, try to keep away from browser specific functions and hopefully able to reproduce the same results for other browsers without much difficulty.

Therefore, the objective of this project is to optimize the web-based Kirrkirr application and exploit the capabilities of IE5. More specifically, this project will investigate the following issues:

- To redesign the architecture of the Kirrkirr applet.
- To investigate IE5's capabilities on dealing with XML files
- To implement some of the functions of Kirrkirr onto the browser using JavaScript, XML and HTML.
- To simplify the Kirrkirr program so as to reduce the program size and increase applet startup speed.

1.5 Main Accomplishment

The main objectives mentioned above were accomplished, notably:

- Implements the HTML, Multimedia and Notes panels using JavaScript and HTML.
- Makes use of Stylesheets to display XML files in IE5.
- Compares the performance between the 2 main Stylesheets – XSL and CSS.
- Implements the above mentioned 3 panels using Macromedia Flash 5.
- Compares the way Flash handles XML files with Stylesheets.
- Reduces the program size by 13% and startup speed by 55%.

Chapter 2 Design Issues

There are many limitations and considerations that we need to take into account when it comes to the design of the web-based version of Kirrkirr. In this chapter, we will discuss about all these issues.

2.1 Environment

2.1.1 Cross Platform Consideration

Platform doesn’t really mean the type of computers rather what’s more accurate is that platform relates to the type of programming interface a computer uses [Kolzschlag1998]. In simple terms, the most critical aspect of platform for the web designer is the operating system used by that computer. There are 3 primary platforms – PC (includes DOS, Windows, Windows NT), Macintosh and UNIX.

2.1.2 Browser Consideration

The two most popular browsers available today are Netscape and IE. Even with the same browsers, on different platforms, there will still be differences in font sizes, style and visible areas on the page. In order to make a site completely accessible, we have to step very far back from recent trends and follow standards that might not apply to the latest technology.

However, for this project, the main concern is to be able to work with and exploit the capabilities of IE, this would not be a major issue. But we would still try to keep away from browser specific functions.

2.2 Display

2.2.1 Main Components

There are 3 main components that Kirrkirr’s display consists of – the *Application* window, the *Wordlist* applet and the contents of the selected word entry in the dictionary, including its explanation, pictures and sound.

This causes quite a big problem with the display as all these components need a substantial amount of space on the screen and they have to be displayed altogether on the same screen. Our design must accommodate small and large displays, monochrome and color, slow and fast transmission, and various browsers that may not support desired features.

2.2.2 Space Constraints

[Weinschenk1997] Research has shown that most people use their system as it comes out of the box. That means many are using 800 x 600 screen resolutions. Moreover, sizes of monitors are increasing nowadays and lots of people are changing to using 17" monitors with 1024 x 768 resolutions. Thus we must make sure that all the 3 components can be displayed properly and provide a quality experience regardless of the user’s display resolution.

[Shneiderman1998] To deal with this space constraint, we have to make our components as compact as possible. The *Wordlist* applet, being just a list by itself, is already in its most compact form and cannot be reduced further. The contents of the word entries should be the main focus of the users and should take up the highest possible percentage of the whole screen, so it cannot be reduced in size too. This leaves us with the *Application* window, which is actually required for background operations most of the time and only needs to be shown when the graphical panels are needed.

2.2.3 Application Window

Originally, the Kirrkirr application window consists of the toolbar, wordlist and 2 graphical panels. This makes the window rather big and in fact it will take up about 30-40% of the whole screen on a 17" monitor. However, now that we have the luxury of the browser at our disposal, the bottom panel, which is mainly displaying the explanations of the words in HTML format, can be removed. It will be replaced by displaying the XML version of the explanations directly on the browser window. Furthermore, the wordlist is also no longer required as it will be replaced by the *Wordlist* applet, residing on the browser window also.

Even after removing the wordlist and bottom panel from the application window, it is still quite big and will cover a big part of the browser window. This is undesirable as the main focus of the users will be on the browser window to get the meanings and examples of the words and will only focus on the graphical panel when they wanted to get the graphical representation of the words or do an advanced search. Thus, the graphical panel can actually be hidden until it is needed and only becomes visible when the user activates it by clicking on the appropriate button.

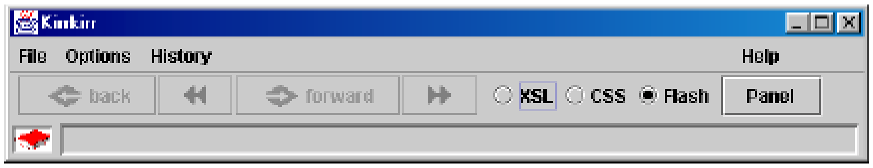


Figure 2. Application window of the new Kirrkirr.

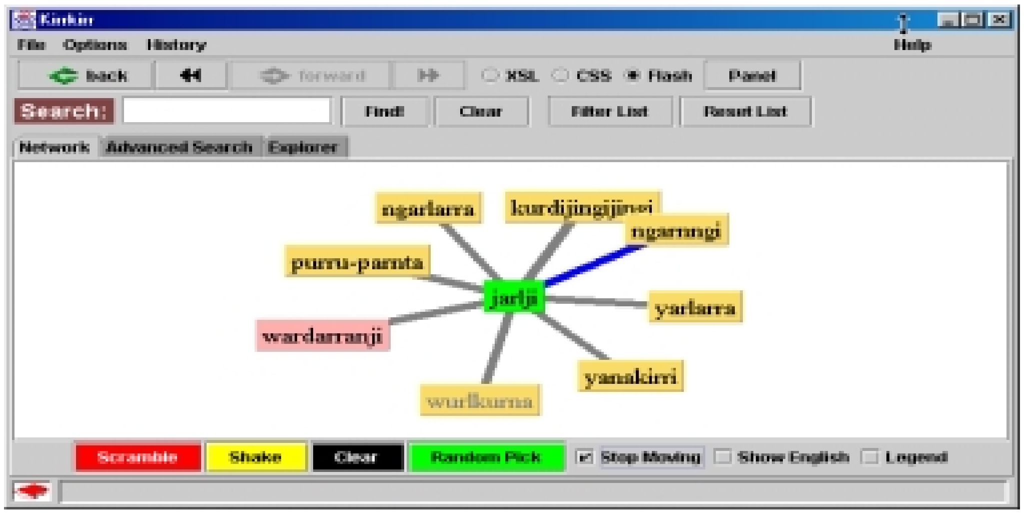

Kirrkirr
File Options History Help
back forward XSL CSS Flash Panel
Search: Find! Clear Filter List Reset List
Network Advanced Search Explorer
ngarlarra
kurdijingijinpi
ngamngi
purru-parnta
jarlji
yarlarra
wardarranji
yanakirri
wurlkurna
Scramble Shake Clear Random Pick Stop Moving Show English Legend

Figure 3. Application window of the new Kirrkirr. (Panel clicked)

Therefore, the application window will be reduced to only a small toolbar or panel that only contains the main functions of the application and will expand to show the graphical panel when called upon.

2.2.4 Browser Window

After discussing on the design of the application window, we will now look at the browser window.

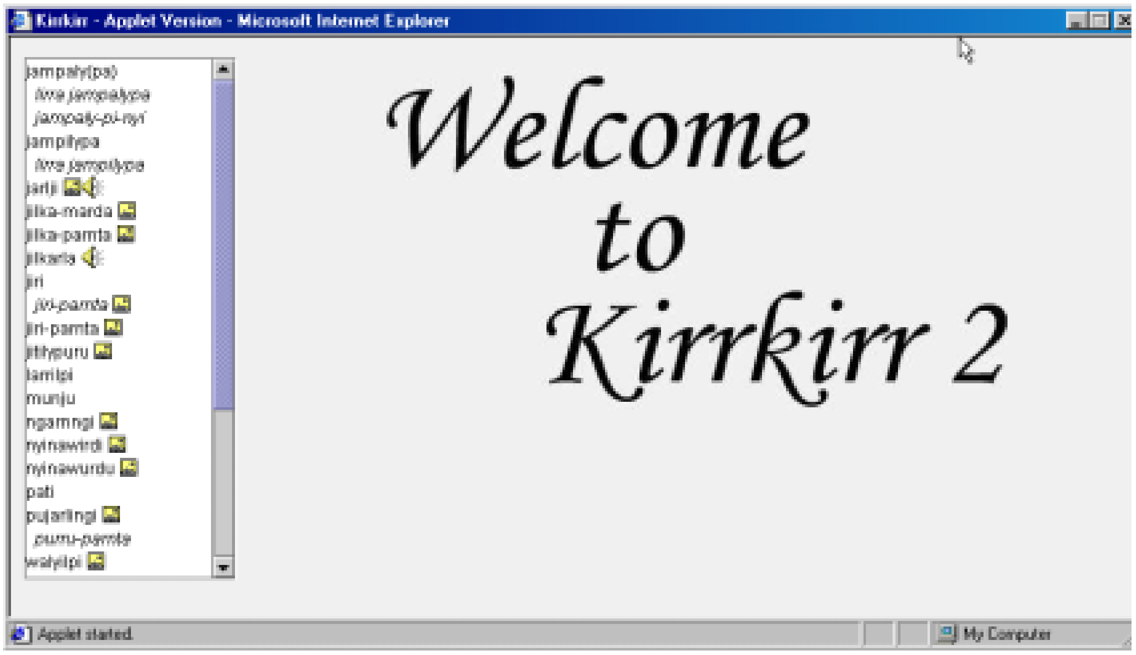


Figure 4. Browser window of the new Kirrkirr.

The application window, which is now a small toolbar, will also take the place of the browser’s toolbars. Although it does not have all the functions of all of the browser’s toolbar options, those functions are not required when running Kirrkirr. Thus, the browser window will only contain the title bar and status bar and this greatly increases the display area of the browser window.

To achieve this, we need to make use of JavaScript event, *onClick*, to open a new window for Kirrkirr to reside in. The example below shows the HTML code for a button to start Kirrkirr from a webpage:

```
<INPUT TYPE=BUTTON VALUE="Start Kirrkirr 2"
onClick="window.open( 'kirrkirr.htm', 'win', 'toolbar=0, location=0,
directories=0, status=1, menubar=0, scrollbars=1, resizable=0');">
```

Since there is no other way to remove the unused components from the browser window once it is started, we can only do it when it is starting. This may prove to be quite inconvenient for webmasters to add Kirrkirr to their websites as it is not just a simple button or link, but it really is a small price to pay to get a better display. Moreover, even if they only make a simple link to start Kirrkirr, it will still work perfectly fine as the only difference is just the display area. Therefore this is actually a win-win situation.

2.2.5 Web-Page

Since we are now talking about the web-based application of Kirrkirr, web-page design is definitely something that we must not overlook. This is where the user’s main focus will be and takes up the main bulk of the display so this is naturally one of the more important issues.

As mentioned earlier, there are 3 main components for display. The *Application* window has already been taken care of, so we are now left with the *Wordlist* applet and the contents of the dictionary entries. To make things simpler, we would separate the *Wordlist* applet with the contents by putting them on different frames.

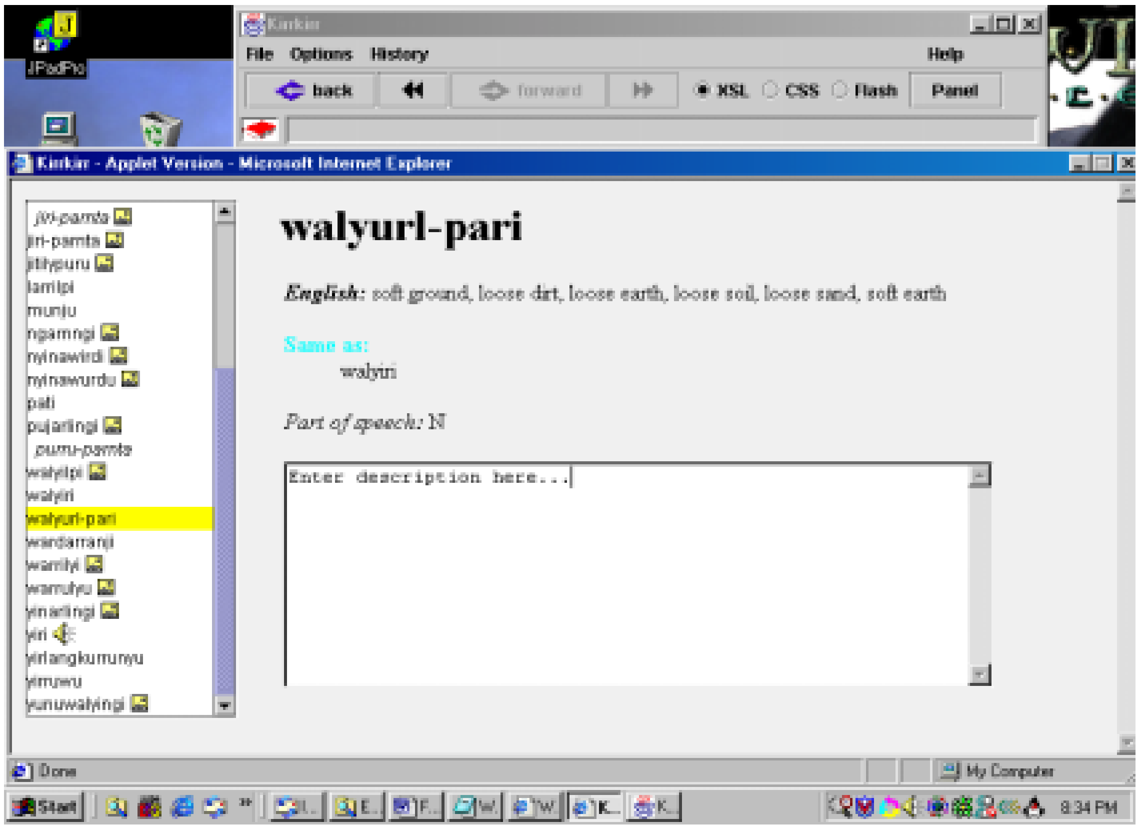
2.2.6 Working with frames

[Weinschenk1997] Many Web users find frames confusing. How frames work depends partially on the web browser version the user is using. It may be hard for users to find the right scroll bar, and there may be too much scrolling required. Users may not know how to go back to a previous page. Printing can also be problematic. Users trying to place a bookmark will find that it only saves the site; frames don't allow them to bookmark an individual content page.

Furthermore, the number of windows or frames should be limited to two. If we break the display area up into more than two windows or frames, we would reduce the content viewing area too much. The users feel like they are looking through a peephole.

Thus, we should only consider frames for global elements. All global elements, such as global navigation, corporate identification, and mail to links, can be placed on frames that remain no matter what content window is displayed. Content windows are freed from displaying this information.

In Kirrkirr's case, the *Wordlist* is displayed in an applet and is the common element on all pages, so it will reside in a frame, which does not change. This removes the problem of having many frames that confuses users, as they will now know where the changes will occur – the content frame. Moreover, if the *Wordlist* applet is not in a frame, then when the contents of the page are changed, the applet will have to be loaded again. This is definitely not an efficient way and may even cause problems to the applet.



walyurl-pari

English: soft ground, loose dirt, loose earth, loose soil, loose sand, soft earth

Same as:
walyiri

Part of speech: N

Enter description here...

Figure 5. Full-screen view of the new Kirrkirr.

Chapter 3 Stylesheets

3.1 Stylesheets

HTML can be frustrating when trying to control the appearance of the web page and its contents. Tables are awkward, frames are annoying, and FONT tags threaten to overwhelm the code. The truth is that HTML was intended to mark up only a web page's structure and not how it displayed on the screen.

Stylesheets work like templates: just define the style for a particular HTML element once, and then use it over and over on any number of web pages. If we want to change how an element looks, we can just change the style and the element automatically changes wherever it appears. Stylesheets let web designers quickly create more consistent pages--and more consistent sites.

3.1.1 How Stylesheets Work

[XSL2000] A *StyleSheet Processor* accepts a document or data in XML and a Stylesheet and produces the presentation of that XML source content that was specified by the Stylesheet. The presentation process will first begin by doing a *Tree Transformation*, that is, constructing a result tree from the XML source tree. After that *Formatting* will take place, interpreting the result tree to produce formatted results suitable for presentation on the display. This process of formatting is performed by the *Formatter*, which may simply be a rendering engine inside a browser.

Tree Transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, one could add a table-of-contents as a filtered selection of an original source document, or one could rearrange source data into a sorted tabular presentation. In constructing the result tree, the tree transformation process also adds the information necessary to format that result tree.

Formatting is enabled by including formatting semantics in the result tree. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and hyphenation control.

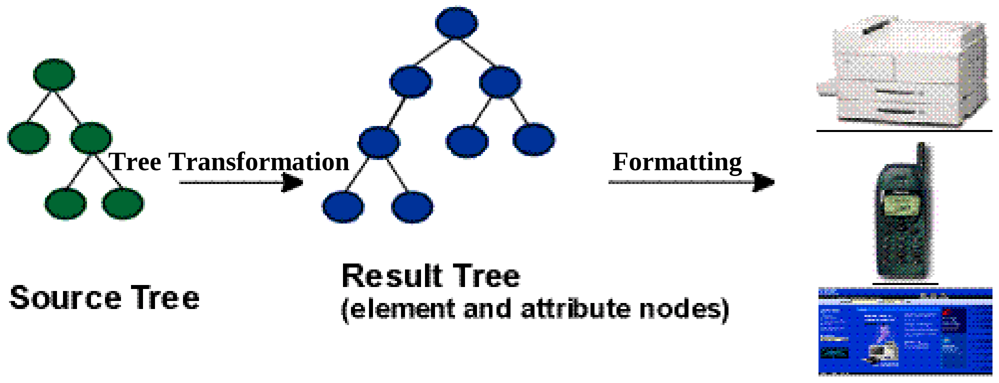


Figure 6. Transformation and Formatting.

3.1.2 HTML styling Vs Stylesheets styling

[Boumphrey1998] Most pages use lots of images to compensate for the difficulty of styling with HTML tags. As a result, it would become a graphics heavy site and this takes time to download, probably around 30 to 40 seconds. However, if Stylesheets are used, the pages will be largely reduced in size and can be loaded in less than a few seconds.

Even if HTML type styling is used in place of the graphics, the pages themselves are also larger - almost twice the size of pages using Stylesheets. All those extra bytes come from the styling tags.

HTML styling has the following disadvantages when it comes to maintenance:-

- Difficult to read the pages due to all the styling elements and tags.
- If content is changed, balance of the pages is often upset and the tags and attributes need to be altered and may even need to do trial and error in order to set things right.
- There is no easy way to alter the overall style of a set of pages without altering every tag individually, usually by hand.

- In fact, it is often easier to write a one-off program to handle the problem with code or script.

However, all these problems do not occur when using Stylesheets and it has the following advantages too:-

- Only need to alter a single Stylesheet rather than the thousands of pages.
- Size of the HTML document is almost halved.
- Separate the authoring and styling part of the job.
- Save time and money.

Since Kirrkirr is a dictionary, it will contain thousands of words, each with their own web page. Thus, if HTML styling were to be used, styling tags and elements will have to be added into every page after extracting out the entry from the main dictionary XML file. Furthermore, the dictionary is in XML format that fully supports Stylesheets and since every page has the same layout and format, it only seems natural and convenient to use Stylesheets. So now the problem is what Stylesheets to use, CSS or XSL.

3.1.3 Connecting Stylesheets to Documents

[Laurent1998] Processing Instruction (PI) is used to connect documents to their Stylesheets, appearing before the root element of the document (i.e. before rendering has begun). This PI takes the general format:

<?xml-style href="*stylesheetURL*" rel="*stylesheet*" type="*stylesheetMIMEtype*"?>

An XML document that referred to an XSL spreadsheet might look like:

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="*warlpiri.xsl*" ?>
<mydoc>

```
……content…..
</mydoc>
```

Because XML has no fixed vocabulary, the HTML standbys of META, LINK and STYLE are not available. Processing Instructions avoid the need to modify DTDs to access styles.

In the case of using a CSS Stylesheet, the syntax for the XML document is nearly the same, except for the second line, which would look like this:

<?xml-stylesheet type="text/css" href="*warlpiri.css*" ?>

Thus, it is actually very simple to switch from one Stylesheet to the other and this allows us to try with different Stylesheets. Those more advanced users can even create their own Stylesheets for use with Kirrkirr.

3.2 Cascading Style Sheets (CSS)

3.2.1 Introduction to CSS

For the appearance of web pages, the World Wide Web Consortium (W3C) developed a complementary markup system called *Cascading Style Sheets*, or CSS, designed to make it easy to define a page's appearance without affecting its HTML structure.

[CSS1999] *Cascading Style Sheets* are groups of style rules, which in turn are groups of properties that define how an HTML element appears in a Web browser--for instance, green, italic, and Arial. Styles can be defined either within an HTML document or in an external file attached to the HTML document. As an external file, a single Stylesheet can affect multiple pages--even a whole site. This makes page-wide or site-wide changes much easier to implement.

As the term *Cascading Style Sheets* implies, more than one Stylesheet can be used on the same document, with different levels of importance. If conflicting styles are defined for the same HTML element, the innermost definition--the one closest to the individual tag--wins. For example, a tag's *style* attribute takes precedence over a surrounding element's style, which has priority over a style defined in the head of the document, which in turn wins over an attached Stylesheet.

Browsers began supporting the first CSS Specification, Cascading Style Sheets, Level 1 (CSS1), in versions 3.0 of Microsoft Internet Explorer and in version 4.0 of Netscape Navigator. The 4.0 and later versions of both browsers also support properties from the newer Cascading Style Sheets, Level 2 (CSS2) specification, which lets us specify elements' visibilities, their precise positions on the page, and how they overlap one another.

3.2.2 Features of CSS

Separating style from content

The main benefit in CSS, is that it manages to separate the style from the content on the web page. If HTML is being used, it will have to handle both the style and content. With a FONT-tag, and some artistic sense, HTML page can be quite stylish.

However, if CSS is used, it will handle the style of the web page, and let HTML do the content part. So why is this good? First of all, the stylish capabilities in HTML are quite limited. Secondly, with the use of an external Stylesheet, the entire site style can be altered by only editing one single file. As an example of the above, let's say we wanted to change our site from using Times New Roman to Arial. With "old-fashioned" HTML, we would have to alter the FONT tag, on each and every page. With CSS, we would just need to alter the Stylesheet, and then all pages would be using Arial.

Even though trends are towards decentralization, we actually make things easier, more efficient, and save time, by centralizing our style into one single Stylesheet.

Improved usability

If Stylesheets were being used extensively, the whole web site would automatically become more usable. By removing all nasty BODY-attributes and font tags from the HTML pages, and moving these into a nice single external Stylesheet, all HTML pages will turn smaller in size. The Stylesheet would only need to be downloaded once (as further requests would later be taken from the users cache), and thus the whole web site would load faster.

With the use of CSS, it's also important to remember that unlike the other nice web technologies (e.g. ActiveX, JavaScript, VBScript etc.), users do not loose functionality, if their chosen browser does not support CSS. They will simply just get plain text - the raw content.

Also, remembering that the FONT-tag, and many BODY-attributes are browser specific, the site also becomes easier to read and understand for browsers.

A costly technology

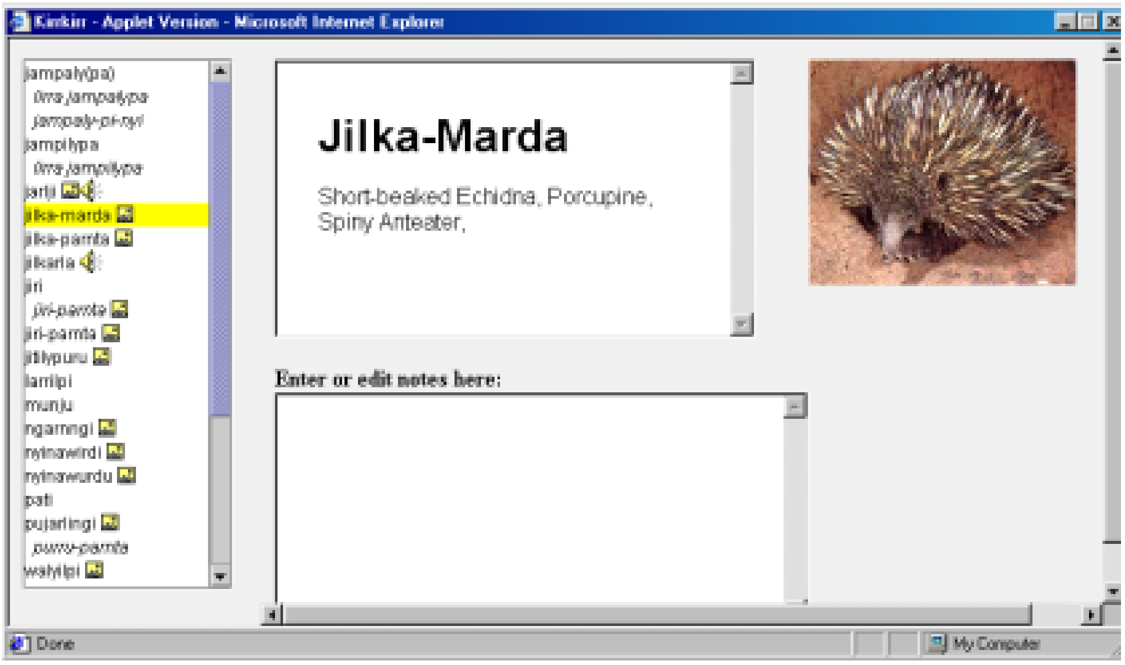
One of the problems of CSS is, that if we want our site to have the very same look and feel, in as many browsers as possible (e.g. both NN3+ and IE3+), we will not be able to remove the style from the HTML code. As among many browsers, NN3 does not support CSS. In fact, it has never heard of it and also IE3 only supports very few parts of it. This means that we would have to carry on with the existing code, and thus using CSS would very likely be an extra burden on our budget.

The above problem gets even further complicated, when we consider that CSS2, the update to CSS1, is now available. CSS2 extends CSS1 and newer browsers will definitely support CSS2 (more or less). The problem is that all the browser vendors seem to have agreed to disagree on which tags they support. So to do things perfectly we would need to know exactly how little CSS is supported in IE5 and exactly how much is supported in NN4.5 etc.

Lowest Common Denominator

On the Web, there seems to be a thing called designing for the lowest common denominator. Perhaps CSS is still not a part of the lowest common denominator at the moment, but CSS brings many advantages to the Web, the user, and also to the web designer. Using it will definitely help save so much time and trouble, thus one should really try to use it on one's web site.

3.2.3 Using CSS in Kirrkirr



Jilka-Marda

Short-beaked Echidna, Porcupine, Spiny Anteater,

Figure 7. A typical word entry displayed using CSS.

In Kirrkirr, CSS is mainly used as a medium to format and display the XML file extracted from the dictionary file. Although the XML format is relatively simple to understand and display, it is definitely not meant for presentation purposes. All the tags (elements and nodes) can make the whole file confusing to the user, so a CSS file is used to select only the required information and display them in a presentable format.

The CSS syntax is very simple to understand and use. The example below shows how a typical node in the XML file can be formatted and displayed.

The XML code:

```
<HW>jiri</HW>
```

The corresponding CSS code to format it:

```
HW
{
   text-transform: capitalize;
   display: block;
   font-size: x-large;
   text-align: left;
   margin-bottom: 0.5em;
   font-weight: bold;
}
```

The above code will make the word "**jiri**" appear in the browser window with extra-large font size, capitalize form and in bold. The syntax is very self-explanatory as shown above.

All the nodes with useful information will be given their corresponding CSS codes while the others will be made invisible to the user. This makes the display a lot more "cleaner" and organized.

CSS has been used in Kirrkirr to display the XML file only and this is really quite a disadvantage of the CSS version, as more JavaScript codes are needed to implement the other functions of Kirrkirr, such as the *Multimedia* panel. This will be discussed further in the later sections concerning JavaScript.

3.3 Extensible Styling Language (XSL)

3.3.1 Introduction to XSL

[XSL2000] XSL (Extensible Stylesheet Language) is being developed as part of the W3C Style Sheets Activity. It is evolved from the CSS language to provide even richer stylistic control, and to ensure consistency of implementations. It also has document manipulation capabilities beyond styling.

XSL consists of 2 main parts. The first part, *XSL Transformations* (XSLT), deals with the syntax and semantics for XSL, applying Stylesheets to transform one document into another. XSLT is designed for use as part of XSL, which is a Stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary. Meanwhile the second part is concerned with the XSL formatting objects, their attributes, and how they can be combined. The formatting objects used in XSL are based on prior work on CSS and DSSSL - the *Document Style Semantics & Specification Language*. XSL is designed to be easier to use than DSSSL, which was only for use by expert programmers.

A separate related specification is the *XML Path Language* (XPath). XPath is a language for addressing parts of an XML document, essential for cases where we want to say exactly which parts of a document are to be transformed by XSL. It allows us to, for example, “select all paragraph belonging to the chapter element”, or “select the elements called *special notes*”. XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSLT and XPointer.

3.3.2 Features of XSL

Unlike the case of HTML, element names in XML have no intrinsic presentation semantics. Without a Stylesheet, a processor could not possibly know how to render the content of an XML document other than as an undifferentiated string of characters. XSL provides a comprehensive model and a vocabulary for writing such Stylesheets using XML syntax.

XSL builds on the prior work on CSS and the DSSSL. While many of XSL's formatting objects and properties correspond to the common set of properties, this would not be sufficient by itself to accomplish all the goals of XSL. In particular, XSL introduces a model for pagination and layout that extends what is currently available and that can in turn be extended, in a straightforward way, to page structures beyond the simple page models.

Paging and Scrolling

Doing both scrollable document windows and pagination introduces new complexities to the styling of XML content. Because pagination introduces arbitrary boundaries (pages or regions on pages) on the content, control of spacing at page, region, and block boundaries become extremely important. There are also concepts related to adjusting the spaces between lines (to adjust the page vertically) and between words and letters (to justify the lines of text). These do not always arise with simple scrollable document windows, such as those found in today's browsers. However, there is a correspondence between a page with multiple regions, such as a body, header, footer, and left and right sidebars, and a Web presentation using frames. The distribution of content into the regions is basically the same in both cases, and XSL handles both cases in an analogous fashion.

XSL was developed to give designers control over the features needed when documents are paginated as well as to provide an equivalent "frame" based structure for browsing on the Web. To achieve this control, XSL has extended the set of formatting objects and formatting properties. In addition, the selection of XML source components that can be styled (elements, attributes, text nodes, comments, and processing instructions) is based on XSLT and XPath, thus providing the user with an extremely powerful selection mechanism.

The design of the formatting objects and property-extensions was first inspired by DSSSL. The actual extensions, however, do not always look like the DSSSL constructs on which they were based. To either conform more closely to the CSS specification or to handle cases more simply than in DSSSL, some extensions have diverged from DSSSL.

Some of the formatting objects and many of the properties in XSL come from the CSS specification, ensuring compatibility between the two.

There are four classes of XSL properties that can be identified as:

1. CSS properties by copy (unchanged from their CSS2 semantics)
2. CSS properties with extended values
3. CSS properties broken apart and/or extended
4. XSL-only properties

Selectors and Tree Construction

As mentioned above, XSL uses XSLT and XPath for tree construction and pattern selection, thus providing a high degree of control over how portions of the source content are presented, and what properties are associated with those content portions, even where mixed namespaces are involved.

For example, the patterns of XPath allow the selection of a portion of a string or the nth text node in a paragraph. This allows users to, for example, have a rule that makes all third paragraphs in procedural steps appear in bold. In addition, properties can be associated with a content portion based on the numeric value of that content portion or attributes on the containing element. This allows one to have a style rule that makes negative values appear in “red” and positive values appear in “black”. Also, text can be generated depending on a particular context in the source tree, or portions of the source tree may be presented multiple times with different styles.

An Extended Page Layout Model

There is a set of formatting objects in XSL to describe both the layout structure of a page or frame (size of the body, number of columns, headers, footers, or sidebars and the size of these) and the rules by which the XML source content is placed into these “containers”.

The layout structure is defined in terms of one or more instances of a “simple-page-master” formatting object. This formatting object allows one to define independently filled regions for the body (with multiple columns), a header, footer, and sidebars on a page. These simple-page-masters can be used in page sequences that specify in which order the various simple-page-masters shall be used. The page sequence also specifies how styled content is to fill those pages. This model allows one to specify a sequence of simple-page-masters for a book chapter where the page instances are automatically generated by the formatter or an explicit sequence of pages such as used in a magazine layout. Styled content is assigned to the various regions on a page by associating the name of the region with names attached to styled content in the result tree.

In addition to these layout formatting objects and properties, there are properties designed to provide the level of control over formatting that is typical of paginated documents. This includes control over hyphenation, and expanding the control over text that is kept with other text in the same line, column, or on the same page.

Comprehensive Area Model

The extension of the properties and formatting objects, particularly in the area on control over the spacing of blocks, lines, and page regions and within lines, necessitated an extension of the CSS2 box-formatting Smodel. The CSS2 box model is a subset of this model. The area model provides a vocabulary for describing the relationships and space-adjustment between letters, words, lines, and blocks.

Internationalization and Writing-Modes

There are some scripts that are typically set with words proceeding from top-to-bottom and lines proceeding either from right-to-left or from left-to-right. Other directions are also used. Properties expressed in terms of a fixed, absolute frame of reference (using top, bottom, left, and right) and which apply only to a notion of words proceeding from left to right or right to left do not generalize well to text written in those scripts.

For this reason XSL uses a relative frame of reference for the formatting object and property descriptions. Just as the CSS frame of reference has four directions (top, bottom, left and right), so does the XSL relative frame of reference have four directions (before, after, start, and end), but these are relative to the “writing-mode”. The “writing-mode” property is a way of controlling the directions needed by a formatter to correctly place glyphs, words, lines, blocks, etc. on the page or screen. The “writing-mode” expresses the basic directions noted above. There are writing-modes for “left-to-right - top-to-bottom” (denoted as “lr-tb”), “right-to-left - top-to-bottom” (denoted as “rl-tb”), “top-to-bottom - right-to-left” (denoted as “tb-rl”) and more.

Typically, the writing-mode value specifies two directions: the first is the inline-progression-direction which determines the direction in which words will be placed and the second is the block-progression-direction which determines the direction in which blocks (and lines) are placed one after another. In addition, the inline-progression-direction for a sequence of characters may be implicitly determined using bi-directional character types for those characters from the Unicode Character Database for those characters and the Unicode Bidi Algorithm.

Besides the directions that are explicit in the name of the value of the “writing-mode” property, the writing-mode determines other directions needed by the formatter, such as the shift-direction (used for sub- and super-scripts), etc.

Linking

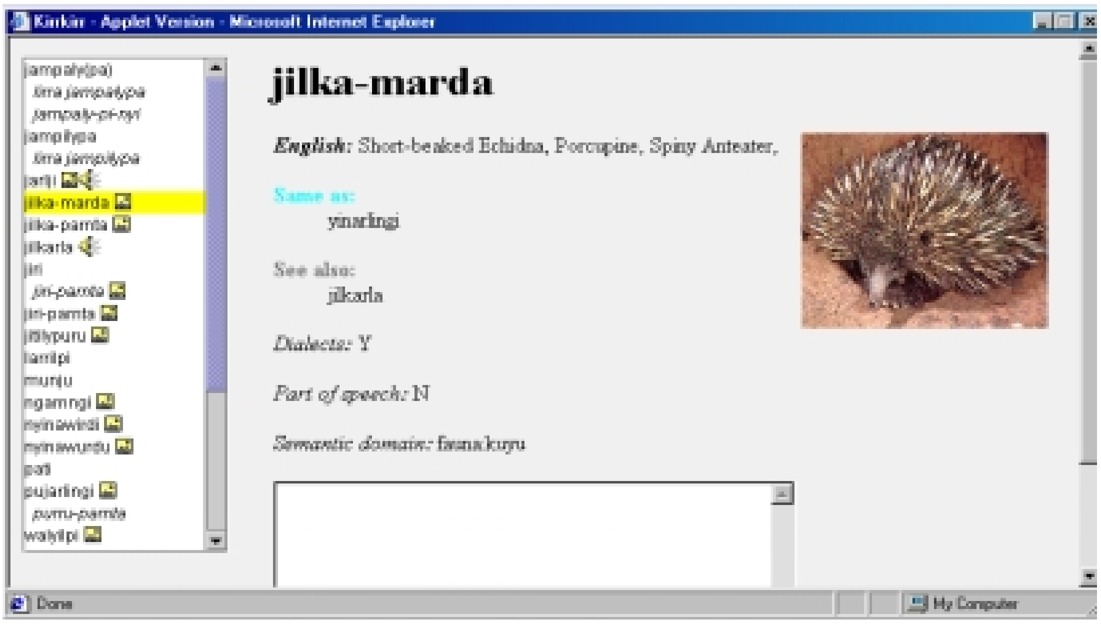
Because XML, unlike HTML, has no built-in semantics, there is no built-in notion of a hypertext link. Therefore, XSL has a formatting object that expresses the dual semantics of formatting the content of the link reference and the semantics of following the link.

XSL provides a few mechanisms for changing the presentation of a link target that is being visited. One of these mechanisms permits indicating the link target as such; another allows for control over the placement of the link target in the viewing area; still another permits some degree of control over the way the link target is displayed in relationship to the originating link anchor.

XSL also provides a general mechanism for changing the way elements are formatted depending on their active state. This is particularly useful in relation to links, to indicate whether a given link reference has already been visited, or to apply a given style depending on whether the mouse, for instance, is hovering over the link reference or not.

3.3.3 Using XSL in Kirrkirr

As mentioned earlier, there are many benefits of using XSL and thus, it is being used extensively in the web-based version of Kirrkirr. Similar to CSS, it is required to format and display the XML file but other than that, it also has the capabilities of using all of the HTML elements, such as <TEXTAREA> and displaying of picture files. There is also no need to include extra JavaScript functions to implement the *Multimedia* panel.



jilka-marda

English: Short-beaked Echidna, Porcupine, Spiny Anteater,

Same as:
yinarlingi

See also:
jilkarla

Dialects: Y

Part of speech: N

Semantic domain: fauna:kuyu

Figure 8. **A typical word entry displayed using XSL.**

The syntax of XSL is much more complex as compared to CSS, but on the other hand, this also makes it more powerful. Simple programming syntax, such as for-loops, is allowed. The example below shows a typical for-loop:

```
<xsl:template match="IMAGE">
  <xsl:for-each select="IMGI">
<P><IMG>
        <xsl:attribute name="src">../images/<xsl:value-of select="."/>
</xsl:attribute>
 </IMG></P>
  </xsl:for-each>
</xsl:template>
```

For XSL many “templates” are created and when called upon, it will just be substituted into that part of the document. The above example is simply to display all the images of the selected word onto the browser and to use it, we just need to use the following code:

```
<xsl:apply-templates select='IMAGE'/>
```

The ability to use simple programming syntax together with using HTML and JavaScript codes will allow the whole web page to be created from a single XSL file. Thus, this actually lets it have an edge over CSS.

3.4 CSS Vs XSL

XSL is designed to be used with XML so that XML can take advantage of the flexibility that comes from braking free of the HTML straight jacket. It uses XML syntax and creates its own flow objects, so it can be used for advanced formatting such as rearranging, reformatting and sorting elements. This enables the same XML document to be used to create several sub document views. XSL also adds provisions for the formatting of elements based on their content in the document and allows for generation of text and definition of formatting macros.

CSS is primarily a declarative language and apart from the ‘visibility’ and ‘none’ properties, allows little control over the content or the order of its appearance. It is also non-extensible by the user. CSS is also simpler to use and its declarative style is ideal for multimedia and HTML.

Chapter 4 Programming Language

4.1 JavaScript

4.1.1 Introduction to JavaScript

JavaScript is Netscape’s cross-platform, object-based scripting language for client and server applications. JavaScript lets us create applications that run over the Internet. Using JavaScript, dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases, can be created.

[DevEdge] Server-side and client-side JavaScript share the same core language. This core language corresponds to ECMA-262, the scripting language standardized by the European standards body, with some additions. The core language contains a set of core objects, such as the *Array* and *Date* objects. It also defines other language features such as its expressions, statements, and operators. Although server-side and client-side JavaScript use the same core functionality, in some cases they use them differently.

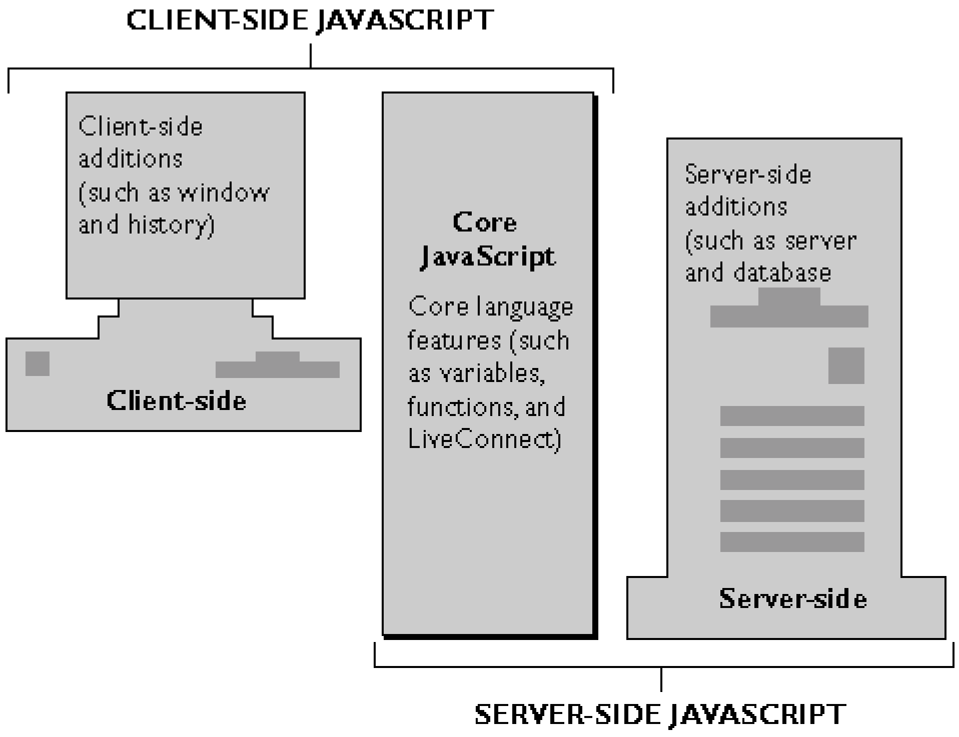

CLIENT-SIDE JAVASCRIPT
Client-side additions (such as window and history)
Client-side
Core JavaScript
Core language features (such as variables, functions, and LiveConnect)
Server-side additions (such as server and database
Server-side
SERVER-SIDE JAVASCRIPT

Figure 9. The JavaScript language

[Champeon1999] *Client-side JavaScript* encompasses the core language plus extras such as the predefined objects only relevant to running JavaScript in a browser. It is embedded directly in HTML pages and is interpreted by the browser completely at runtime. Because production applications frequently have greater performance demands upon them, JavaScript applications that take advantage of its server-side capabilities are compiled before they are deployed.

4.1.2 Features of JavaScript

[Flanagan1998] JavaScript is a general-purpose programming language and we can write programs in it to perform arbitrary computations, ranging from simple scripts that compute *Fibonacci* numbers, or search for primes to a program that can compute the sales tax on an online order by using HTML forms. As mentioned earlier, the real power of JavaScript lies in the browser and document-based objects that the language supports.

Control Document Appearance and Content

The JavaScript *Document* object, through its *write()* method allows us to write arbitrary HTML into a document as the document is being parsed by the browser. For example, this allows us to include the current date and time in a document or to display different content on different platforms.

The *Document* object can also be used to generate documents entirely from scratch. Properties of the *Document* object allow us to specify colors for the document background, the text, and for the hypertext links within it. What this amounts to is the ability to generate dynamic and conditional HTML documents, a technique that works particularly well in multiframe documents. Indeed, in some cases, dynamic generation of frame content allows a JavaScript program to replace a traditional CGI script entirely.

The new technology of Dynamic HTML is based on the ability to use JavaScript to dynamically modify the contents and appearance of HTML documents. Internet Explorer 4 and above supports a complete document object model that gives JavaScript access to every single HTML element within a document. In addition, Internet Explorer's DHTML capabilities allow JavaScript to modify the content of any element and to change the appearance of the element dynamically by modifying its style sheet properties.

Control the Browser

Several JavaScript objects allow control over the behavior of the browser. The Window object supports methods to pop up dialog boxes to display simple messages to the user and to get simple input from the user. This object also defines a method to create and open (and close) entirely new browser windows, which can have any specified size and any combination of user controls. This allows us, for example, to open up multiple windows to give the user multiple views of our web site. New browser windows are also useful for temporary display of generated HTML, and, when created without the menubar and other user controls, these windows can serve as dialog boxes for more complex messages or user input. The Window object allows JavaScript to display arbitrary messages to the user in the status line of any browser window.

JavaScript does not define methods that allow us to create and manipulate frames directly within a browser window. However, the ability to generate HTML dynamically allows us to write programmatically the HTML tags that create any desired frame layout. Yet another method of the Window object allows JavaScript to display arbitrary messages to the user in the status line of any browser window.

JavaScript also allows control over which web pages are displayed in the browser. The Location object allows us to download and display the contents of any URL in any window or frame of the browser. The History object allows us to move forward and back within the user's browsing history, simulating the action of the browser's *Forward* and *Back* buttons.

Interact with HTML Forms

[Neou1999] Another important aspect of client-side JavaScript is its ability to interact with HTML forms. This capability is provided by the *Form* object and the *Form* element objects it can contain: *Button*, *Checkbox*, *Hidden*, *Password*, *Radio*, *Reset*, *Select*, *Submit*, *Text*, and *Textarea* objects. These element objects allow us to read and write the values of the input elements in the forms in a document. For example, an online catalog might use an HTML form to allow the user to enter his order and could use JavaScript to read his input from that form in order to compute the cost of the order, the sales tax, and the shipping charge. JavaScript programs like this are, in fact, very common on the Web. JavaScript has an obvious advantage over server-based CGI scripts for applications like these: JavaScript code is executed on the client, so the form's contents don't have to be sent to the server in order for relatively simple computations to be performed.

Another common use of client-side JavaScript with forms is for verification of a form before it is submitted. If client-side JavaScript is able to perform all necessary error checking of a user's input, the CGI script on the server side can be much simpler and, more importantly, there is no round trip to the server to detect and inform the user of the errors. Client-side JavaScript can also perform preprocessing of input data, which can reduce the amount of data that must be transmitted to the server. In some cases, client-side JavaScript can eliminate the need for CGI scripts on the server altogether. On the other hand, JavaScript and CGI do work well together. For example, a CGI program can dynamically create JavaScript code on the fly, just as it dynamically creates HTML.

Interact with the User

An important feature of JavaScript is the ability to define event handlers - arbitrary pieces of code to be executed when a particular event occurs. Usually, these events are initiated by the user, when, for example, she moves the mouse over a hypertext link, enters a value in a form, or clicks the *Submit* button in a form. This event-handling capability is a crucial one, because programming with graphical interfaces, such as

HTML forms, inherently requires an event-driven model. JavaScript can trigger any kind of action in response to user events. Typical examples might be to display a special message in the status line when the user positions the mouse over a hypertext link or to pop up a confirmation dialog box when the user submits an important form.

Read and Write Client State with Cookies

"Cookie" is Netscape's term for a small amount of state data stored permanently or temporarily by the client. Cookies are transmitted to and from the server and allow a web page or web site to "remember" things about the client - for example, that the user has previously visited the site, or has already registered and obtained a password, or has expressed a preference about the color and layout of web pages. Cookies help us provide the state information that is missing from the stateless HTTP protocol of the Web.

When cookies were invented, they were intended for use exclusively by CGI scripts; although stored on the client; they could only be read or written by the server. The idea was to allow a CGI script to generate and send different HTML to the client depending on the value of a cookie (or cookies). JavaScript changes this because JavaScript programs can read and write cookie values, and as I noted earlier in this chapter, they can dynamically generate HTML based on the value of cookies. The implications of this are subtle. CGI programming is still an important technique in many cases that use cookies. In some cases, however, JavaScript can entirely replace the need for CGI.

What JavaScript Can't Do

Client-side JavaScript has an impressive list of capabilities. They are mainly confined to browser-related and HTML-related tasks. Since client-side JavaScript is used in a limited context, it does not have features that would be required for standalone languages:

- JavaScript does not have any graphics capabilities, except for the powerful ability to generate HTML dynamically (including images, tables, frames, forms, fonts, etc.) for the browser to display.
- For security reasons, client-side JavaScript does not allow the reading or writing of files. Obviously, we wouldn't want to allow an untrusted program from any random web site to run on our computer and rearrange our files.
- JavaScript does not support networking of any kind, except that it can cause the browser to download arbitrary URLs and it can send the contents of HTML forms to CGI scripts, email addresses, and Usenet newsgroups

4.1.3 Java vs JavaScript

JavaScript and Java are similar in some ways but fundamentally different in others. The JavaScript language resembles Java but does not have Java's static typing and strong type checking. JavaScript supports most Java expression syntax and basic control-flow constructs. In contrast to Java's compile-time system of classes built by declarations, JavaScript supports a runtime system based on a small number of data types representing numeric, Boolean, and string values. JavaScript has a prototype-based object model instead of the more common class-based object model. The prototype-based model provides dynamic inheritance; that is, what is inherited can vary for individual objects. JavaScript also supports functions without any special declarative requirements. Functions can be properties of objects, executing as loosely typed methods.

[Jaworski1999] Java is an object-oriented programming language designed for fast execution and type safety. Type safety means, for instance, that we can't cast a Java integer into an object reference or access private memory by corrupting Java bytecodes. Java's object-oriented model means that programs consist exclusively of classes and their methods. Java's class inheritance and strong typing generally require tightly coupled object hierarchies. These requirements make Java programming more complex than JavaScript authoring.

In contrast, JavaScript descends in spirit from a line of smaller, dynamically typed languages such as HyperTalk and dBASE. These scripting languages offer programming tools to a much wider audience because of their easier syntax, specialized built-in functionality, and minimal requirements for object creation.

JavaScript	**Java**
Interpreted (not compiled) by client.	Compiled bytecodes downloaded from server, executed on client.
Object-based. No distinction between types of objects. Inheritance is through the prototype mechanism and properties and methods can be added to any object dynamically.	Object-oriented. Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Code integrated with, and embedded in, HTML.	Applets distinct from HTML (accessed from HTML pages).
Variable data types not declared (loose typing).	Variable data types must be declared (strong typing).
Dynamic binding. Object references checked at runtime.	Static binding. Object references must exist at compile-time.
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.

Table 1. JavaScript compared to Java

4.1.4 Using JavaScript in Kirrkirr

JavaScript is being used extensively in the Web-based version of Kirrkirr. It has been used to replace the Notes and Multimedia panels. This section will talk about the technique used in Kirrkirr to communicate between the Java classes and JavaScript and then the different methods used to implement these two panels in JavaScript.

Controlling Java Applet from JavaScript

[Vacca1997] Controlling an applet with a script is fairly easy but it does require knowledge of the applet we are working with. Any public variable, method, or property within the applet is accessible through JavaScript. Next, we will name our applet in our <APPLET> tag. This is not really necessary but it makes our code more readable and the applet more accessible.

<APPLET CODE="*WordList.class*" NAME="wordlist">
</APPLET>

To use a method of the applet from JavaScript, use the following syntax:

document.wordlist.*methodOrProperty*

If the applet were not named, we would have to know the order in which the applets are called (when there are more than one applet) and use the following syntax:

document.applet[0].*methodOrProperty*

This method seems very simple and powerful but there is a downside to this implementation. The time taken by the JavaScript to pass the instruction to the applet is too long to be liked by users. Any methods called from the JavaScript will take about 5-10 seconds to be passed to the applet before it is actually being processed. This is not acceptable and thus we will avoid this as much as possible.

Controlling JavaScript from Java Applet

With the addition of a new set of classes provided with Netscape Navigator 3.0, Java can take a direct look at our HTML page through JavaScript objects. This requires the use of the *netscape.javascript.JSObject* class when the applet is created, thus we would need to import this package into our Java program.

An important addition is also necessary in the applet tag - *MAYSCRIPT*. This is a security feature, which gives specific permission for the applet to access JavaScript objects.

<APPLET CODE="WordList.class" NAME="*wordlist*" MAYSCRIPT>

Without it, any attempt to access JavaScript from the applet results in an exception. If we wish to exclude an applet from accessing the page, simply leave out the MAYSCRIPT parameter.

Using it in the program is also fairly simple; the following syntax shows an example of how to initialize it, retrieve an element from the HTML document and then to call a JavaScript function in the document.

Applet wl = getApplet("wordlist");

JSObject jsWin = JSObject.getWindow(wl);

String str = (String) jsWin.eval("top.main.notes.value");

jsWin.call("changestr", *args*);

There is really not much disadvantage in using this method of communication except for the fact that any carelessness in the initialization process will result in an exception. However this can be easily corrected and avoided. As for speed wise, there is no noticeable delay in the execution of the commands, so this is the method that will be mainly used in Kirrkirr.

Notes panel

In the original version of Kirrkirr, the users could edit and save their own notes to any dictionary entry, thus enabling them to remember the words with greater ease. This feature was then implemented using a Java class and loaded as a panel in place of the graphics panel. However in the new Web-based version, this can be replaced with a simple *TEXTAREA* element on the HTML file or Stylesheet used.

To load the notes onto the browser, the Java Window will call a JavaScript function on the browser to display the notes. The contents are passed as the arguments of the function. Saving the notes from the browser will be different, we do not invoke a Java method using JavaScript as its takes too much time, but rather, we would make the Java Window read in the value using the *eval()* method as mentioned earlier.

Multimedia Panel

Displaying pictures and playing audio files using the browser only involves the use of HTML tags **<IMG src="*imageFile*">** and **<A href="*audioFile*">**. Currently, browsers do not allow pictures to be loaded instantaneously but requires the page to be refreshed first. Thus, we need to pass the filenames of all the image and audio files to the browser before the page is loaded. Since the array structure of the Java and JavaScript programming language is different, we cannot pass an array of filenames. The Java Window will therefore pass a delimited string for the JavaScript function to process.

Chapter 5 Web Animation

5.1 Macromedia Flash

5.1.1 Introduction to Flash

Macromedia Flash is the key to designing and delivering low-bandwidth animations, presentations, and Web sites. It offers scripting capabilities and server-side connectivity for creating engaging applications, Web interfaces, and training courses. Once the content has been created, any online audience with the Macromedia Flash Player will be able to view it.

5.1.2 Features of Flash

Transmit structured XML information

[Macromedia] Web sites are evolving from static, brochure-ware based pages to data-rich, interactive Web applications. Transferring information to and from client-side applications to server-side architectures requires a hefty data conduit and must support a standard format for data exchange.

Macromedia Flash 5 allows developers to utilize XML structured data within their Macromedia Flash-based Web applications for a broad range of e-commerce purposes. Using XML for rich data and Macromedia Flash for logical and intuitive user interfaces, companies can create sales forms, virtual shopping carts, customer surveys, and stock availability matrixes. Continuous XML connectivity allows immediate updates of any mission-critical displayed information.

Incorporate HTML text formatting

Macromedia Flash 5 combines the best of both graphically engaging and traditional browser text display to provide a complete design and delivery solution for rich Web experiences. Designers now have a choice between anti-aliased display text or

HTML-rich text to create a new caliber of Web content. Primary HTML 1.0 text tags are supported and available during design time. Additionally, HTML-formatted files can be dynamically loaded during run-time for immediate text content updates.

Macromedia Flash Player

With Macromedia Flash 5, we can be assured that our content will play back consistently and reliably across a broad range of browsers, operating systems, and devices. Currently, more than 96.4% of online users can view Macromedia Flash content, without having to download a new player.

When authoring with Macromedia Flash 5, it's now even easier to deploy Macromedia Flash content with confidence:

- Publish with compatibility for any previous version of the Macromedia Flash Player.
- Macromedia Flash 5 contains tools to automatically export HTML detection scripts when publishing our movies, so that we can easily direct viewers to download the latest Macromedia Flash Player when necessary.
- we can download the free Macromedia Flash Deployment SDK with more advanced techniques including Macromedia Dreamweaver Objects, sample Macromedia Flash movies, and HTML/JavaScript code.
- When authoring with ActionScript, Macromedia Flash 5 will automatically color code Actions so that we know which areas of our scripts will have backward compatibility with earlier versions of the Macromedia Flash Player.

Distribution Partners

The Macromedia Flash Player is currently distributed with every major browser, operating system, and Web-enabled appliance. Partners include Microsoft, Apple, AOL, Netscape, Prodigy, WebTV, RealNetworks, and Excite@Home.

Develop customized applications for seamless, high-quality Web printing

Printing content on the Web today is a very limiting experience for both publishers and consumers. The print output of publishers' materials is inconsistent and varies by browser and platform.

Macromedia Flash 5 offers Web-native printing, providing customizable development options for publishers and positive viewing experiences for consumers. Macromedia Flash Web-native printing offers:

- WYPINWYS (What You Print Is Not What You See). Printable content can be downloaded on demand, providing a faster site-viewing experience.
- Consumers can view content and applications suitable for the screen and output a separate design suitable for print.
- No download necessary. Using Macromedia Flash Player, consumers have all the software necessary to view and print high-impact, high-quality content.

QuickTime Support

Macromedia Flash 5 offers native support for importing, extending, and exporting Apple QuickTime 4 movies. Additionally, Apple has licensed the Macromedia Flash Player and built it into QuickTime, allowing QuickTime movies to include Macromedia Flash graphics, animations, and interactivity. Together, the Macromedia Flash and QuickTime combination delivers the next level of streaming media on the Web.

Macromedia Flash 5 support for QuickTime 4 means that Web developers can easily incorporate compact, distinctive Macromedia Flash interface elements into QuickTime movies. Web users will be able to enjoy high-quality streaming video overlaid with Macromedia Flash interfaces that stream over the Internet to provide broadband-quality interactive Web experiences. Macromedia Flash effects that can enhance a QuickTime movie include navigational controls, text effects, animation, titling, and more.

RealPlayer Support

Macromedia Flash 5 provides animators and developers the ability to publish RealFlash content for the RealPlayer G2 and RealPlayer 7 and 8. Macromedia Flash 5 adds support in the Publish Settings to export all the necessary streaming RealAudio tracks, specially tuned Macromedia Flash movies to add seeking support, and SMIL code to ensure syncing playback in the RealPlayer.

Authors can select the SMIL settings, tuning bit rate, buffer time, and audio rates including SureStream technology for the best possible playback experience for various types of network connections. In addition, with the new support for Macromedia Flash 4 content in RealPlayer 8, Macromedia Flash developers can create not only engaging RealFlash animations but also compelling streaming e-commerce experiences, such as adding items to a shopping cart while watching a video about a product.

Create sophisticated Web applications

Sophisticated interactivity was first added to Macromedia Flash 4. Included were variables, conditional logic and run-time manipulation of object properties. The interfaces for applying this interactivity, or actions, were approachable for designers, but a bit limiting for serious programmers. The content developed with this level of interactivity was sophisticated and included Web application front-ends, games, mathematical modeling, and complex user interfaces.

Macromedia Flash 5 continues to maintain a familiar and approachable interface for interactivity while vastly expanding the level of tools for Web application development. The new tools give developers ways to create reusable interactive components and enable rapid development. Most notably:

- **ActionScript Syntax.** The scripting language has now been exposed and matches the syntax and structure of JavaScript.
- **ActionScript Editor.** The ActionScript panel provides both Novice and Expert

modes from drag-and-drop interactivity to full-blown text entry for ActionScript.

- **External Scripts.** All ActionScript syntax can be exported to an ASCII file editable within any external text editor then re-imported into a Macromedia Flash authoring document.
- **Debugger.** Isolate ActionScript variables to debug complex applications during development.

Smart Clips

Smart Clips abstract complex interactivity into parameterized movie clips to share and reuse across a site, several sites, team members, or the Macromedia Flash community. Smart Clips are essentially special intelligent movie clips that can act upon the parameters associated with them, and can be created and utilized in the Macromedia Flash 5 environment.

Create re-usable elements for designing user interfaces, building applications and more. Smart Clips can be used to create checkboxes, radio buttons, pull-down menus, navigational menu systems and learning interactions.

- **ActionScript—**New, powerful ActionScript functions, properties, and built-in objects, such as hitTest for automatic collision detection.
- **Clip Actions—**Event Actions attached to a movie clip, such as mouse and keyboard events
- **Clip Parameters—**Parameters that are associated with an instance of a movie clip, such as text for a pull-down menu
- **Custom UI—**The option to use a Macromedia Flash movie within the authoring environment to define the parameters for a Smart Clip, such as a visual menu builder that shows us a preview of the menu as we add parameters

Flash-based forms

Macromedia Flash lets us integrate field entry forms for data gathering and e-commerce applications. Macromedia Flash Player can pass information to a Web server so that we can use input text fields for password fields. All text entered into password-specified text fields will automatically convert characters to an indistinguishable form.

Create Web application front-ends using Get and Post actions to place text from and to a Web server easily. Pass data to any CGI script for closer integration with Microsoft Active Server Pages, Allaire ColdFusion, or Macromedia Generator Enterprise Edition server.

Bandwidth Profiling

Macromedia Flash provides a graphical representation of how a single scene or entire movie is streaming. Adjust for a target modem speed, such as 28.8, and the Bandwidth Profiler will preview and test movies to find potential problem areas. Bandwidth profiling enables us to quickly fix and optimize movies so that we can deliver a fast-loading, smooth-playing Web experience.

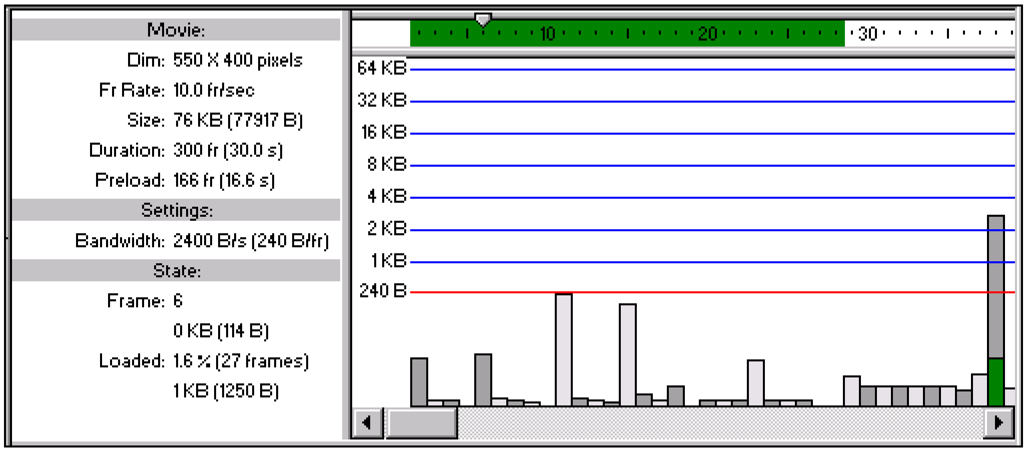
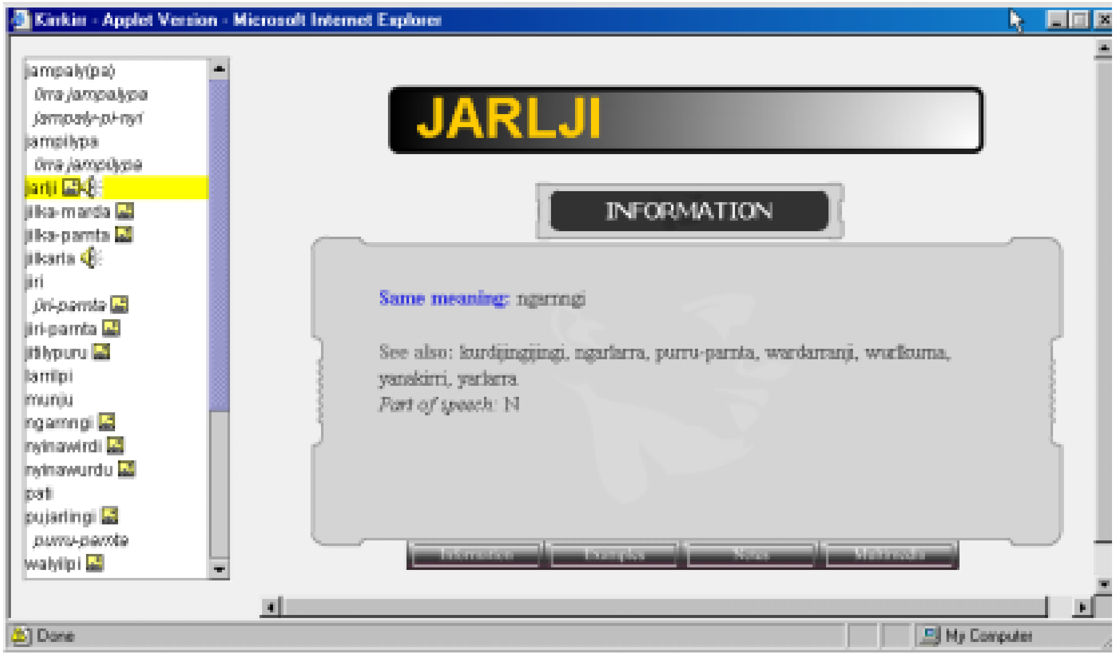

Movie:
Dim: 550 X 400 pixels
Fr Rate: 10.0 fr/sec
Size: 76 KB (77917 B)
Duration: 300 fr (30.0 s)
Preload: 166 fr (16.6 s)
Settings:
Bandwidth: 2400 B/s (240 B/fr)
State:
Frame: 6
0 KB (114 B)
Loaded: 1.6 % (27 frames)
1 KB (1250 B)

Figure 10. Previewing a movie with the Bandwidth Profiler.

5.1.3 Using Flash in Kirrkirr

In Kirrkirr, Flash has been used to take over the Stylesheets in handling the XML files, displaying the pictures and playing the sounds of the dictionary entries and also implementing the Notes panel. In addition, Flash also provides the user with a more enriching audio and visual experience as it can include animations and sounds onto the webpage. All the browser needs is the Macromedia Flash Player, which is readily downloadable.



Same meaning: ngarrngi

See also: kurdijingijingi, ngarlarra, purru-parnta, wardaranji, wurlkuma, yarakirri, yarlarra
Part of speech: N

Figure 11. **A typical word entry displayed using Flash.**

In this section, we will not focus on the creation of the animations and sounds in Flash as it is rather trivial and simple as the software is really user friendly and easy to use. Another reason is that there are not many animations added into this version of Kirrkirr and this is mainly due to time constraints and also the aim of the project is to explore the capabilities of the IE and not just beautifying it with Flash. Instead, we will focus on the ActionScript part of interaction between JavaScript and Flash and also how Flash handles the XML files.

Controlling JavaScript from Flash

[Hartman2001] Flash attempts to send a message to JavaScript whenever the movie playback head hits a keyframe on which an FS Command is set, or when a button event with an FS Command action occurs. When the browser receives an FS Command from Flash, it checks if there's a matching JavaScript function (or VB Script in the case of IE) that can "catch" the command. If there is, it starts the function, sending two parameters to it - the values of the "Command" and "Arguments" fields from the Flash FS Command action dialog.

There are a few things that are required to be done for this method to work. First, name the Flash Object.

```
<OBJECT ID="kkf" …..>
        <EMBED>
                NAME="kkf"
                ………..
                ………..
        </EMBED>
</OBJECT>
```

Then the VB Script to "catch" the command must be added in the HTML file.

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub kkf_FSCommand (ByVal command, ByVal args)
   call kkf_DoFSCommand(command, args)
end sub
//-->
</SCRIPT>
```

Next, add in the JavaScript function to process instruction.

```
<SCRIPT LANGUAGE="VBScript">
<!—
function kkf_DoFSCommand(command, args) {
    if (command == "saveNotes") {
       ………………..
    } else if (command == "multimedia") {
       ………………..
    } else  if ……..
      ……………..
    }
//-->
</SCRIPT>
```

Finally, to call any of the defined functions from Flash, just add the following syntax into the ActionScript.

```
FSCommand("saveNotes", "some notes to save");
```

After that the VB Script will catch it and pass to the JavaScript to process.

This is a very powerful way to interact with JavaScript, as anything that JavaScript can do is now available to Flash. All we need is to add in a couple of lines of codes and we can control JavaScript functions using Flash.

Controlling Flash from JavaScript

JavaScript can send commands to Flash by invoking built-in methods on embedded movie objects. Calling Flash methods works exactly like all calls to built-in methods on JavaScript objects (eg. document.write() or window.close()). From a developer point of view, this direction of communication is much more one sided than the opposite direction--most of the simple JavaScript commands that control a Flash movie can be carried out entirely in JavaScript, without doing anything to the Flash movie.

Similarly, the Flash Object must be named. After that, use the following syntax that to send a command to Flash.

window["*FlashName*"].SetVariable("xmlfile", "jiri.xml");
window["*FlashName* "].TGotoFrame("_level0/", 2);

The above 2 lines are just some examples of the functions that can be called as there are actually a whole list of built-in methods that can be invoked. Since it is quite a long list, we shall not mention it here. For more information on the rest of the available methods, please go to the Macromedia Website at
http://www.macromedia.com/support/flash/ts/documents/f4_scripting_methods.htm

Using XML in Flash

With the addition of the XML Object to the ActionScript in Flash 5, we can now use the methods and properties of the XML Object to load, parse, send, build, and manipulate XML document trees. The constructor **new XML()** must be used to create an instance of the XML object before calling any of the methods of the XML object. The actual coding for the program is really quite tedious and complex as when the XML tree gets complex with many levels, keeping track of the different nodes becomes difficult. The XML Object in Flash is still very new thus it only includes the most primitive and essential commands. The 4 main ones used to parse the XML file are as follow:

1) **xmlObject.firstChild**
This returns the first child node of the xmlObject.
2) To get to the next node, use **xmlObject.nextSibling**
3) To retrieve the value of the node, use **xmlObject.nodeValue**
4) Lastly, we would sometimes need to know all the child nodes of an xmlObject and so we can use **xmlObject.childNodes** to get an array of all the child nodes of the xmlObject.

These are the main methods that we used in Kirrkirr as we only need to parse the XML files in our program.

Chapter 6 Evaluation

6.1 Program Performance Evaluation

This section compares the performance of the new version with original version. The tests in this section were conducted on an AMD K62-450 machine with 64 Mbytes of RAM and running JRE 1.3.0, Internet Explorer 5.5 and with Microsoft XSL Parser installed.

Kirrkirr Version	**Startup Time**	**Size of click.jar**
Original	7250 ms	147 KB
New	3240 ms	128 KB
Percentage Improved	55 %	13%

Table 2. Comparison between the original and new versions of Kirrkirr

6.1.1 Startup Speed and Program Size

Table 2 summarizes the comparison of the start-up time between the original and new versions of Kirrkirr. In the original version, the HTML, Notes and Multimedia panels need to be loaded but in the new version, they are not included as they are implemented by exploiting the features of the browser. Although the difference in the startup times on a fast machine is not significant, but we can tell from the percentage improved that these three classes actually takes up a big bulk of the total loading time. The difference in the size of click.jar (which is the compressed file containing of the classes used by Kirrkirr) is also reduced as quite a number of classes have been removed.

6.1.2 Other Improvements

There are also other improvements, which cannot be measured by statistics. For example, in the original version, all the display is done in the Java Window and we cannot provide the user with much visual entertainment or satisfaction without going through lots of coding. However, in this new version, we can make use of any technologies that are supported by the web browser, in this case, the use of Stylesheets and Macromedia Flash 5. Since these technologies are built to work with web browsers, they can be utilize much fully by the browsers than the Java program, and hence should perform much better in terms of speed and stability.

From the programmer's point of view, the use of the web browser to handle some of the functions of the program will greatly simplify the development of the code. The programmer can then focused more on the interaction between the program and the browser, which is a fairly simple task, rather than to go around looking for packages and drivers or even write their own codes in order to make use of the latest technologies. The only disadvantage of this approach is that testing and debugging will be more tedious as the errors could come from any of the components used, and also when deploying the program for implementation, all of the components must be checked to be configured correctly.

6.2 Stylesheet Performance Evaluation

Since this project is to optimize Kirrkirr to work on Internet Explorer, there is more support for XSL as compared to CSS. Without support for the latest CSS2 Specifications, many things cannot be done. For example, we cannot add in words in front or after the XML element to make the display more readable to the reader. CSS also does not support HTML elements and this means we cannot use it to display pictures and create links or form elements such as *<BUTTON>* and *<TEXTAREA>*. It does not even support the use of JavaScript. All of the above makes it very inflexible, as it is only capable of formatting the XML document and displaying in the style we wanted. To CSS, it seems that the XML file is the main focus and the Stylesheet file is just there to assist the browser to decide how to display the text to the user.

On the other hand, for XSL, the main focus is on the Stylesheet file. The Stylesheet file contains all the essential elements of a well-formed webpage as it can use anything available to the browser like the HTML format. These things include all HTML elements, JavaScript, Flash and many other more. The browser actually looks at the

Stylesheet and then brings in the necessary XML elements as and when they are needed. This definitely makes XSL much more flexible than CSS.

Of course, with such flexible also comes a price – complexity. The coding for XSL is rather complex as it requires a certain degree of programming skills and the mixture of the different codes (JavaScript, HTML and XSL) makes it more confusing. CSS only deals with displaying the text so coding is very clear-cut and simple thus it is actually a much cheaper alternative if text is the only thing that needs to be displayed.

6.3 Macromedia Flash Performance Evaluation

Other than using Stylesheets to handle and display the XML files, another way is to use the scripting language of Macromedia Flash 5 – ActionScript. With the addition of XML handling functions in the latest version of Flash, we are able to create an interactive webpage with animations and sounds and at the same time, making use of our XML data. This greatly increases the possibilities and number of ways in which we can enhance the display and give users a better visual satisfaction.

There are, however, disadvantages too. The ActionScript support for XML is still in its early stages and it is still quite buggy and not as easy to use when compared to Stylesheets. A good example would be that the root element of the XML file has got to be in the first line of the file, otherwise the program cannot access it. Another problem is that the Flash format needs to be compiled before use, thus unlike Stylesheets, which are text files, changes to the Flash program is not as easy as to a Stylesheet. Lastly, the syntax for the ActionScript to access XML elements is very complex and coding is very tedious.

Chapter 7 Future Work and Conclusion

7.1 Future Work

At present, only 3 of the panels are handled by the browser. Most of the work is still done by the Java Window and the main reason for this is actually due to the lack of support for dynamic graphical display on the browser. Even with the latest Dynamic HTML technology, there is still no way to create moving boxes and lines efficiently on the browser. Therefore, the graphical will definitely have to stay in the applet form.

However, for the Search panel, we may still be able to bring it out. It may not be easy as there are some complex elements to implement using HTML such as sliding bars. Maybe with the help Flash's graphical capabilities, we can actually implement it. Another consideration will be its search algorithm. It may not necessarily be easy to implement in Flash's ActionScript.

In order to exploit the browser, we can also try to convert some of the Java codes into JavaScript and let the browser handle some of the background processing work. The modules that can be converted should mainly be those doing calculations and searching but not those involving write operations to the disk as JavaScript has many security limitations. Thus there is still no way to totally get rid of all the Java and use JavaScript as long as disk I/O operations are involved.

Last but not least, we can aim to improve Kirrkirr visually by making use of the Flash animations. Since we can already incorporate Flash into Kirrkirr, we can certainly make more use of its animation capabilities if given more time to come up with a good design.

7.2 Conclusion

In this project, we have successfully investigated the different ways of handling XML files, more specifically, using XSL, CSS and Flash. Each of these 3 ways have their own pros and cons and depending on the type of application that is involved. CSS is used for simple formatting and displaying of XML files where the main focus is on the information and not much graphics are needed. XSL is a bit more powerful in the sense that, other than interfacing with the XML files, it also supports all the elements supported by the HTML specifications, such as HTML form elements, images and links. It is suitable for all generic webpages and it is not too complex also. As for Flash, it is the most powerful of the 3. Other than being able to do the things the other 2 can, it also allows animations and audio clips to be used in the interactive webpage. This is more suitable for multimedia-based applications.

Another aspect of the browser capabilities investigated is the use of JavaScript. JavaScript is getting more and more powerful and it has now got the ability to interact with Flash and Java Applets through simple means. This makes applications very easily upgradeable as new modules of Flash movies or Java Applets can be added and controlled. By taking some of the classes from the Java side, we can effectively reduce the program size and startup time of the program.

In summary, the efforts undertaken for Kirrkirr to exploit the capabilities of the browser, specifically IE5, have indeed yielded performance improvements and, together with the newly added features provided by Flash, have resulted in a more enjoyable Kirrkirr experience for its users.

Chapter 8 References

Ben Shneiderman, University of Maryland. 1998, *Designing the User Interface. Strategies for Effective Human-Computer Interaction.* Third Edition, Addison-Wesley Longman, Inc.

Cascading Style Sheets, level 1. *W3C Recommendation 17 Dec 1996, revised 11 Jan 1999.*

At: http://www.w3.org/TR/1999/REC-CSS1-19990111

Cascading Style Sheets, level 2. CSS2 Specification. *W3C Recommendation 12 May 1998.*

At: http://www.w3.org/TR/1998/REC-CSS2-19980512

David Flanagan. 1997*, JavaScript: The Definitive Guide, 3rd Edition*. Oreilly Books

DevEdge Online – JavaScript Reference Manual. Netscape.
At: http://developer.netscape.com/docs/manuals/communicator/jsguide4/index.htm

Extensible Stylesheet Language (XSL) Version 1.0. *W3C Candidate Recommendation 21 November 2000*
At: http://www.w3.org/TR/2000/CR-xsl-20001121/

Frank Boumphrey. 1998, *Professional Style Sheets for HTML and XML*, Wrox Press Ltd.

James Jaworski. 1999*, Mastering JavaScript and JScript*, Sybex Inc.

J. R. Vacca 1997, *JavaScript development*, AP Professional, Boston.

Jansz, K. 1998, *Intelligent processing, storage and visualisation of dictionary information*, Honours Thesis., Department. of Computer Science, University of Syndey.

Jansz, K.; Manning, C. and Indurkhya, N. 1999, *Kirrkirr: Interactive Visualisation and Multimedia from a Structured Warlpiri Dictionary*, Ausweb 1999.

Macromedia Flash TechNotes
At: http://www.macromedia.com/support/flash/

Molly E. Kolzschlag. 1998, *Web by design. The Complete Guide*, Sybex Inc.

Patricia Hartman. 2001, *Flash 5: Visual Jumpstart*, Sybex Inc.

Simon Laurent & Ethan Cerami. 1999, *Building XML Applications*, McGraw-Hill Companies, Inc. Chapter 8: Styles: Presenting Information

Steven Champeon & David S. Fox. 1999, *Building Dynamic HTML GUIs*, IDG Books Worldwide, Inc. Chapter 6: Client-side Scripting

Susan Weinschenk, Pamela Jamar & Sarah C. Yeo. 1997, *GUI Design Essentials*, John Wiley & Sons, Inc.

Vivian Neou. 1999, *HTML 4.0 CD with JavaScript*, Prentice-Hall, Inc. Chapter 13: Embedded Objects: Java and ActiveX