Nanyang Technological University

“XUL Application Development”

Jessica Halida Harjono

School of Computer Engineering
2001

Nanyang Technological University

SCE00-087

"XUL Application Development"

Submitted in Partial Fulfillment of the Requirements
For the Degree of Bachelor of Applied Science
of Nanyang Technological University

by

Jessica Halida Harjono

School of Computer Engineering
2001

Abstract

Extensible User Interface Language or XUL is recently being a member of GUI world. The XML-based User Interface Language, or XUL (pronounced "Zuul"), is an implementation of XML used for the graphical components of a computer program.
A GUI is most everything we see on modern computer screens, such as buttons, tabs, menus, sliders, windows, file listings, and more. The significance of XUL is that it allows rapid GUI development and a high level of customisation possibility. XUL was developed by Mozilla, and will be used throughout their upcoming product. With the entire GUI of browser written in XUL, it can actually become part of the web browsing experience. One possible implementation would be that Internet service providers could modify the browser GUI to integrate with their services to a great extent.
Web sites could also dynamically modify the XUL GUI as we browse, offering a more integrated experience. An example could be a web site map temporarily displayed in tabbed-window as part of the web browser. It is impossible to tell if others in the software industry will adopt XUL, as it is only a specification currently in use by Netscape and affiliates. However, XUL is a very good example of XML being used beyond an Internet web page.
By combining the GUI Design guidelines and Style Sheet Design guidelines, this paper suggest a guide for efficient interface design on browser (exclusively in Mozilla) application.

Acknowledgements

Hereby, I like to present my sincere appreciation, first of all, to Dr. Nitin Indurkhya, for his patience, advice, direction, careful guidance and counseling. He had assisted me to quest the information imperative for my project and remind me to be discipline to my timetable in completing my project, which had been an essential contribution in order for me to accomplish my project on time.

Secondly, I am very grateful to my senior colleague, Evelyn Kurniawati, and my classmate, Nadia Nalaningrum who has provided me the hardware, software and application necessary for my project as well as assisting me in learning Java programming.

At last, very special thanks to Yusdi Santoso, who was mostly sustained me in doing my project and has given me courage to complete my work.

Contents

Chapter 1 : Introduction

Application is software that performs a specific task or function, such as word-processing, creation of spreadsheets, generation of graphics, facilitating electronic mail, *etc.* There are local application (run on your desktop) and the online application (run on the net). The local application usually runs faster and since it is supported almost by any kind of language, local application is able to render almost every kind of complex and complicated algorithm. For example, on this project we make use of Java language to support the dynamic movement/animation of the graph panel, which is required to accurately compute the distance among nodes.

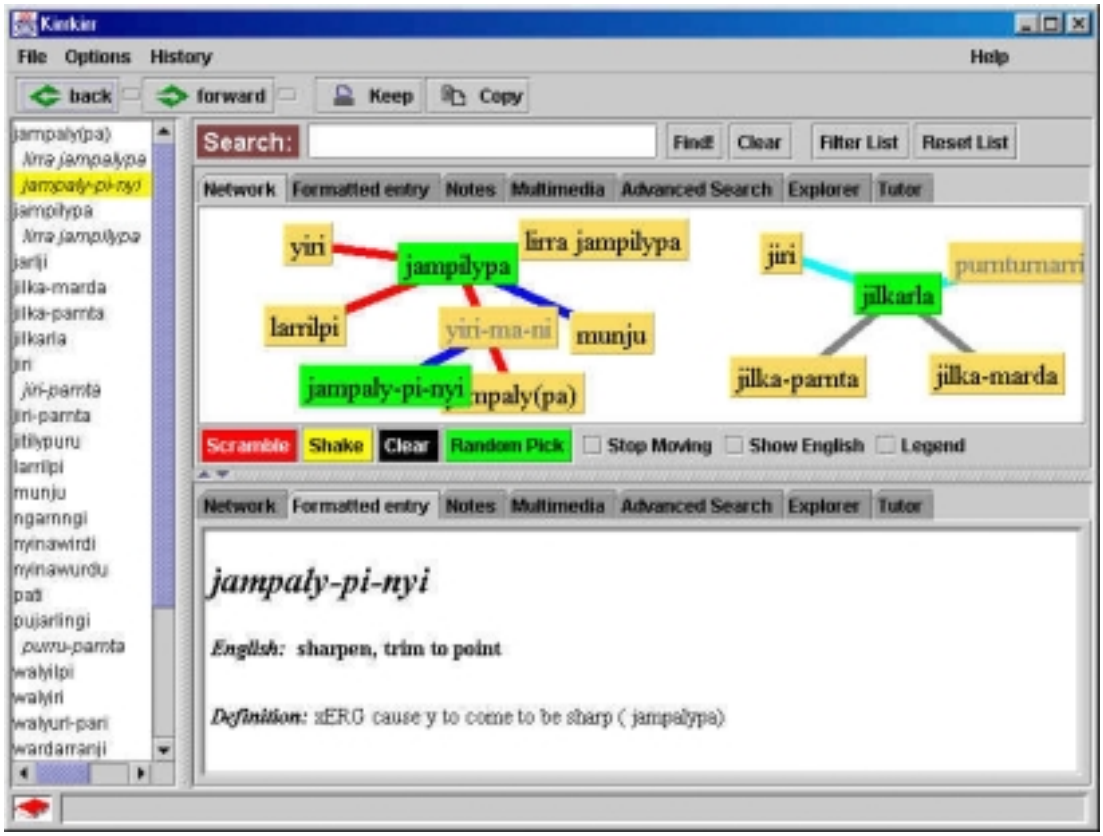


Figure 1.1 Snapshot of Kirrkirr (Aboriginal Dictionary) as one embedded applet, example of online application, base on Java language

There is an obligation of launching the application as online application so that everybody can easily get accessed to it without installing the application on his or her own PC. But, to port existing application into browser enable utilisation, of course, there is another consequence in exchange.

Before moving further on, the following is some related issue with Web Pages presentation. Cascading Style Sheet, that is a more efficient way to display pages. It gives designers far more control than slower, clumsier hacks like tables, spacer GIFs, and <FONT> tags. The CSS-1 spec covers style sheets for things like text and block formatting. It also lays out the syntax for setting CSS attributes and marking up content. CSS-P, or positioning, is part of CSS-2, as well as visibility and clipping, and Navigator 6 supports these components.

DOM-1 defines the structure of HTML and XML documents and is accessed by JavaScript or Java. It defines the web page through a tree structure, in which each object is a node that can be accessed and modified; letting users do things like image rollovers. DOM-2, soon to be a final spec, adds CSS to the document structure, allowing JavaScript to change the style sheet code for truly dynamic HTML.

People try to create a technology that is able to combine CSS, DOM with some scripting to create powerful browser. Then they come out with HTML 4.0, which moves toward separating structure and presentation. It was the advancement over HTML 3.2 (supported since Navigator 3 and Explorer 3), and provides better mechanisms for CSS and scripting on a Web page, multi-language support, and various refinements to tables, forms, and image maps. Column groups and other new attributes give developers more power over table structure and allow them to render incrementally rather than all at once, which speeds up rendering.

So, why don't we actualise all applications on the browser? There are some restrictions, consider some cases below :

- Browser, which is only supported by HTML and some scripting language (such as JavaScript and VB Script even Active Server Pages) usually will be able to carry out simple task/computation in comparison with Java or C. For example, doing merge sort and database searching is more efficient in C code rather than Java Script.
- But, introducing the entire/some portions of Java or C class file may cause another problem arise, such as browser timed out (the application too big that take sometime to load), security exception, or may require user to set up certain plug-ins.

Since implementing the entire application using only HTML with additional scripting language (even with XUL) sometimes is not possible. Foremost, we need to know what kind of tasks on the application, which are more efficient to be accomplished in regard of browser capability. Mainly, the application implementation can be undertaken by replacing those

above-mentioned parts (with browser scripting language, widget or object model) and let the remaining as its original language.

The objective of this project is to port an existing application (in this case : Kirrkirr is Java base application) into browser specific environment (Mozilla) by maximising the usage of browser features (the main feature of Mozilla is to display XML interface markup language, XUL).

Although when Kirrkirr was formerly implemented as single applet class file, it's size is relatively bigger than incorporation between Java Script, XUL and seven java applets, but the main restriction on mix language application is how to synchronize and maintain interaction among different languages.

Other problems are the security and speed problem, that is user may need to pause upon the applets loading time as well as experiencing some delay as java method try to communicate with the GUI part (which is build in XUL) via LiveConnect and JavaScript.

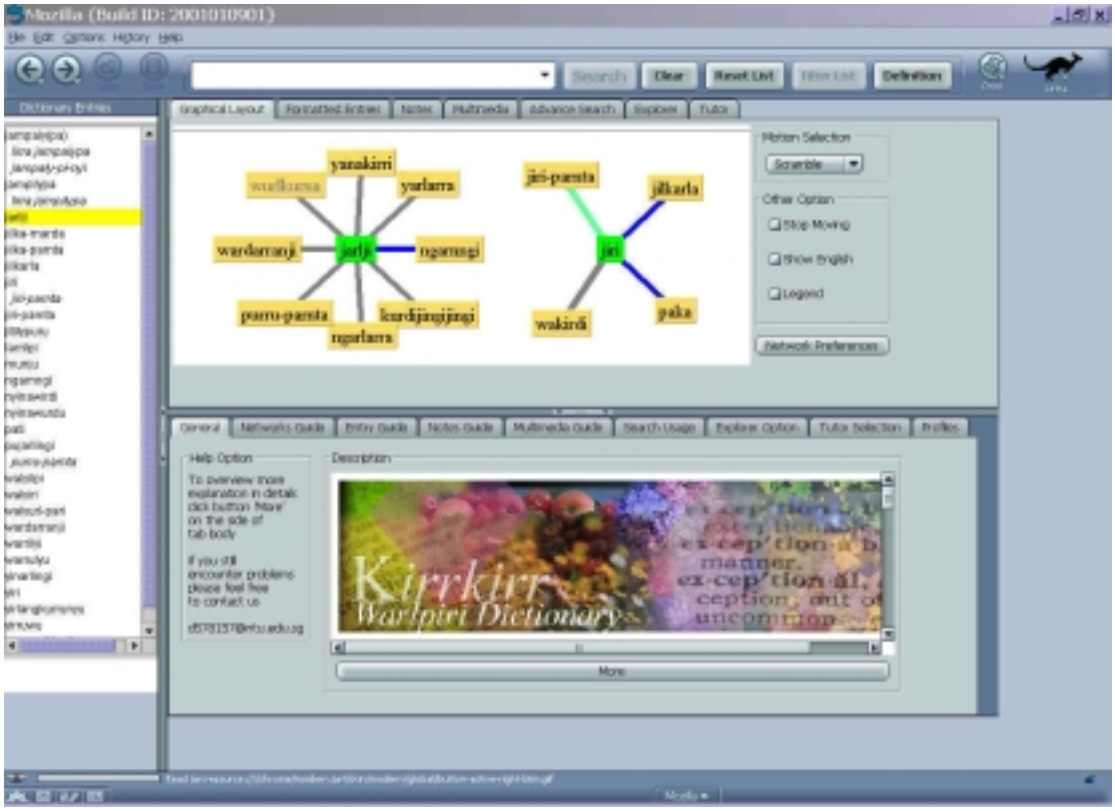


Figure 1.2 Kirrkirr on Mozilla browser. It has been constructed by combination of Java applets, XUL as well as JavaScript

In order for the work simplicity, we decided to try Kirrkirr and Mozilla locally. User needs to download a zip file (consist of executable code) and instead of opening a link/url using Mozilla, they redirect the browser to point on local file.

1.1 Mozilla Browser and Features

Mozilla is industry-leading support for HTML 4.0, XML, CSS1, DOM1, and RDF, plus support for features of CSS2 like content positioning and features of DOM2, such as the CSS interface and event model. Meant that for the first time, it would be possible to build the entire user interface for desktop application using web standards. XML and HTML could structure the user interface, CSS could format it, the W3C DOM could handle events and expose the UI elements to script control, RDF could represent collections of resources like bookmark lists and email in-boxes, and JavaScript would tie it all together.

Since Mozilla Milestone18 supports XML, it also incorporates other standards that lay the groundwork for the exchange of data and the extensibility of the browser. XML is gradually

emerging as a standard for creating documents that work across various media. Further, it includes XUL (XML-based user interface language), a DTD (document type declaration) for XML. XUL defines the various tags used to create the buttons and the rest of the browser interface, letting the programmers to customise the browser and appropriate these features for their own web pages.

Essentially it provides a language to describe a user interface, with many more widgets than are provided in HTML itself. Such widgets include tree controls, scrollbars, and splitters. A user interface description ("package") contains various elements that control the interface appearance and behaviour:

- Content: the description of the widgets in an interface
- Appearance: CSS styles dictate the appearance of the widgets
- Behaviour: specified either in JavaScript or via native code (C++)
- Locale: interface localisation information
- Platform specific information

XUL is implemented from within the Mozilla browser: it is in this framework that XUL application run.

[Krock, 2000]*The use of XUL has several benefits. In addition to being cross-platform, it also has the potential to define cross-device interfaces.* XUL can be used to build small quickly downloadable applications, leveraging the power of the browser environment; and finally, it adds to web browsers UI elements that HTML does not provide. He also mentioned that XUL has the potential to "democratise" application interface design, bringing it further into the grasp of a designer accustomed to HTML and JavaScript.

1.2 First Consideration of Developing Application on Mozilla

Mozilla is an open-source web browser, designed for standards compliance, performance and portability. It co-ordinates the development and testing of the browser by providing discussion forums, software engineering tools, releases and bug tracking. “Mozilla" was the original code name for the product that came to be known as Netscape Navigator, and later, Netscape Communicator.

On the first move, because Mozilla has inaugurated different releases, certainly meaning there are some minor differences in every Mozilla releases' capability. This issue is regardless of Mozilla main property on supporting XUL as its distinct characteristics compare with IE or Netscape Communicator; most of Mozilla releases currently support XUL. Consequently, first consideration of developing Application on Mozilla is "What version of Mozilla to be used?"

Eventually this Kirrkirr project was switched from using Netscape 6 to Mozilla 0.7 as its browser. As this project aimed to make use of the all features of the browser, XUL ability of creating GUI has been the focus of our interest. Since Kirrkirr is Java based application, it may need LiveConnect bridged the communication between Java method and GUI component (XUL). Therefore, since LiveConnect becomes essential, Netscape 6 and previous version of Mozilla (0.6 and below) with disabled LiveConnect are incompatible.

As this project proceeds to the end of its implementation in Mozilla 0.7, Mozilla 0.8 was released. This version of Kirrkirr is compatible with Mozilla 0.8, but since there are some minor differences in term of alignment and positioning the widgets, we decided to make use of Mozilla 0.7.

Hereby the road maps of Mozilla, open source project to see the development position of netscape 6 and Mozilla milestone 18 or Mozilla 0.7.

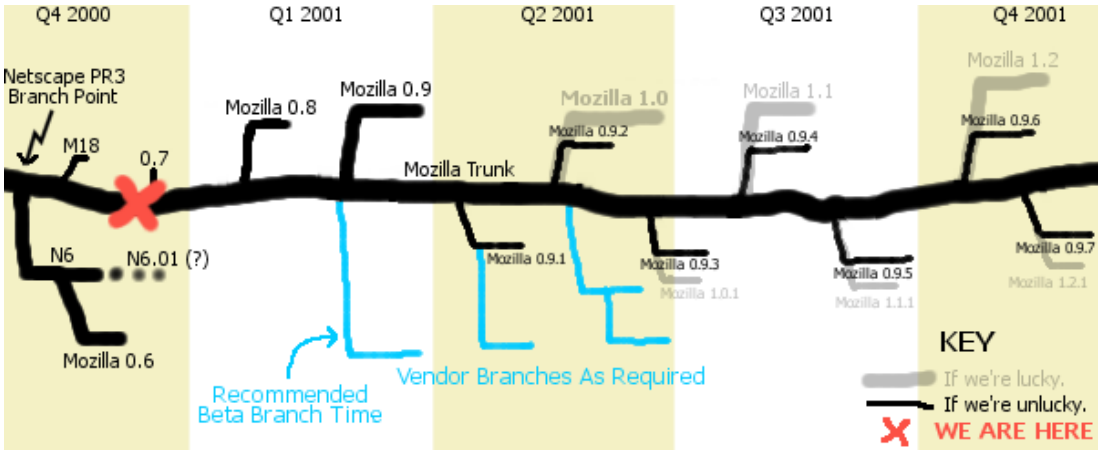


Figure 1.3 Mozilla Roadmap

[Bredan Eich, 2000] *Going forward, as netscape.com prepares to ship a commercial browser based on Mozilla, therefore it separated to other branch. Mozilla needs a roadmap*

that prescribes fewer technical points and more planning and scheduling techniques. Mozilla doesn't need new features, or any particular "new" or "next generation" module. Mozilla needs performance, stability, and correctness.

Figure 1.4 Netscape 6 and Mozilla trunk

The Netscape 6 branch is shown here starts with the PR3 (`nsbeta3`) branch-point. Mozilla will release branded M18 milestone that is built based on the trunk, soon after the time Netscape releases PR3.

The Mozilla 0.6 branch exists only to support MathML and other extensions or "not-quite-plugins" that may depend on Gecko internal APIs.

As this Kirrkirr project mainly needs LiveConnect and Java plugin, it is important to keep track with every releases of Mozilla. Hereby the Milestone plan of Mozilla taken from Mozilla.org:

Table 1.1 Mozilla Release Date

Milestone	Start	Freeze	Branch	Release	Key Features / Highlights	Comments
M18				Oct 12, 2000 (completed)		Work with Kirrkirr

Mozilla 0.6				Dec 6, 2000 (completed)	- Basis, or alias, of Netscape 6.0 RTM - MathML, other extensions	LiveConnect disabled (not work with Kirrkirr)
Mozilla 0.7				Jan 9, 2001 (completed)	"Best of December 2000" release	Work well with Kirrkirr
Mozilla0.8	Jan 3, 2001	Feb 7, 2001	Feb 9, 2001	Feb 12, 2001		Work with Kirrkirr
Mozilla0.9	Feb 9, 2001	Mar 14, 2001	Mar 16, 2001	Mar 19, 2001	"Beta release" target for embedders	
Mozilla 0.9.1 (or Mozilla 1.0)	Mar 16, 2001	Apr 18, 2001	Apr 20, 2001	Apr 23, 2001		
Mozilla 0.9.2 (or Mozilla 1.0)	Apr 20, 2001	May 23, 2001	May 25, 2001	May 28, 2001		

1.3 Report Organisation

Based on the objective, as we demand to port existing Kirrkirr Java application unto Netscape/Mozilla browser, this will involve the decision about which Mozilla version to be used, due to some language ability restrictions as mentioned in the first chapter. Move further on, there is a section over viewing the XUL as mark up language for implementing graphic user interface. On the third chapter, we compared the Kirrkirr Mozilla application and its previous Java application in term of speed, user preference, code modularity, portability and readability.

The design explained in third chapter, are based on mark-up language design guidelines, about how to implement effective and efficient mark up language. Mark-up language guidelines are more for the benefits of developer in order to assemble the application efficiently. However, XUL are user to construct over graphical user interface, therefore GUI guidelines stay as important as mark-up language guidelines as revealed in section four. The focus on this chapter is exposing the method of obtaining GUI success that is to satisfy user needs.

After all, it is concluded that using XUL as online GUI builder is useful for certain extend. Some recommendation for further expansion and improvement are suggested on the last chapter.

Chapter 2 : Overview on XUL

[Robin Cover, 2000] *XUL stands for 'extensible user interface language'. It is an XML-based language for describing the contents of windows and dialogs. XUL has language constructs for all of the typical dialog controls, as well as for widgets like toolbars, trees, progress bars, and menus.*

At this point of time, the few browsers that support XUL file extension are Netscape 6 and Mozilla family. XUL is an XML based grammar for specifying the static GUI. The idea of developing XUL is to make UIs as easy to build as web pages, and as well as making application easier to write and to customize along the way.

[Mozilla.org, 2000] *The primary facility the developer will provide is that of instantiating windows, dialogs, and other hunks of user interface machinery from an XML description. The description will, hopefully, offer equivalent and broader power over the UI than currently supplied by HTML. Where HTML describes the contents of a single document, XUL describes the contents of an entire window. Unlike HTML, XUL is case sensitive as XML. The good points about XUL is that A XUL file can contain XML elements and HTML elements, as well as special elements unique to XUL: XUL elements.*

The declaration of XUL, so called, namespaces have to be correctly defined. Correct namespace usage dictates that the namespace be used only for the tag, not in individual attributes. Rare exceptions to this rule are bugs.

Allowing UIs to be constructed entirely from XML and JavaScript significantly lowers the bar for UI builders. Even, XUL grants freedom to programmer to set up their own skin, create their own button, menu or toolbar and defining certain rule to them. An application built on this service has the choice to expose it to end-users. This opens up many possibilities including, downloadable chrome, personal customization, etc. Making UIs as easy to construct, as web pages will open up UI evolution to the same massively parallel development that has so richly benefited open source code.

Writing XUL is nearly the same as generating XML file. XUL has a bunch of style sheet files declaring the behavior of its components. Having said that, the details of a particular

application of XML for example, the syntax for specifying a toolbar are left to separate documents describing those particular applications. A XUL interface is only a collection of disconnected widgets until it has been programmed. So far, with the help of JavaScript, that will tie up the widgets together and perhaps to give them extra functionality, or even to link them up with some method or function call that is build on C++ or even Java (using LiveConnect).

Other important matter in building complete UI using XUL is the RDF file. **[Mozilla.org, 2000]** *XUL templates are a way of embedding "live data" into a XUL document. A XUL template is a collection of rules that is used to build XUL and HTML content from one or more RDF data sources. A template specifies a "cookie cutter" content model pattern, along with the conditions indicate when the pattern should apply. Once a template has been specified, the browser handles the construction of the XUL (or HTML) content by copying the "cookie cutter" pattern and "filling it in" with appropriate values derived from the underlying RDF data sources. If the information in the data source changes, the XUL template engine keeps the generated content current.*

While DTD file defines variable with certain value that is mentioned everywhere in one file or multi files, this make the XUL looks neat and organized. This is done in order to make XUL files localizable, and can be done easily by substituting entities for any content, which may change as the locale changes. The whole system is then configured with different locale-specific DTDs, and the correct DTD will be chosen for a given XML file at runtime, depending on the current locale settings. To make XUL stream/file localizable, several guidelines are to be adopted to make the translation work easier and less error-prone.

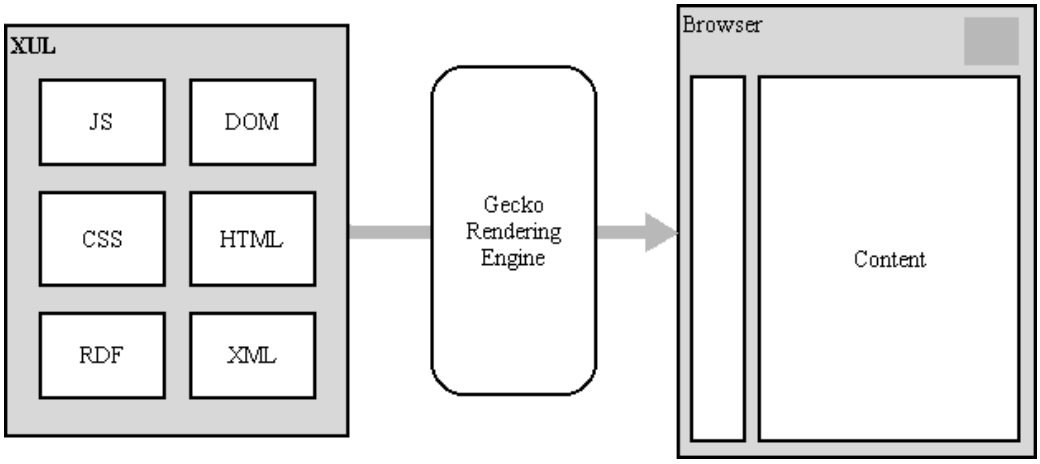


Figure 2.1 The operation of Mozilla rendering engine

Just as our specially created memo is nothing but text unless there is an application for making use of the structure in some way, so too our XUL widgets are just marked-up text unless there is software to interpret the structures and render the XUL as actual widgets. All XML requires a parser or an engine of some kind at the receiving end to make it run. This is, in short, what Gecko is. Gecko is a rendering engine that understands XUL and knows how to make it into pixels on the screen. Mozilla is a product made with all these technologies, but Gecko is the thing sitting inside that renders that product.

2.1 XUL Benefits

Web application (server-resident) and client application (locally-installable) developers have experienced the usefulness of XUL since :

- It is easy to build cross-platform, cross-device user interface quickly
- XUL adds common UI metaphors (boxes, springs) that web standards lack
- It can build small, powerful, quickly-downloadable application by leveraging power of browser instead of duplicating it

And as consumers :

- XUL applications are small, they download quickly, saving user time
- XUL applications run on any platform, they increase consumer's freedom of choice in selecting platform or device
- The ease of customising UI increases ability to tailor application appearance to own preferences

2.2 XUL Structure

As mentioned before, XUL applications consist of XML files created with .xul extensions. The files define the content of the application. Additional application data is located in Resource Description Framework (RDF) files. CSS files provide formatting, style, and some behavior, for the application. JavaScript files provide scripting support. Multimedia files, such as PNG images and other audio/visual files, might also be needed for additional user interface information.

[Edd Dumbill,2000]*All of the file types are specifications recommended by the W3C, and collectively are referred to as the XUL application's "chrome", the contents, behavior, and appearance of the application's user interface.*

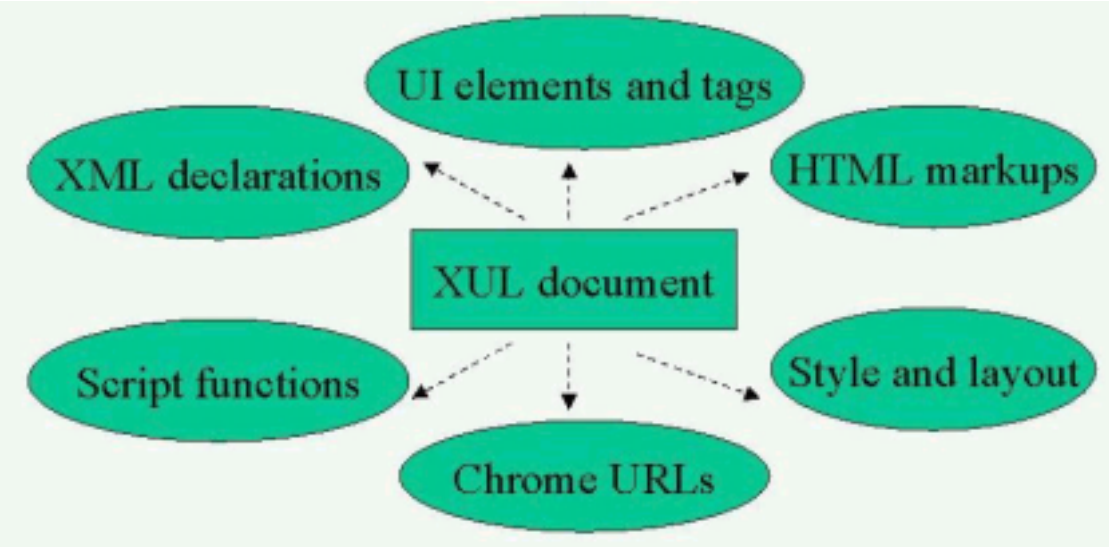


Figure 2.2 ***[Mozilla.org, 2000]*** *Components of an XUL document*

The following are part of XUL structure (note that the term structure is not the equivalent as components):

Chrome

[Dave Hyatt, 2000] *The Mozilla browser is itself designed as an XUL application. To manage the chrome for the browser, both Mozilla and Navigator have subdirectories labeled chrome, located off each browser's main directory. Within the chrome directory, separate XUL applications are packaged into separate subdirectories. Within each application directory, subdirectories further divide the application into content (containing the XUL, JS, and RDF files), skin (CSS files), and locale (DTD files).*

[Mozilla.org, 2000] *Then chrome is that part of the application window that lies outside of a window's content area. Toolbars, menu bars, progress bars, and window title bars are all examples of elements that are typically part of the chrome. There are four types of chrome providers: skin providers, content providers, platform providers, and localization providers.*

- *A skin provider* is responsible for providing a complete set of files that describe the visual appearance of the chrome. Typically a skin provider will provide CSS files and images.
- *A content provider* is responsible for providing a complete set of files for defining the structure of the chrome (e.g., the actual contents, such as menu items and toolbar buttons). The platform provider provides all of the files necessary to distinguish certain platforms.
- *The platform provider* typically provides both appearance and structure information.
- *The localization provider* is responsible for providing the actual string resources that entities in the XUL expand into.

The four providers work together to supply a complete set of chrome for a particular window, from the images on the toolbar buttons to the files that describe the text, contents and appearance of the window itself. The main source file for a window description comes from the content provider, and it can be any file type viewable from within Mozilla. It will typically be a XUL file, since XUL is designed for describing the contents of windows and dialogs. Since anyone can write a XUL file and supply images, anyone can be a chrome provider for a particular window.

Even within a particular window, the user may choose different providers for portion of the chrome. Since the files that come from the providers can live anywhere (e.g., on the user's local hard drive or on a remote site), the chrome itself can reside on multiple web sites and can be downloaded dynamically. Thus allowing providers to supply new services and commands as part of the chrome, and to update the user interface without having to make any change in the source code of the application. This feature is referred to as downloadable chrome.

Chrome Registry is another issue. The Mozilla client maintains a table known as the chrome registry that provides mappings from window types (a string identifier) to information about each of the four providers for the window types. For each provider type, a unique name is specified for the current provider, a base directory at which all of the files are located is specified, as well as the location of the main file to load for the provider type. In addition a URL to an archive (e.g., a JAR file) can be specified.

The archive is optional, but it is required in order to prevent remote chrome files from being downloaded one at a time (which could adversely impact the time from the window open to finished chrome display).

[Ian Oeschger, 2000] *Chrome URL is a concept of chrome as an integrated, dynamic thing in some way divorced from the "appcore" is realized in the use of the chrome url to point to chunks of XUL and their related files.* The chrome url, which appears in place of the http url in pointers like the following global skin processing instruction:

```
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
```

The code specifies chrome to be loaded by the Gecko rendering engine. The chrome is simply a skin file to be loaded into the XUL file, the chrome can also be used to load whole chromes, as when a menuitem in one window brings up a new chrome:

```
<menuitem
  value="Mozilla Help"
  oncommand="window.openDialog
('chrome://help/content/help.xul','_blank',
     'chrome,all,dialog=no')" />
```

In this example, the chrome url is being used to point to a chrome within the package hierarchy of the Mozilla application. A “Help” chrome defined in “mozilla/bin/chrome/help/” is being invoked from the Help menu.

Note that when no file name is specified after the chrome directory path, a file name with the same name as the package is assumed. In other words, a chrome url like the global pointer above picks up a file called global.css, and the help pointer above could also be written as “chrome://help/content”, because the name of the package itself is “help”.

When a chrome URL is specified, the client does the following:

1. The window type entry is found in the chrome registry.
2. If an archive is specified, then the client looks in the archive first. If the file is found there it is used.
3. If the file is not found in the JAR file, then the client looks in the appropriate place in the chrome cache for the file. If the file is found in the cache, it is used.
4. If the file is not found in the cache, the client will attempt to fetch it from its original location. If the file is found there, it is used. If not, then the client will give up the ghost.

To understand the hierarchy even more, refer to the example below : Directory Structure Outline

```
[Mozilla]
  [chrome]
    [navigator]
         [skin]
           [default]
             navigatorSkin.jar
             ... other files bundled with navigator ...
           ... other skins used by users for navigator ...
         [content]
           [default]
             navigatorContent.jar
             ... other files bundled with navigator ...
          [platform]
            [default]
             ... optional JAR file and other files ...
         [locale]
           [default]
             ... optional JAR file and other files ...
    ... subdirectories for other specific window types ...
```

Package

[Mozilla.org, 2000] *A package is in some ways like chrome, but it is specific to the Mozilla architecture. A package is a chunk of interface code that sits in a particular place within Mozilla's package hierarchy. Like chrome, that chunk usually contains XUL content, CSS and graphic skin information, localization strings, and maybe some platform-specific code.*

The default directory underneath each of these main package subdirectories is assumed in the chrome url (i.e., chrome://help/content/help.xul does not include a default directory as part of the url, though that directory exists in the actual structure). When different chrome created for a package, a subdirectory can be created underneath content whose contents are loaded instead of default. For example, if we want to create a different skin for the navigator package, we can create a subdirectory underneath navigator/skin/ whose contents will be loaded instead of default skin.

Widgets

The different sections of a XUL application are contained in widgets known as “boxes”. Boxes don't have any visual appearance themselves; their only purpose is to encapsulate several widgets as a whole, and to provide layout orientation for their contents. The orientation is defined with the `orient` attribute, and other attributes can be used to set the box width and height.

Example of widgets :

- Menu Bars and Menus
- Toolbars and Toolboxes
- Titled Buttons
- Tree Widget
- Tab Widget
- Sliders and Scrollbars
- The Splitter Widget
- Progress Meter
- Checkbox (Tri-state)

Each of these will be used and discussed on the Design Issue Portion

Namespaces

[Edd Dumbill, 2000] *Definition : An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute name. URI references, which identify namespaces, are considered identical when they are exactly the same character-for-character. Note that URI references, which are not identical in this sense, may in fact be functionally equivalent. Examples include URI references, which differ only in case, or which are in external entities which have different effective base URIs.*

From the definition, concluded that XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.

Scripting

A XUL interface is only a collection of disconnected widgets until it has been programmed. "Programming" can be as simple as some JavaScript to tie the widgets together and perhaps to give them extra functionality, or as complex as application (C++) code which is free to do anything.

JavaScript: XUL can contain HTML content, including JavaScript. JavaScript functions may be added in a fashion similar to HTML. There is no `<head>` section of a XUL file, so script is just mixed in with the other content, delimited by a `<script>` tag in the HTML namespace. JavaScript can be referenced as in HTML documents: as `onClick` handlers and the like. See individual widget documentation referenced at the index for a list of attributes accepting JavaScript values.

[Mozilla.org, 2000] *JavaScript is most safely kept in a separate file and included in the XUL file or relegated to the contents of a CDATA section, to prevent the XML parser from choking on JavaScript which may look like XML content (a '<' character, for instance.)*

Style and Layout

Style and Layout is most often brought up in the context of dynamic changes to the overall look of an application. Though this is not yet in the browser, very soon it will be possible to change the look of a whole application dynamically, but only to the extent that the skin is actually defined in the main `global.css,` or global skin. When programmers create styles in <style> tags, as style attributes for individual elements, or in custom CSS files, we break the ability of Gecko to skin the application to which our XUL belongs.

When a URL of the form `chrome://WindowType/skin/` is encountered, the default CSS file for that WindowType is loaded from the skin provider. Any files from the skin provider can be referenced using chrome URLS, with the assumption that `chrome://WindowType/skin/` refers to the base directory of the skin provider as specified in the chrome registry.

A theme is an RDF file that specifies its own chrome registry. When a theme is installed, all of the entries contained in the theme's chrome registry are copied into the user's chrome registry. This allows chrome writers to specify a theme for several different window types and

provider types at once and have the theme be installed in one fell swoop onto the user's machine.

Chapter 3 : Interface Design

3.1 Mark up Language Style Guidelines

[Tao Cheng, 2000] *The procedure to generate XUL effectively are listed as follow:* (all the example here are within the Kirrkirr project on Mozilla)

1. *Convert HTML files to XUL. All UI descriptions written in HTML shall be converted into XUL. Currently, programmers do not have a clean way of localizing HTML files. XUL is our UI description language. In addition, there are certain XUL specific features, such as XUL fragments, which are supported in XUL content model only.*

   This is another alternative way instead of hard coding, for we hardly can find XUL editor (some exist on the net, but they are buggy). The code below demonstrates how front page or dream weaver can be employed to generate a table and inserting texts. The plain tags are HTML and the blue tags are XUL tags.

```
<html:div align="left">
   <html:table border="0">
          <html:tr>
                 <html:td width="60" height="200">
                        <titledbox orient="vertical">
                               <title>
                                      <text value = "Help Option" />
                               </title>
                        <text value= "To overview more "/>
                        <text value= "explanation in details "/>
                        <text value= "click button 'More'"/>
                        <text value= "on the side of "/>
                        <text value= "tab body "/>
                        <text value= " "/>
                        <text value= "if you still "/>
                        <text value= "encounter problems"/>
                        <text value= "please feel free"/>
                        <text value= "to contact us"/>
                        <text value= " "/>
                        <text value="d578157@ntu.edu.sg"/>
                        </titledbox>
                 </html:td>
```

```
                <html:td width="600" height="200">
                        <titledbox width="600" height="200"
                        orient="vertical">
                                <title>
                                        <text value = "Description"/>
                                </title>
                                <iframe id="help-window" width="600"
                                height="190" src="html/help_help.html"/>
                                <button id="more" value="More"
                                class="dialog"
                                onclick="window.open('html/help.html');"/>
                        </titledbox>
                </html:td>
        </html:tr>
    </html:table>
</html:div>
```

But there are some XUL specific features, in which we don't have any other choice but to hard code and try to view it on the browser. Codes without "html" tags above (in blue) are those, which is XUL specific code. Figure 3.1 reveal how the code being viewed in the browser.

Figure 3.1 General.xul-this page consisting HTML altogether with XUL code.

2. *The XML declaration. All XML declaration shall contain the following information:*

- Version number.

  Version number indicates that this XML document conforms to a particular version of XML specification. It is provided, as a means to allow the possibility of automatic version recognition, should it become necessary.

Processors may signal an error if they receive documents labeled with versions they do not support [Prolog and Document Type Declaration].

- UTF-8 encoding.

  Wherever applicable, the recommended encoding of the content data and attribute values is UTF-8. In Mozilla, when such information is not specified, UTF-8 is the default encoding of all XML, XUL, and RDF documents. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.

- Style sheet.

  An XML file must have a style sheet associated with it, otherwise, it won't know how to display the document. Note that, as suggested below in "Configurable Chrome", chrome type URLs shall be used to reference the style sheet.

```
<?xml version="1.0"?>
<!--
This line simply declares that this is an XML file.
Currently only one version exists (1.0) but if or when a new version is published, the browser can determine what version to expect. It would normally being added at the top of each xul file, much like one would put the HTML tag at the top of an HTML file.
-- >

 <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
 <!--
 This line is used to specify the style sheets to use for the file.
 This is the syntax that XML files use to import style sheets. In the case, we import the styles found in the global/skin chrome directory. We didn't specify a specific file so Mozilla will determine which files in the directory to use. In the case, the all-important global.css file is selected (by default).
 This file contains all the default declarations for all of the XUL elements.
 -- >

 <window
     id="Kirrkirr-window"
     title="Help Files"

      xmlns="http://www.Mozilla.org/keymaster/gatekeeper/there.is.only.xul
      ">
      ...
 </window>
 <!--
```

This line declares on describing a window. Each user interface window is described in a separate file. This tag is much like the BODY tag in HTML, which surrounds the entire file. Several attributes can be placed in the window tag.
-- >

3. In XUL documents, namespace is set by default to “xul” and declared in the root element. Other namespace referenced shall be declared before use and preferably in the root element of the document.

```
xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.Mozilla.org/keymaster/gatekeeper/there.is.only.xul"
.. .
```

<!--
This is an XML feature that indicates that the file is a XUL file.
The xmlns attribute, which stands for XML namespace, indicates how Mozilla should interpret the tags. The URL specified would normally point to a definition of XUL. Note that this URL is never actually downloaded. Mozilla will recognise this URL internally.
-- >

4. Case sensitivity. Be aware that XML is case sensitive. In compliance with XHTML working draft, all element names and attribute names shall be in lower case.

5. Configurable Chrome. Three XUL style guidelines are proposed :
   - All resource reference in XUL shall use the chrome type URLs instead of hardwired URLs.
   - Place all language neutral files in packages/ and skins/ directories.
   - Place all locale sensitive files under their specific sub-directories such as “chrome/locales/en-US/navigator/locale/*”.
   - Note that all languages are treated equally and stored in their respective sub-directories. Candidates of such files are language-specific DTD files. Why is it important to put language-specific files in locale-specific sub-directories? Because user may want to be able to switch between languages on the same user machine. This is essential on UNIX and maybe on other Operating systems.

6. JavaScript code. XUL does not make use of “script” tag in order to start and end java scripting, instead of mixing JavaScript and HTML it separated the JavaScript file and later on include it on the namespaces of XUL files. The advantages of this are :

   - Legibility.

     The main purpose of the XUL is to describe the UI content.

     As style information shall be in style sheet, JavaScript code shall be in *.js as well to keep the main XUL file clean. In some XUL files, JavaScript code takes up 20-25% of the length of the file. Often times, programmers need to go to the very bottom of the file to find the UI descriptions.

   - Maintainability.

     By putting JavaScript code in external files, what can be done is to isolate the function declaration in a separate file. The developer does not need to touch the main XUL to change its runtime behavior.

   - Flexibility.

     Since the JavaScript code is in an external file, customization can be achieved by switching the JS file. With chrome registry, developers, content providers, or even the end users will be able to edit the registry table to use a customized JS code for their browser client.

     For example :

```
<script language="Javascript"
src="chrome://global/content/globalOverlay.js">
</script>

<script language="JavaScript" src="chrome://global/content/strres.js">
</script>

<script language="JavaScript"
src="chrome://communicator/content/bookmarks/bm-props.js">
</script>
```

7. DTD declaration.

   As the following referring to bm-props.dtd :

```
<!DOCTYPE window SYSTEM
"chrome://communicator/locale/bookmarks/bm-props.dtd" >
```

8. Predefined Entities.

9. Other guidelines, for example on how to write HTML content in XML documents that are portable across conforming parsers.

3.2 Overview of Kirrkirr

Kirrkirr is described as, Warlpiri dictionary (a central Australian language), a web-based application for interactive exploration of dictionaries. The main feature of this dictionary is because it has converted the existing Warlpiri dictionary into a richly structured XML version.

[Kevin Janz, 1998] *Instead of using simple relation structured data, the dictionary has been using XML, which has rich hierarchical structured which then allowed the precise definition of dictionary contents.*

[Kevin Janz, 1999] *We describe Kirrkirr, a web-based application for interactive exploration of dictionaries. It currently targets Warlpiri (a Central Australian language). A key feature of our work is that we have converted the existing Warlpiri dictionary into a richly structured XML version. The flexibility and hierarchical structure of XML is ideally suited for supporting rich but loosely structured content such as dictionaries, while web-based distribution is particularly attractive because dictionary maintenance can be done on a central server, and Java-based clients can access up-to-date dictionary information as needed. Kirrkirr provides a graph-based display of semantic links between words, which provides an engaging interface that can be explored, manipulated and customised interactively by the user (for example, a language learner).*

[C. Manning, 1999] *The goal of this work is somewhat different from that of most other projects at this workshop, they are:*

- *To provide software that was usable by people other than tertiary-educated linguists. Within the Australian context, indigenous dictionary structure and usability has usually been dictated by professional linguists, while the needs of others (speakers, semi-*

speakers, young users, and second language learners) are not met. But since Kirrkirr delivered with multimedia ability, it can help the learner to pronounce the word by recording the pronunciation and accent of a native speaker.

- *Linguists have been collecting rich lexical materials since the 1950s, resulting in one of the most comprehensive lexical databases for any Australian Language (Laughren and Nash 1983). Using XML this rich hierarchical structured can be easily constructed and displayed on the web.*

- *There is a relatively large community of people, who speak Warlpiri as their first language, and some of whom have had the opportunity of bilingual schooling in Warlpiri and English, who would be able to benefit from the existence of a suitable dictionary*

- *To make better use of computers for visualization, hypertext linking and multimedia in order to provide a richer experience of dictionary content. The main idea is to provide a fun dictionary tool that is effective for browsing and incidental language learning, as well as for serious research, in part because indications are that current interfaces are unlikely to have much direct educational benefit for students. From this viewpoint, the low level of literacy in the region, and the inherently captivating nature of computers suggests that an e-dictionary is potentially more useful than a paper edition. The dictionary come with the idea to emphasize the ability of electronic, network / web based application that paper dictionary couldn't have.*

- *To promote standards-based computing within descriptive linguistics.*
  A leading reason for the lack of adequate tools for field linguistics has been the reliance on homespun software tools, which do not effectively inter-operate with mainstream software. This has greatly restricted the ability of people to get functionality for free.

The following are Kirrkirr main modules :

- Graph Layout
  Describing words and their various relationships with other words are drawn in the form of an animated network graph of nodes with color-coded links between them. Every color displayed will representing different relation between words. The nodes "float" on the screen and the user is encouraged to move the words around. This is supported by

complicated algorithm and thread outfitted in Java. The network can be widen up by the user clicking, or even can be shrunken down.

- Formatted Entries

  Representing the graphical layout in textual form. The user is also able to read the large amount of information contained in the dictionary entries. Unlike the compact formatted entries of a paper dictionary, the entries are nicely laid out with sensible use of color where appropriate. There is also functionality for the user to click on a cross-referenced word and have the system leap to the entry for that word (hyperlink ability).

- Notes

  This panel is for user to add something for his/her personal use. It is important that there is a facility for something like “pencil notes in the margin”. The users can very easily jot down (i.e., type) notes for a specific word as they use the system. These notes are saved in a user profile that can be searched later on There is even the option to move these notes around in separate windows, like post-it notes.

- Multimedia

  A rare feature for an e-dictionary is the ability to hear the words of the dictionary to understand their pronunciation. This feature coupled with various pictures relating to the word being looked at makes the system very user friendly.

- Advanced Search

  Included with the many ‘fun’ aspects of the system is the ability for serious searching of the database. The users can perform, not only searches according to the nearby words in alphabetical order, but also searches using regular expressions, approximate ‘sounds like’ spelling or just plain text. This ability us facilitated by the well marked-up XML dictionary that is by brute force method, any field in the dictionary can be searched for.

- Explorer

  Is presenting the tree / hierarchical version of selected word. This feature will help user to understand the relation among words based on semantic domain.

- Tutor

  Tutoring tools will help the users to load certain module and be able to learn the semantics part of the language by try on to answer the given question on the tutor unit module.

In order to assist the analysis part of the project, this java version of Kirrkirr has being arranged into several folders and compressed into one file.

Table 3.1 Kirrkirr Java Application Structure

Folder Name	Size	Details
HTML	1.81 MB	Consist of 62 files and 6 subdirectories. This HTML folder keeps all files with extension .html, some jpg files associated with the HTML files as well as those files needed for Java Help set (ctHelp folder).
Images	2.65 MB	Includes all the multimedia display part (up to 154 jpg and gif files).
Tutor	7.52 kB	Consist all the tutoring input files
Users	0 kB	Folders where users profile will be kept
XSL	14.2 kB	Includes dictionary-input files on XSL form. Currently there are 3 files inside.
Some of library files, not include in one of the folder above	981 kB	9 Files
Demo.jar	154 kB	Keeps all Java class files

Total of 5.59 Mega Bytes

3.3 Enhancing Kirrkirr Functionality via Browser

As the objective of this project is to rebuild Kirrkirr application to make it available via browser, particularly Mozilla browser, yet the main thing is how to make use of all the browser distinct ability. For Mozilla, its special ability is to view document with mime type text/xul. The following will explain the plan design of Kirrkirr in Mozilla.

Kirrkirr is Java based application, and in order to run it on online browser, java enable plug-in is essentially needed**.[R-cube system ltd, 1995]** *The latest industry standard Web browsers are Java-compatible, so there is widespread support for Java, in terms of both the number of*

sites where it is available to Web users, and the number of hardware platforms on which it is supported.

Seamonkey, or Mozilla milestone 18, launched on October 12, 2000 is not latest Mozilla project (as it is shown in introduction part). In fact there is another latest version which is Mozilla 0.8. Milestone 18, Mozilla 0.7 and 0.8 are fully equipped with JRE (java plug-in) within comprehensive installation option as well as Live Connect enabled.

In this project, all of Kirrkirr XUL files has mixed platform, which is taken from Mozilla chrome. With this configuration, this make it is possible for the theme to change then followed by the conversion of overall appearance. The code below is a portion of Kirrkirr-main.xul, main file of Kirrkirr, which implies the usage of 6 navigator local (chrome) files.

```
<?xml-stylesheet href="chrome://navigator/skin/navigator.css"
type="text/css"?>
<?xml-stylesheet href="chrome://communicator/skin/sidebar/sidebar.css"
type="text/css"?>
<?xml-stylesheet href="chrome://messenger/skin/messenger.css"
type="text/css"?>
<?xml-stylesheet
href="chrome://messenger/skin/messengercompose/messengercompose.css"
type="text/css"?>
<?xml-stylesheet href="chrome://communicator/skin/communicator.css"
type="text/css"?>
<?xml-stylesheet href="chrome://communicator/skin/bookmarks/bookmarks.css"
type="text/css"?>
```

As users download new skin on the browser and apply it, another six files with same name but different content will replace previous files. For example, the following figures (Figures 3.2) make use of navigator.css, sidebar.css, messenger.css, and etc from original/default installed skin on Mozilla (modern.jar).

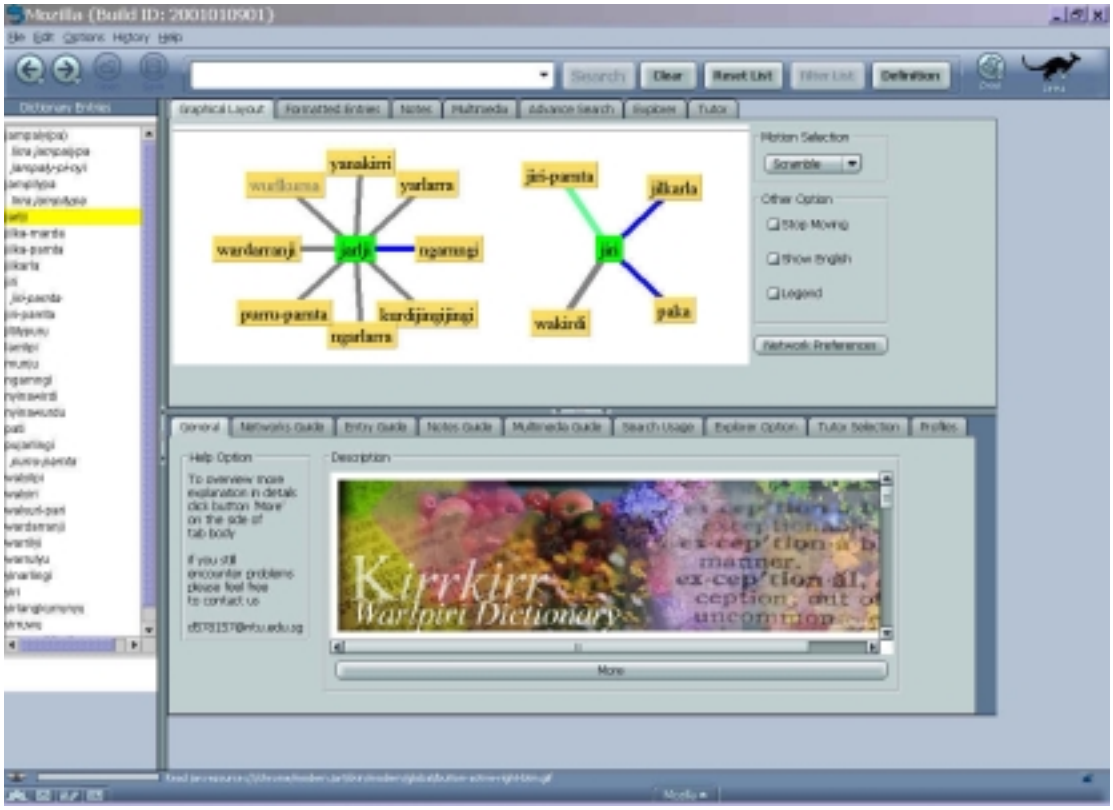


Figure 3.2 Kirrkirr on Modern Skin

Then if users download Gray Modern Skin (in the form of jar file : graymodern.jar) and decided to use it, then our browser will pointing to navigator.css (and the rest of css files) reside inside graymodern.jar file instead of navgator.css inside modern.jar and causing the change of the overall appearance of the browser skin.

Next, is the planning of applet placement. **[Java Look and Feel Guidelines, 1999]** *Applet in current user's browser window well suited for displaying applets in which users perform a single task. This approach enables users to perform the task and then resume other activities in the browser, such as web surfing.* An applet displayed in the current browser window should not include a menu bar, having a menu bar in both the applet and the browser might confuse users. The mnemonics assigned in the applet must also be diverse from the mnemonics used to control the browser window; otherwise, the mnemonics might conflict.

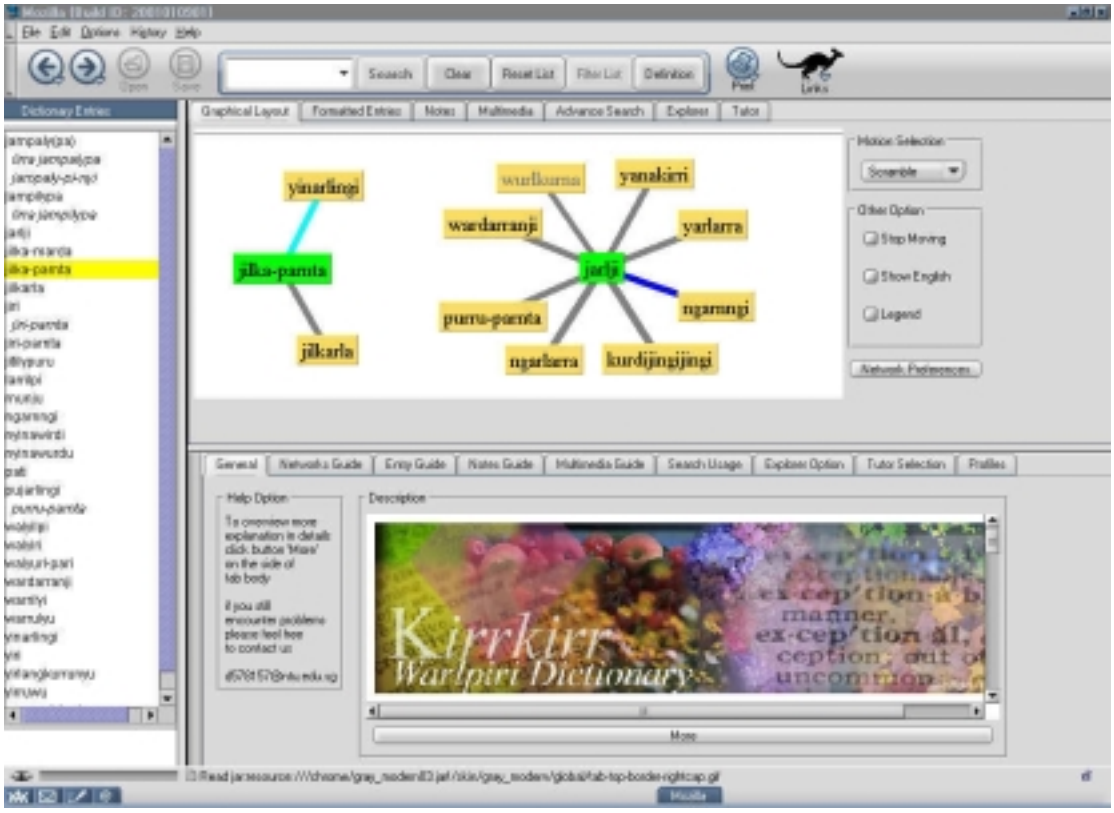


Figure 3.3 Kirrkirr on Millennium Modern Skin

A disadvantage of using the current browser window is that the applet terminates when users navigate to another web page. The current settings and data in the applet are lost. To use the applet again, users must navigate back to the page that contains the applet and reload the page.

On the other hand, if the applet involves more than one task or if users might visit other web pages before completing the task, launch a separate browser window and display the applet there. This approach enables users to interact with the applet and persevere the original browser window for other activities. Navigating to another web page in the original browser window does not affect the applet in the separate browser window.

Figure 3.4 Index.html is the introduction part to Kirrkirr application

Designing an applet for a separate browser window is simpler if you remove the browser's normal menu and navigation controls. Doing so to anticipate confusion between the browser's menu and controls and the applet's menus and controls. You also avoid potential conflicts between mnemonics in the two windows. Due to this consideration, it is good that Kirrkirr designed to open a new browser since it has it own menu panel and option on its top. Through Java script, it is feasible to disable the sidebar and menu bar default option. From the browser point of view, it is best to make use of the skin (.css) of current display so that it is unnecessary to author new .css file. For the final version of Netscape, the XUL file added should be zipped (.jar) to one of recognisable existing jar file, as well as any other new file(picture, rfd,dtd).

As a part of design, we consider about the installation issue and a problem when users try to access Kirrkirr with non Mozilla browser. On index.html (Figure 3.4) page, user may get into: the link of application system requirements and installation issue and another separated link is to Kirrkirr.xul (Figure 3.5) files, which is Mozilla specific viewed. Note that browser menu and editor are still available on this stage.

On the bottom of Kirrkirr.xul page, there are a button which link up to main Kirrkirr application window as well as hiding menu and another browser's tools which consider may be perplexing the users. The link from Kirrkirr.xul page is formed as XUL button, so if user try to open this page by using another browser which is not XUL enabled, they simply can not see the button/the link to the primary application. Then another concern is how to place and arrange the current Kirrkirr Java application on the browser. Kirrkirr Java application presently, has its Java applet version (as a single applet) which look precisely as figure 1.1 above. As we attempt to replace the entire Java tab panel with XUL tab panel, meaning that the recent Java applet needs to be broken up to occupy each panel.

Figure 3.5 Kirrkirr.xul, to make sure that users are currently using Mozilla before get to the main application

Hence, the application consists of 7 main panel with each applet on each tab panel (Graphical Layout, Formatted Entries, Notes, Multimedia, Semantic Search, Explorer and Tutor Panel), and the left side is the headword applet (as independent applet). The bottom pane is enlarged

to Help Options. The application has its own Menu and Browser option therefore it is open on another independent window.

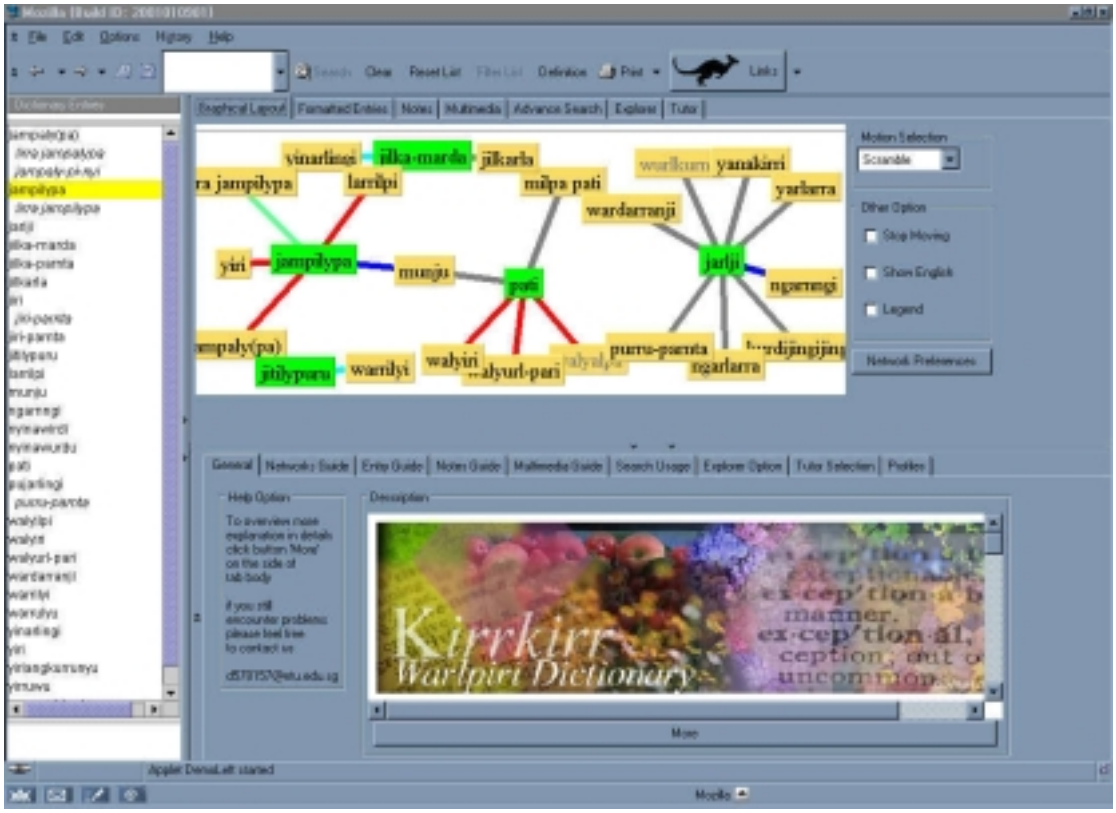


Figure 3.6 Kirrkirr Main Windows viewed on Native Window Skin, has its own menu and browser option

Compared to the previous Kirrkirr Java application, the entire menu (including popup menu and drop down menu), button and check box are achieved using XUL. The idea of implementing those widgets on XUL instead of leaving those as they are (in java), is because:

- The code to create one widget in XUL is more concise than in Java. For example this line of code is to add one button in java compare with to add one button in XUL. Since the code is more compact, the execution duration will be faster.

  Creating “scramble” button :

```
private JButton scramble;

scramble = new KirrkirrButton("Scramble", this);
scramble.setBackground(Color.red);
```

```
scramble.setForeground(Color.white);
p.add(scramble);
```

And Kirrkirr button class has method as the following to add the action listener:

```
KirrkirrButton(String text, ActionListener al)
 {
       super(Helper.getTranslation(text));
       setMargin(makeSmall ? smallInsets: regularInsets);
       addActionListener(al);
 }
```

To attached the button with certain action :

```
if (e.getSource() == scramble)
{
       funPanel.scrambleShake(true);
}
```

While in XUL, the portion of program will be :

```
<button id=”Scramble” value=”SCRAMBLE” class=”dialog”
onClick= "document.getElementById('GraphDemo').scrambleButton();"/>
```

On the example above, the applet name “GraphDemo” has method scrambleButton(); which called function funPanel.scrambleShake(true), in fact we can make use of the preceding Java class method. The following is the full section of XUL code for displaying the graph panel.

```
<box align="vertical" id="First">
<!--
This part is to include the GraphPanelDemo class file into the window, the GraphPanelDemo
class catch the size of the applet and another parameter needed. This is for the programming
maintainability and flexibility as another programmer can simple change the size the applet,
change the folder name or the parser name if they wish to, without touching on the java code
and do compilation.
-->
       <box align="horizontal">

                     <html:applet name = "GraphDemo"
                            id = "GraphDemo"
                            code ="GraphPanelDemo.class"
                            archive = "demo.jar, gnu.jar, oro.jar,
                            jh.jar, parser.jar, xt_old.jar"
```

```
                        width = "600" height = "250">

                <html:param name="xml_file" value="Wrl.xml"/>
                <html:param name="index_file" value="Wrl.clk"/>
                <html:param name="html_folder" value="html"/>
                <html:param name="applet_width" value="600"/>
                <html:param name="applet_height" value="250"/>
                <html:param name="applet_num" value="0"/>
                </html:applet>

<box align="vertical">
<!--
```
This is the interface part, consists of dropdown menu, button and check box. Each of GUI will call java method, which declared in Graph Demo. The class, which originally provides the method, is (in this case) GraphPanel.java. Then we forward the method to be on the GraphPanelDemo.java so that it is accessible by the JavaScript.
```
-->
        <titledbox>
                <title>
                <text value="Motion Selection"/>
                </title>

                <menulist id="match">
                        <menupopup id="matchPopup">
                                <menuitem data="scramble"
                                value="Scramble"
                                oncommand="document.getElementById('
                                GraphDemo').scrambleButton();"/>

                                <menuitem data="shake"
                                value="Shake"
                                oncommand="document.getElementById('
                                GraphDemo').shakeButton();"/>

                                <menuitem data="clear"
                                value="Clear"
                                oncommand="document.getElementById('
                                GraphDemo').clearButton();"/>

                                <menuitem data="random"
                                 value="Random Pick"
                                oncommand="document.getElementById('
                                GraphDemo').randomButton();"/>
                        </menupopup>
```

```
                    </menulist>
                </titledbox>

                <titledbox orient="vertical">
                    <title>
                    <text value="Other Option"/>
                    </title>

                    <checkbox id="StopMoving" value="Stop Moving"
                    oncommand="document.getElementById('GraphDemo').st
                    opCB();"/>
                    <text id="SM"/>

                    <checkbox id="ShowEnglish" value="Show English"
                    oncommand="document.getElementById('GraphDemo').en
                    glishCB();"/>
                    <text id="SE"/>

                    <checkbox id="Legend" value="Legend"
                    oncommand="document.getElementById('GraphDemo').le
                    gendCB();"/>
                    <text id="L"/>
                </titledbox>

                <button id="network" class="dialog"
                value="Network Preferences"
                onclick="document.getElementById('GraphDemo').graphOptio
                nPanel();"/>
            </box>
    </box>
</box>
```

Additionally, with XUL, the programmers need not to be troubled about the appearance and the alignment of the button. Yet those attributes are declared and defined in CSS files. So, to transform it, minimally open, revise and correct corresponding CSS file (changing the colour, shape, etc) or even change the alignment.

Now the application has cuts lots of Java interface and reduces a lot of Java code. But in fact, although we had preserved a lot of space by replacing more Java code by XUL, the controlling part become very convoluted as those 8 applet need to communicate and synchronise with each other.

With the same number of class file (45-source file), previously the source code consumes 467 kilobytes and now it uses not more than 444 kilobytes. But since we need to forge another 7 new Java class to display and control new applets, yet the size become bigger than when it is resolved on single applet. As the project still works in progress, we still strive to do code minimisation. Currently, the old jar file is 154 kilobytes and the new jar file (including the new applets class file) not more than 181 kilobytes.

There is certain amount of delay of loading each jar files needed by each applet. This is slower than if there are only one distinct applet (only once loaded on the initiation of browser window) whereas the loading will be done every time an applet is loaded, initialised and started and hence, there are 8 applet now. Therefore the loading time will lots slower (the loading, initialising and starting of applets can be monitored by select the fifth option on Mozilla Java console window).

It is also not responsive to first time action performed on the interface, since JavaScript need to call Java method on inner Java class file via Liveconnect. The delay is determine more on the kind of task need to be completed rather than the depth of the forwarding method (because, as explained before, that we forward the original Java method to the Java Applet class file). If the task is complicated, although the original method is on the applet class itself, the user may experience more delay than simple task which method originally placed on inner class.

- In Java, it is obstinate to change the presentation of the GUI, since the programmers essentially look through the code and decided what to alter and enhance. While in XUL, programmer just need to play with related CSS files or even just the XUL file itself. Hence the code is high level in the term of usability.

  The idea is to use various source of .css file, which declared same element with different attributes and appearance. For example both package modern.jar and classic.jar in Mozilla, contain navigator.css files. Both navigator.css files have class name “navigator-toolbar” embedded inside. But both have dissimilar looks and style. Let say for modern package, the toolbar defined in this class will have blue background while it will be grey in classic package.

Another examples are these search-button :

(The following example showing the concepts of “theme”. Where the browser will decide on to load different jar file containing CSS file one at a time. Hence, the jar file accommodates similar CSS files with distinct exterior and characteristics, the search button on classic mode will guise and behave differently in contrast with search button on modern theme)

Search button in navigator.css from Modern.jar

```
#search-button
    {
    margin-right          : 4px;
    }
```

Search button in navigator.css from Classic.jar, this button has different image mapped when the mouse over action occurred.

```
#search-button
    {
    margin                : 1px 3px 1px 2px !important;
    list-style-image      : url("chrome://communicator/skin/search.gif");
    font-weight           : bold;
    }

#search-button:hover
    {
    list-style-image      : url("chrome://communicator/skin/search-
    hover.gif");
    margin                : 0px 2px 0px 1px !important;
    }

#search-button:hover:active
    {
    list-style-image      : url("chrome://communicator/skin/search-
    active.gif");
    margin                : 0px 2px 0px 1px !important;
    }
```

- As mentioned before : As HTML tag may be included in XUL, make it easy for programmers to mixed between XUL and HTML. Certain portion such as long text, is easier to be assembled in HTML since it has wide spread editor rather than hard coding, as occurs in XUL. Therefore, certain segment of this application, for instance “help”, has

made use of frame in the reference with a bunch of HTML files while the rest of the widgets are affirmed in XUL.

For example entrenching applet inside XUL files by adding “html:” phrase in front of every html tag to bring about it recognizable by the XUL file as XUL tag.

```
<html:applet name = "GraphDemo"
code ="GraphPanelDemo.class"
archive = "demo.jar, gnu.jar, oro.jar, jh.jar, parser.jar, xt_old.jar"
width = "600" height = "250">

<html:param name="xml_file" value="Wrl.xml"/>
<html:param name="index_file" value="Wrl.clk"/>
<html:param name="html_folder" value="html"/>
<html:param name="applet_width" value="600"/>
<html:param name="applet_height" value="250"/>
<html:param name="applet_num" value="0"/>
</html:applet>
```

Another case is to make a table and XUL frame. First of all, just do the table on HTML editor, then add “html:” tag in front of those HTML tags in order the parser to be able to interpreted them. The frame concept is the same as on HTML file. It acquires the source file from the location specified (can be on .html extension, xul, or even opening jpg files)

```
<html:div align="left">

<html:table border="1" width="584" height="119">
<html:tr>
   <html:td width="246" height="119"></html:td>
   <iframe flex="1" id="contents" src="content.xul"/>
   <html:td width="326" height="119"></html:td>
</html:tr>
</html:table>

</html:div>
```

- It is easier to align components in XUL than in Java, since the HTML tag can be worn as well (center, left and right), as well as “orient” key word to bring into line the components vertically or horizontally. To see the effects of altering the code, the programmers just need to view the files on the browser without compiling them.

- Some of intricate widgets cannot be implemented in java easily or even on another browser. With Mozilla, the basic behaviour and appearance is defined internally, for example the following tab panel.

In java :

```
static JTabbedPane[] tabbedPanes = { new JTabbedPane(),
new JTabbedPane(/* no! JTabbedPane.BOTTOM */) };
```

//Creating 2 bunch of panel, where as TOPPANE contains similar item as BOTPANE.

```
CTTabs[TOPPANE][GRAPH] = new GraphPanel(window, this);
CTTabs[TOPPANE][GRAPH].setName("Network");
CTTabs[BOTPANE][GRAPH] = new GraphPanel(window, this);
CTTabs[BOTPANE][GRAPH].setName("Network");
..
..
CTTabs[TOPPANE][TUTOR] = new TutorPanel(this);
CTTabs[TOPPANE][TUTOR].setName("Tutor");
CTTabs[BOTPANE][TUTOR] = new TutorPanel(this);
CTTabs[BOTPANE][TUTOR].setName("Tutor");

for (int j = TOPPANE; j <= BOTPANE; j++) {
    for (int k = 0; k < CTPANES; k++) {
           tabbedPanes[j].insertTab(CTTabs[j][k].getName(), null,
           CTTabs[j][k], paneRollover[k], k);
           }
        tabbedPanes[j].setBorder(null);
        tabbedPanes[j].addChangeListener(this);
        tabbedPanes[j].setPreferredSize(new Dimension(400, 200));
    }
    tabbedPanes[BOTPANE].setSelectedIndex(HTML);
    splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                 tabbedPanes[TOPPANE], tabbedPanes[BOTPANE]);
..
..
```

In XUL:

```
    <tabcontrol id="Kirrkirr" orient="vertical">
          <!--heading of the panel-->
          <tabbox>
```

```
                <tab value="Graphical Layout"/>
                <tab value="Formatted Entries"/>
                <tab value="Notes"/>
                <tab value="Multimedia"/>
                <tab value="Advance Search"/>
                <tab value="Explorer"/>
                <tab value="Tutor"/>
        </tabbox>
        <!—contain of each corresponds panel-->
        <tabpanel>
                ..
                ..
        <box>
        </box>
                ..
        </tabpanel>
</tabcontrol>
```

Other this is that Java Help Modules (version 1.1) has been removed and replaced by full XUL and HTML files, since creating Java Help will force the need of generating another applet and hence bring about more delay. The new Help Module make use of Mozilla tree structure and framing ability.

```
<tree id="treeset" flex="1">
        <treecols><!--alignment purpose-- >
                <treecol flex="2"/>
                <treecol flex="1"/>
        </treecols>

        <treechildren>

                <treeitem container="true">
                        <treerow>
                                <treecell class="treecell-indent"
                                 value="Kirrkirr Help"
                                onclick="document.getElementById('advance')
                                .setAttribute('src','help.html#top');
                                document.getElementById('picture').setAttri
                                bute('src','../picture/help.jpg');"/>
                        </treerow>

                        <treechildren>
```

```
<treeitem>
	<treerow>
	<treecell class="treecell-indent"
	value="Main Menu"
	onclick="document.getElementById('ad
	vance').setAttribute('src','help.htm
	l#toolbar');
	document.getElementById('picture').s
	etAttribute('src','../picture/menu.j
	pg');"/>
	</treerow>
</treeitem>

<treeitem>
	<treerow>
	<treecell class="treecell-indent"
	value="History"
	onclick="document.getElementById('ad
	vance').setAttribute('src','help.htm
	l#toolbar');
	document.getElementById('picture').s
	etAttribute('src','../picture/menu.j
	pg');"/>
	</treerow>
</treeitem>

<treeitem>
	<treerow>
	<treecell class="treecell-indent"
	value="Printing Option"
	onclick="document.getElementById('ad
	vance').setAttribute('src','help.htm
	l#toolbar');
	document.getElementById('picture').s
	etAttribute('src','../picture/menu.j
	pg');"/>
	</treerow>
</treeitem>

<treeitem container="false">
	<treerow>
	<treecell class="treecell-indent"
	value="Search"
	onclick="document.getElementById('ad
```

```
                                    vance').setAttribute('src','help.htm
                                    l#toolbar');
                                    document.getElementById('picture').s
                                    etAttribute('src','../picture/menu.j
                                    pg');"/>
                                    </treerow>
                             </treeitem>

                             <treeitem>
                                    <treerow>
                                    <treecell class="treecell-indent"
                                    value="Main Window"
                                    onclick="document.getElementById('ad
                                    vance').setAttribute('src','help.htm
                                    l#window');
                                    document.getElementById('picture').s
                                    etAttribute('src','../picture/genera
                                    l.jpg');"/>
                                    </treerow>
                             </treeitem>
                      </treechildren>
               </treeitem>
...
```

Figure 3.7 is the snapshot of the tree. The bold lettering codes are those which generating each title of the tree. Kirrkirr Help is the root of Main menu, History, Printing option, Search and Main Window.

The onclick action enclosed for each action performed on the tree is to open two files on two different frames. The examples are shown on figure 3.8 and 3.9. When Kirrkirr Help is clicked, the “Picture” window will redirect to help.jpg while the “Description” frame will open top part of “help.html”.

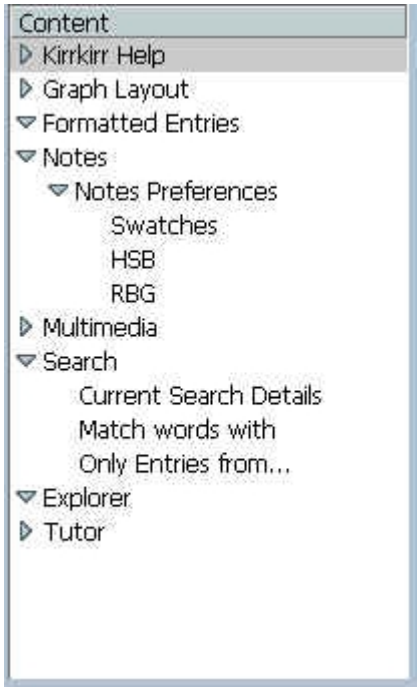


Figure 3.7 XUL tree examples

Figure 3.8 Kirrkirr Help Window

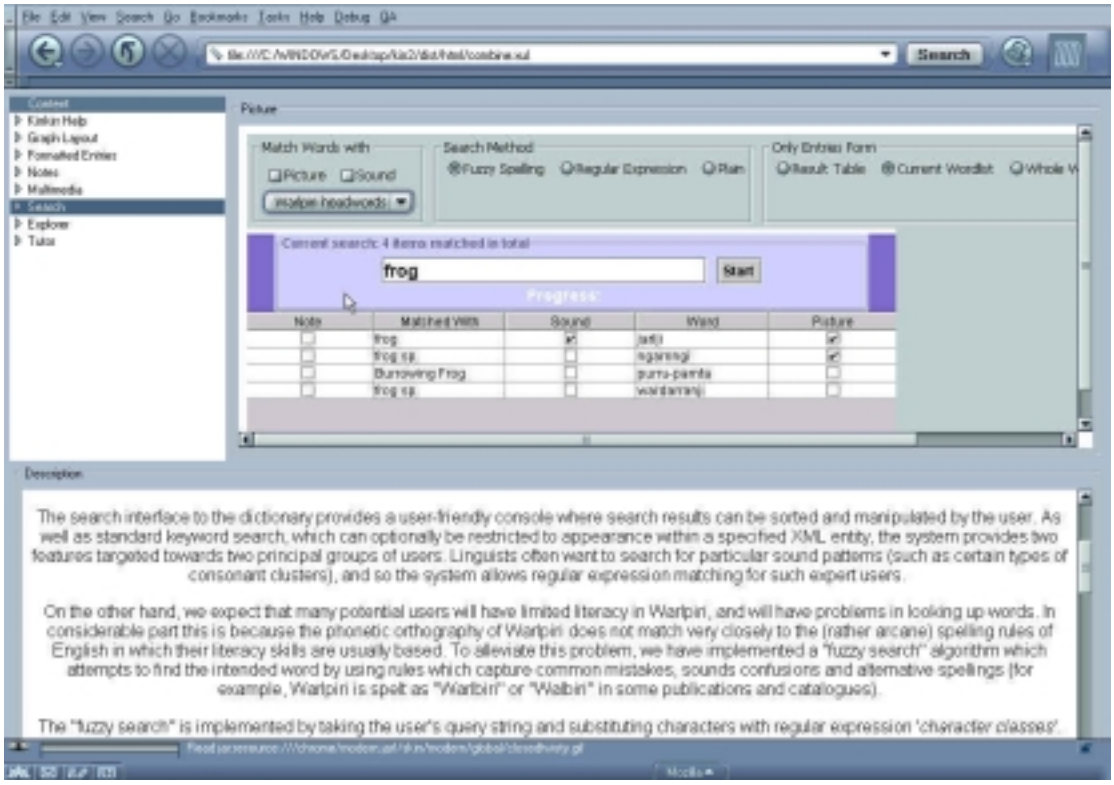


Figure 3.9 Kirrkirr Help Window, when Search is selected, the frames will display search panel picture and search panel explanation

- Adding action in XUL is by delineating the onAction component = java script method. Java script here is utilized to connect the java method from the previous application.

- Removing particular button and listener from java code and trade it with XUL code, actually has abridged the size of java classes (which will be loaded by the browser every time the initialisation of applet has been requested).

- For the ease of debugging, every XUL files created as independent as possible one to another (on every independent window, meaning it can be viewed independently) even if, later on they should be merge into single window. This also helps the readability of the code, as it is the same as breaking the code into function and to serve a request, simply by calling the function.

```
. . .<!—GraphLayout.xul files-- >
<window. . .>

       <script language="Javascript"
       src="chrome://global/content/globalOverlay.js">
       </script>

       <script language="JavaScript"
       src="chrome://global/content/strres.js">
       </script>

       <script language="JavaScript"
       src="chrome://communicator/content/bookmarks/bm-props.js">
       </script>

       <box id="Network">
       <html:div align="left">

       <html:table border="0">
       <html:tr>
       <html:td width="760" height="200">

              <titledbox width="760" height="200" orient="vertical">
                     <title>
                     <text value = "Description"/>
                     </title>

              <iframe id="graph-help" width="560" height="200"
              src="html/graph_help.html"/>

              <button id="more" value="More" class="dialog"
              onclick="window.open('html/graph.html');"/>
              </titledbox>
       </html:td>
       <html:td width="160" height="200">

       </html:td>
       </html:tr>
       </html:table>

       </html:div>
       </box>
</window>
```

The above code can be inspected on separated window as independent file as shown on figure 3.10. The bold line on the code above indicates the line where the widgets is given an identity and later will be called on the Kirrkirr-botpanel.xul file.

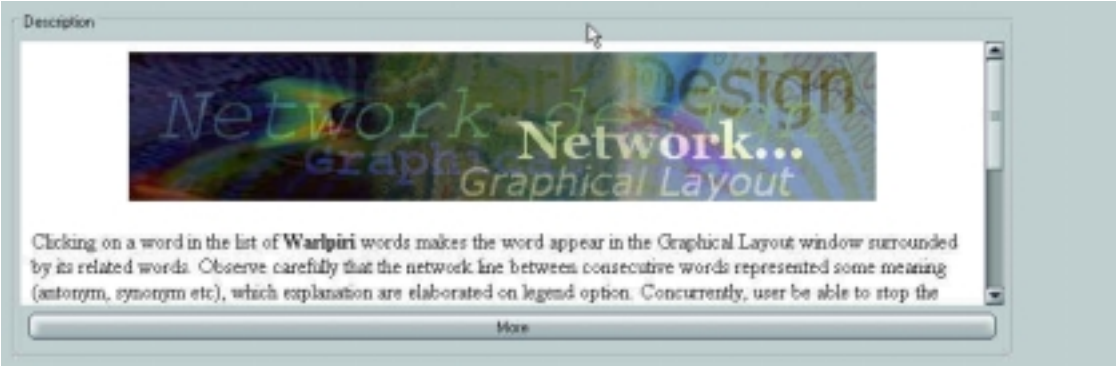


Figure 3.10 GraphLayout.xul files

```
. . .<!— Kirrkirr- botpanel.xul files, include graphLayout.xul file-- >
<?xul-overlay href="graphLayout.xul"?>
<?xul-overlay href="formatted.xul"?>
<?xul-overlay href="notes.xul"?>
<?xul-overlay href="multimedia.xul"?>
<?xul-overlay href="search.xul"?>
<?xul-overlay href="explorer.xul"?>
<?xul-overlay href="tutor.xul"?>
<?xul-overlay href="general.xul"?>
<?xul-overlay href="profile.xul"?>

. . .
<window.. .>

. . .

<tabcontrol id="help-option" orient="vertical">
       <tabbox>
              <tab value="General"/>
              <tab value="Networks Guide"/>
              <tab value="Entry Guide"/>
              <tab value="Notes Guide"/>
              <tab value="Multimedia Guide"/>
              <tab value="Search Usage"/>
              <tab value="Explorer Option"/>
              <tab value="Tutor Selection"/>
```

```
            <tab value="Profiles"/>

        </tabbox>

        <tabpanel>

        <box align="vertical" id="General"/><!--Kirrkirr Help-->
        <box align="vertical" id="Network"/><!--graphical layout-->
        <box align="vertical" id="Formatted"/><!--Formatted entries-->
        <box orient="vertical" id="Notes"/><!--Notes-->
        <box align="vertical" id="Multimedia"/><!--multimedia-->
        <box align="vertical" id="Search"/><!--advance search-->
        <box align="vertical" id="Explorer"/><!--Explorer-->
        <box align="vertical" id="Tutor"/><!--Tutor-->
        <box align="vertical" id="Profile"/><!--Profiles-->

  </tabpanel>
</tabcontrol>
</window>
```

Figure 3.11 The“Network guide” is came from graphLayout.xul file

3.4 Ant, Java Build Tool

[Jaques Bergeron, 2000] *Ant is a Java based build tool. In theory it is kind of like make without make's wrinkles.*

Why another build tool when there is already make, gnumake, nmake, jam, and others? Because all of those tools have limitations that its original author couldn't live with when developing software across multiple platforms. Make like tools are inherently shell based. They evaluate a set of dependencies and then execute commands not unlike what you would issue on a shell. This means that you can easily extend these tools by using or writing any program for the OS that you are working on. However, this also means that you limit yourself to the OS, or at least the OS type such as Unix, that you are working on.

Instead a model where it is extended with shell based commands, it is extended using Java classes. Instead of writing shell commands, the configuration files are XML based calling out a target tree where various tasks get executed. Each task is run by an object, which implements a particular Task interface.

We had make use of ant is for the sake of convenience since this application can not run fully online then users need to download the executable file or even the source file. For those who wish to build the source code, ant is going to be helpful. First, divide the folder into groups and create the build file. Here, we already arrange the folder and create the build file so user can simply install ant and execute build.xml and the executable file will be produced.
The following is the example of build.xml file.

```
<project name="Kirrkirr" default="all" basedir=".">

        <!-- set global properties for this build -->
        <property name="src" value="src" />
        <property name="build" value="build" />
        <property name="dist" value="dist" />
        <property name="resource" value="resource" />
        <property name="picture" value="picture" />
        . . .

        <!-- change build.compiler to 'classic' if you don't have jikes installed -->
        <property name="build.compiler"  value="classic" />
```

```
<property name="classpath"  value="${lib}:${build}" />

<target name="prepare">
       <!-- Create the build directory structure used by compile -->
       <mkdir dir="${build}" />
       <mkdir dir="${dist}" />
       <mkdir dir="${dist}/html" />
       <mkdir dir="${dist}/xml" />
       <mkdir dir="${dist}/xsl" />
       <mkdir dir="${dist}/images" />
       <mkdir dir="${dist}/users" />
       <mkdir dir="${dist}/tutor" />
       <mkdir dir="${dist}/picture" />

       <copy file="${resource}/run/applet.html" todir="${dist}"
       overwrite="yes"/>

       <copy file="${resource}/run/Kirrkirr-main.xul" todir="${dist}"
       overwrite="yes"/>

       <copy file="${resource}/run/Kirrkirr-botpanel.xul"
       todir="${dist}" overwrite="yes"/>

       <copy file="${resource}/run/Kirrkirr.xul" todir="${dist}"
       overwrite="yes"/>
       . . .

       <copy todir="${dist}" overwrite="yes">
              <fileset dir="${resource}/index"/>
       </copy>
       <copy todir="${dist}/html" overwrite="yes" >
              <fileset dir="${resource}/html"/>
       </copy>
       <copy todir="${dist}/xsl" overwrite="yes" >
              <fileset dir="${resource}/xsl"/>
       </copy>
       <copy todir="${dist}/images" overwrite="yes" >
              <fileset dir="${resource}/images"/>

       . . .
</target>

<target name="compile" depends="prepare">
       <!-- Compile the java code from ${src} into ${build} -->
```

```
            <javac srcdir="${src}" excludes="*.old" destdir="${build}"
            classpath="${classpath}" />
        </target>

        <target name="dist" depends="compile">
            <!--create jar file -->
            <jar jarfile="${dist}/demo.jar" basedir="${build}" />
        </target>

        <target name="clean">
            <!-- Delete the ${build} and ${dist} directory trees -->
            <delete dir="${build}" />
            <delete dir="${dist}" />
        </target>

        <target name="all" depends="clean, dist">
        </target>

</project>
```

By using this ant and build.xml, we can simply type “ant” to compile all the codes and copy it to dist directory as well as jar all java class file indo demo.jar. Build directory is contrived as temporary directory storage of all java class file. To do the cleaning (deleting) we can type “clean” to remove build and dist directory that created upon compiling/executing ant.

Chapter 4 : Design Analysis

[Chua Whay Yong, 1999] *However the main objective of a good user interface design is to create minimum environment in which the user can accomplish tasks with minimum anxiety, confusion, fatigue and boredom.* In chapter 3 we have the Mark up Language Style Guidelines, this is more unto code better than to the GUI itself.

[James Hobart] *Good GUI designs don't happen naturally. They require that the developer learn and apply some basic principles, including making the design something the user will enjoy working with every day. They also require that the developer get as much experience as possible in working on and being exposed to good GUI designs. Remember, if you apply the principles and get some experience in good GUI design, your users will have an easier time getting their jobs accomplished using the GUIs you produce for them.*

The proceeding part will be the analysis on design in consideration of :

- *Not forgetting the user*

  Developers often design for what they know, not what the users know. This age-old problem occurs in many other areas of software development, such as testing, documentation, and the like. It is even more pernicious in the interface because it immediately makes the user feel incapable of using the product. Avoid this error diligently.

- *Give more control to the user*

  GUI designers' predilection for control is evident in applications that continually attempt to control user navigation by graying and blackening menu items or controls within an application. Controlling the user is completely contradictory to event-driven design in which the user rather than the software dictates what events will occur. As business changes at a faster pace, flexibility in user interfaces will become a key enabler for change. Since the on browser application is not fully tested and bug free, some of button and menu are “statically” disabled to elude user from activating particular action that may arouse an error in the application.

- *Too Many Features at the Top Level*

  Too many features may cause user confusion. Therefore, we try to remove redundant features from the main GUI window. Likewise, we should ensure that features used frequently are readily available. Avoid the temptation to put everything on the first screen or load the toolbar with rarely used buttons. Do the extra analysis to find out which features can go behind the panel instead of on the faceplate.

  In this Kirrkirr Browser Application, browser main toolbar has been removed so that user won't get confused. The redundant presentations are removed and more space is introduced.

- *GUI Successes*

  Successful GUIs share many common characteristics.

  Most importantly, good GUIs are more intuitive than their character-based counterparts. One way to achieve this is to use real-world metaphors whenever possible. For example, an application we recently examined used bitmaps of Visa and MasterCard logos on buttons that identified how a customer was going to pay. This graphical representation was immediately intuitive to users and helped them learn the application faster.

  Another important characteristic of good GUIs is speed, or more specifically, responsiveness. Many speed issues are handled via the design of the GUI, not the hardware.

  Depending on the type of application, speed can be the make-or-break factor in determining an application's acceptability in the user community. For example, if this application is oriented toward online transaction processing (OLTP), slow performance will quickly result in users wanting to abandon the system.

  We can give a GUI the appearance of speed in several ways. Avoid repainting the screen unless it is absolutely necessary. Another method is to have all field validations occur on a whole-screen basis instead of on a field-by-field basis. Also, depending upon the skills of the user, it may be possible to design features into a GUI that give the power user the capability to enter each field of each data record rapidly.

Such features include mnemonics, accelerator keys, and toolbar buttons with meaningful icons, all of which would allow the speed user to control the GUI and rate of data entry.

But for Kirrkirr case, the speed and responsiveness are dependent on the LiveConnect performance and number of jar file loaded. But, as chapter 3 had mentioned that some of GUI parts that can not be implemented on XUL are remained on Java as well as the number of applet is impossibly to be reduced.

The only way to speed up the performance is to remove all the Java part and tries to rebuild everything in XML, XSL, XUL, XPCom, Java Script or any kind of language that are more suitable to run on browser. But this may be time consuming since perhaps only 10% part of the code can be reuse. For example, we transform the java applet on formatted entry to HTML file using XSL, but this is supported in IE and not Netscape.

- *Understand People*

  Applications must reflect the perspectives and behaviours of their users. To understand users fully, developers must first understand people because we all share common characteristics.

  People learn more easily by recognition than by recall. Always attempt to provide a list of data values to select from rather than have the users key in values from memory. The average person can recall about 2,000 to 3,000 words; yet can recognize more than 50,000 words.

- *Be Careful Of Different Perspectives*

  Many designers unwittingly fall into the perspective trap when it comes to icon design or the overall behaviour of the application. To signify some functions, the designer put much artistic effort into creating an icon. Unfortunately, the users of the system had no idea what metaphor the icon was supposed to represent even though it was perfectly intuitive from the designer's perspective.

The icons being used on this Kirrkirr most of it assigned from Mozilla standards icon instead of making use of former gif and jpg files previously used on the Java Application.

- *Design for Clarity*

  GUI applications often are not clear to end-users. One effective way to increase the clarity of applications is to develop and use a list of reserved words. A common complaint among users is that certain terms are not clear or consistent.

  We often see developers engaging in spirited debates over the appropriate term for a button or menu item, only to see this same debate occurring in an adjacent building with a different set of developers. When the application is released, one screen may say "Item," while the next screen says "Product," and a third says "Merchandise" when all three terms denote the same thing. This lack of consistency ultimately leads to confusion and frustration for users.

  Table 4.1 gives an example of a list of reserved words. An application-development group might complete and expand the table with additional reserved words.

Table 4.1 Reserved Words

Text	Meaning And Behaviour	Appears On Button	Appears On Menu	Mnemonic Keystrokes	Shortcut Keystrokes
OK	Accept data entered or acknowledge information presented and remove the window	Yes	No	None	<Return>or <Enter>
Cancel	Do not accept data entered and remove the window	Yes	No	None	Esc
Close	Close the current task and continue working with the application; close view of data	Yes	Yes	Alt+C	None
Exit	Quit the application	No	Yes	Alt+X	Alt+F4
Help	Invoke the application's Help facility	Yes	Yes	Alt+H	Fl
Save	Save data entered and stay in current window	Yes	Yes	Alt+S	Shift+Fl2
Save As	Save the data with a new name	No	Yes	Alt+A	F12

Undo	Undo the latest action	No	Yes	Alt+U	Ctrl+Z
Cut	Cut the highlighted characters	No	Yes	Alt+T	Ctrl+X
Copy	Copy highlighted text	No	Yes	Alt+C	Ctrl+C
Paste	Paste the copied or cut text at the insertion point	No	Yes	Alt+P	Ctrl+V

- *Design for Consistency*

  Good GUIs apply consistent behaviour throughout the application and build upon a user's prior knowledge of other successful applications. When writing software for business applications, provide the user with as much consistent behaviour as possible. For example, if we are using Ctrl-P to open the preferences of Kirrkirr Java Application, is the best to keep it that way on the browser version.

- *Provide Visual Feedback*

  If we've ever found ourselves mindlessly staring at the hourglass on our terminal while waiting for an operation to finish, we know the frustration of poor visual feedback.

  Our users will greatly appreciate knowing how much longer a given operation will take before they can enjoy the fruits of their patience. As a general rule, most users like to have a message dialog box with a progress indicator displayed when operations are going to take longer than seven to ten seconds.

  This number is highly variable based on the type of user and overall characteristics of the application. On Kirrkirr, browser progress bar is used.

- *Provide Audible Feedback*

  However, audible feedback can be useful in cases where you need to warn the user of an impending serious problem, such as one in which proceeding further could cause loss of data or software. Allow users to disable audio feedback, except in cases when an error must be addressed. This is optional since not all users make use of speaker.

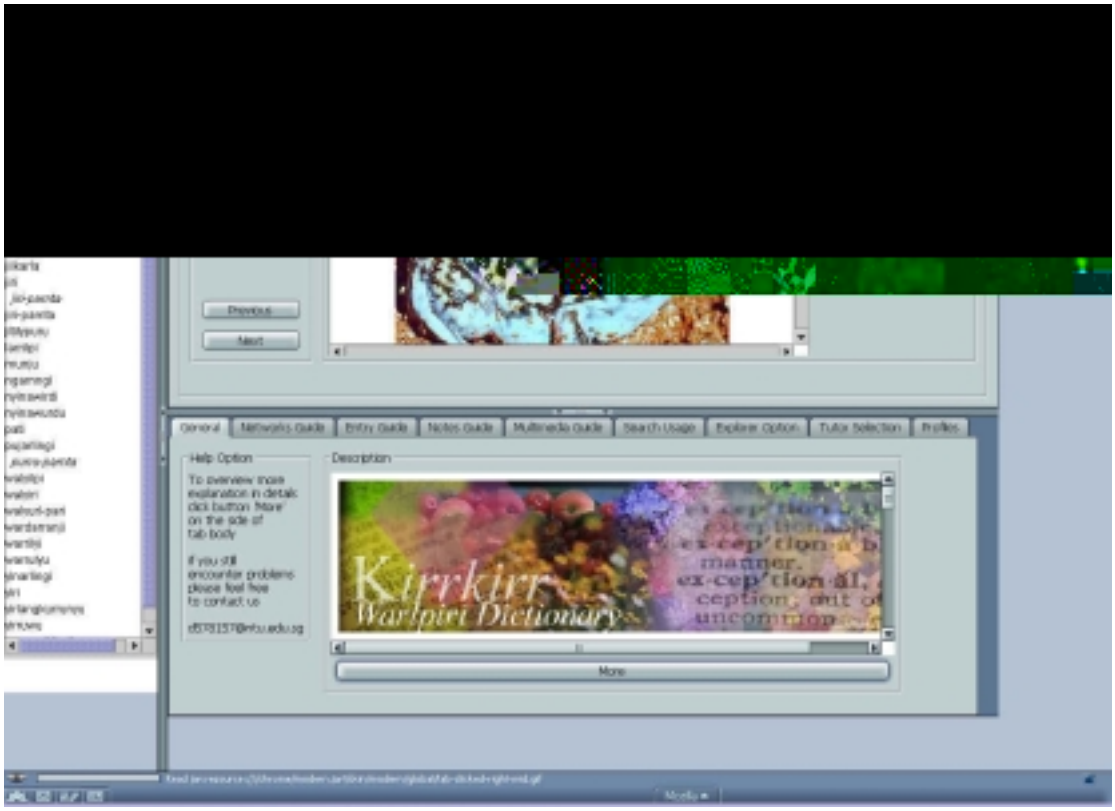


Figure 4.1 Notice that on the left side corner of the window there are common progress bar

- *Keep Text Clear*

  Developers often try to make textual feedback clear by adding a lot of words. However, they ultimately make the message less clear. Concise wording of text labels, user error messages, and one-line help messages is challenging.

  Textual feedback can be handled most effectively by assigning these tasks to experienced technical writers. Therefore, the first main window only display short help and references, details can be found by clicking on “more” option.

- *Provide Traceable Paths*

  Providing a traceable path is harder than it sounds. It starts with an intuitive menu structure from which to launch our specific features.

  We must also identify areas where we can flatten the menu structure and avoid more than two levels of cascading menus. Providing a descriptive title bar within each dialog box will help greatly to remind the user what menu items or buttons were

pressed to bring them to the window now in focus. In Kirrkirr, some of windows and dialog name is transfigured for this purpose. It is also good to set forth-adding tool tips for button and menu.

- *Provide Keyboard Support*

  Keyboards are a common fixture on users' desktops and provide an efficient means to enter text and data. With the introduction of GUI applications, we often assume users will embrace a mouse as the primary interactive device. This can become time-consuming and inefficient for the touch typist or frequent users of your application.

  Keyboard accelerators can provide an efficient way for users to access specific menu items or controls within a window. The accelerators used should be easy to access and limited to one or two keys (such as F3 or Ctrl-P). Keyboards have limitations in the GUI world, such as when trying to instrument direct-manipulation tasks like drag and drop, pointing, and resizing. This may be one of the considerations of future enhancement in Kirrkirr since this field is not really on the main concern meanwhile.

- *Watch the Presentation Model*

  A critical aspect that ties all these facets of the interface together is the interface's look and feel. The look and feel must be consistent. On the basis of users' experiences with one screen or one dialog box, they should have some sense of how to interact with the next screen or control.

  Searching the interface model for good design and continuity is most important. The model should involve careful decisions, such as whether the application will have a single or multiple document interfaces. The model also will validate how users perform their main tasks within the application.

  Identifying the appropriate presentation for the application will greatly facilitate the subsequent windows being developed since they will have a common framework to reside in. On the other hand, if you do not define the presentation model early in the design of your GUI, late changes to the look and feel of the application will be much more costly and time-consuming because nearly every window may be affected.

The alignment of Kirrkirr is redone to be more succinct and interactive. There are some relevant picture added since human are more engaged and attentive in information in the form of picture better than writings.

- *Modal vs. Modeless Dialogs*

  When we need input from the user, we often use a modal dialog box. Many developers as too constraining on the user have long shunned using modal dialogs. However, modal dialogs do have many uses in complex applications since most people only work on one window at a time.

  Try to use modal dialogs when a finite task exists. For tasks with no fixed duration, modeless dialogs will normally be the preferable choice with a major caveat: Try to keep the user working in no more than three modeless windows at any one time.

Table 4.2 When to Use Dialog Boxes Or Windows

Type	Description	Use	Example
Modal	Dialog box	Presentation of a finite task	File Open dialog box Save As dialog box
Modeless	Dialog box	Presentation of an ongoing task	Search dialog box History List dialog box Task List dialog box
Application Window	Window frame with document (child) windows contained within	Presentation of multiple instances of an object Comparison of data within two or more windows	Word Processor Spreadsheet
Document Window	Modeless dialog box or document window contained within and managed by Application window	Presentation of multiple parts of an application	Multiple views of data (sheets)
Secondary Window	Primary window of a secondary application	Presentation of another application called from parent	Invoke Help within an application

- *Control Design*

  Controls are the visual elements that let the user interact with the application. GUI designers are faced with an unending array of controls to choose from. Each new

control brings with it expected behaviours and characteristics. Choosing the appropriate control for each user task will result in higher productivity, lower error rates, and higher overall user satisfaction.

Table 4.3 Control Usage

Control	Number Of Choices In Domain Shown	Type Of Controls
Menu Bar	Maximum 10 items	Static action
Pull-Down Menu	Maximum 12 items	Static action
Cascading Menu	Maximum 5 items, 1 cascade deep	Static action
Pop-up Menu	Maximum 10 items	Static action
Push-button	1 for each button, maximum of 6 per dialog box	Static action
Check Box	1 for each box, maximum of 10 to 12 per group	Static set/select value
Radio Button	1 for each button, maximum of 6 per group box	Static set/select value
List Box	50 in list, display 8 to 10 rows	Dynamic set/select value
Drop-down List Box	Display 1 selection in control at a time, up to 20 in a drop-down box	Dynamic set/select single value
Combination List Box	Display 1 selection in control at a time in standard format up to 20 in a drop-down box	Dynamic set/select single value; add value to list
Spin Button	Maximum 10 values	Static set/select value
Slider	Dependent on data displayed	Static set/select value in range

Finally, try to keep the basic behavior and placement of these controls consistent throughout the application. As soon as we change the behavior of these basic controls, our user will feel lost. Make changes thoughtfully and apply the changes consistently.

- *Applying Design Principles*

  Understanding the principles behind good GUI design and applying them to our applications can be a challenge. Let's examine an application to see how these principles can result in an improved interface.

4.1 Main Menu and Toolbars

The design on the main menu is as below (Figure 4.2). This toolbar is approximately in similar with the application outline. Some of the button and menu are disabled for some reason (security reason). Transformation had settled so far such as : menu "Option"→"Preferences" is deducted from the list, since there will be "preferences" button is placed on every related panel. Previously, preferences option leading to network preferences, notes display option, as well as formatted entry display figures (Figure 4.3)

The browser menu and navigation bar is removed and replaced with Kirrkirr toolbar, but as a user, we may wish to reset browser related matters or others, for example, open a related links, change proxy setting, apply theme, do cut and paste, change language preferences or print a file. All of these options are by default provided by Mozilla browser, operating under complete window display. Therefore, instead of re-implement the options, Kirrkirr has taken some of Mozilla menu options that user probably need during executing the application, and inserting those between Kirrkirr menu.

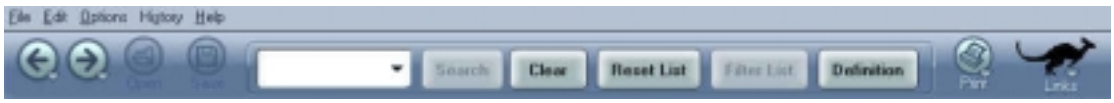


Figure 4.2 Main Toolbar of Kirrkirr

For the ease of use, the preferences window has been portioned to 3 independent parts that is network preferences, notes preferences and formatted entry preferences. Each of the preferences encompass they own button on every correspond panel. This is for the ease of use, to prevent user confusion.

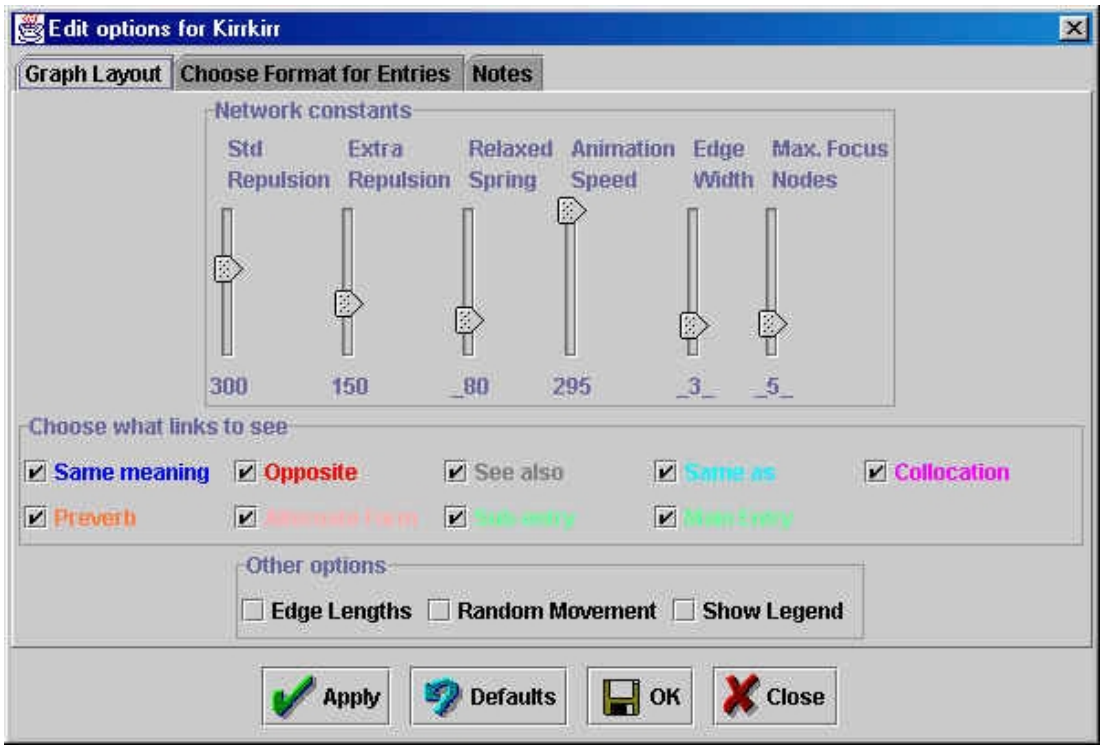


Figure 4.3 Previously, the "Preferences" option on Kirrkirr menu will open up this window

By way of illustration, the following network panel has network preferences button that will popup network preferences (Graph Layout tab panel) window only devoid of displaying any other tab panel.

The separation of the preference panel to each comparative applet in fact doesn't make the code much more intricate. This is done to evade designing more applets and induce the program become more complex. Because if we want to imitate exactly the original Kirrkirr GUI, like the original preferences window on Figure 4.3, we need to place on another independent applet communicate appropriately with Graph Demo (the applet class file name for GraphPanel.class), Notes Demo (the applet class file name for NotesPanel.class) and Formatted Entry Demo(the applet class file name for htmlPanel.class).

The window displaying Graph Panel Preferences, for example, is originally fabricated as a part of GraphPanel class, meaning-creating (in fact only through a function call which originally has been provided) this Kirrkirr Preferences window for Graph Panel Option not necessarily involving a new applet class. We bestow of Graph Demo as the applet class for

both preference window and Graph Layout panel. Therefore the code maintenance for the programmers is relatively easier.

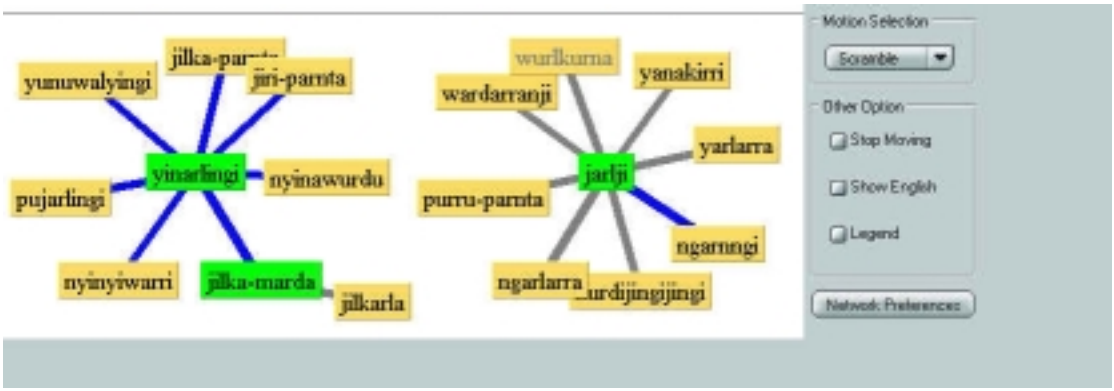


Figure 4.4 Graph Panel with "Network Preferences" button

Search field, search button, clear button, reset list and filter list button is allocated on the top so that the window section for window contents will be much bigger. According to the GUI guidelines, naming the window should be in relation with the name of the button in order for the user to be able to trace back. Therefore the name of the window now is : "Kirrkirr Preference Window".

An extra thing is the bottom pane present preferences, which are from another version of Kirrkirr. Originally the user can either have the top pane occupied the window and eradicating the bottom pane, or splits the window into bottom pane and top pane (where the user may elect what to display on bottom pane). But since in this design, it is decided not to have mirrored pane (as top pane and bottom pane are the same) to reduce numbers of applet loaded.

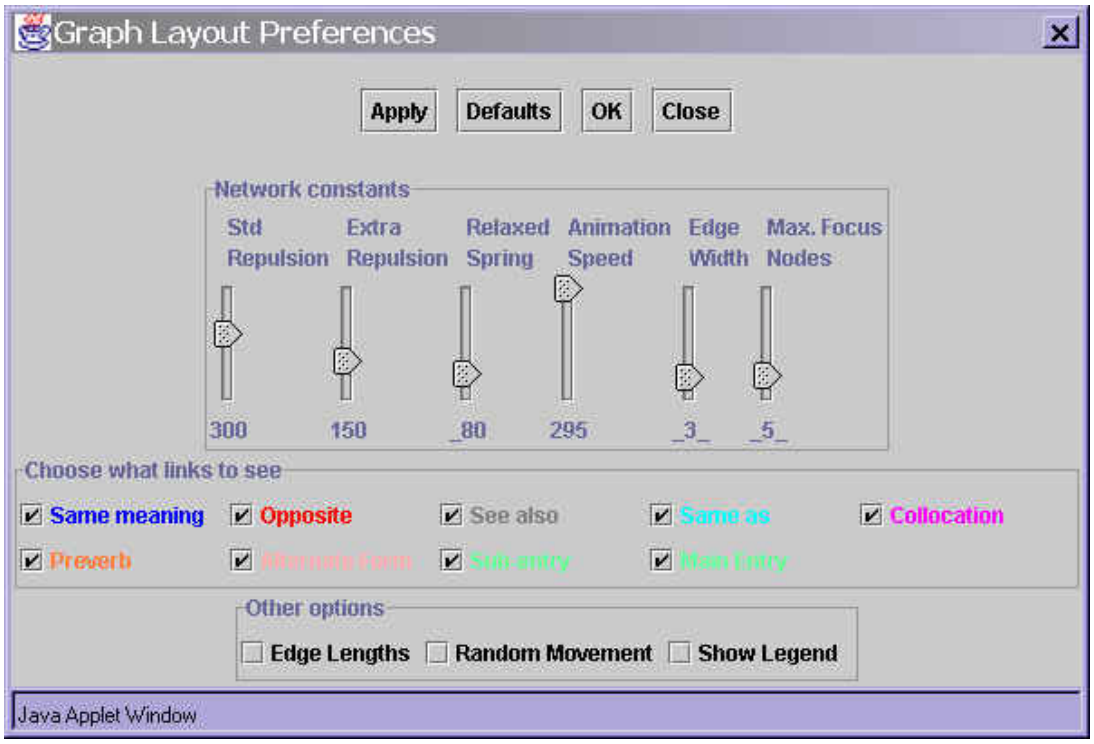


Figure 4.5 "Edit Option for Kirrkir (Graph Panel)" as part of previous preference window is now separated from another preference tab panel.

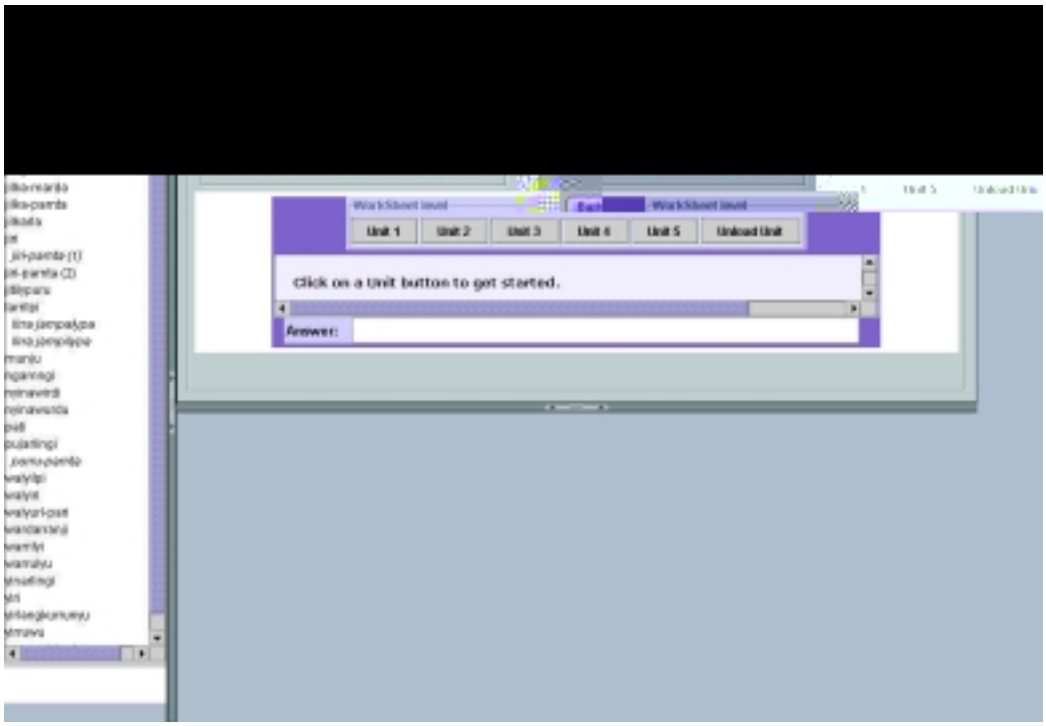


Figure 4.6a Hiding the bottom pane can be done by do a click on the arrow over the sidebar

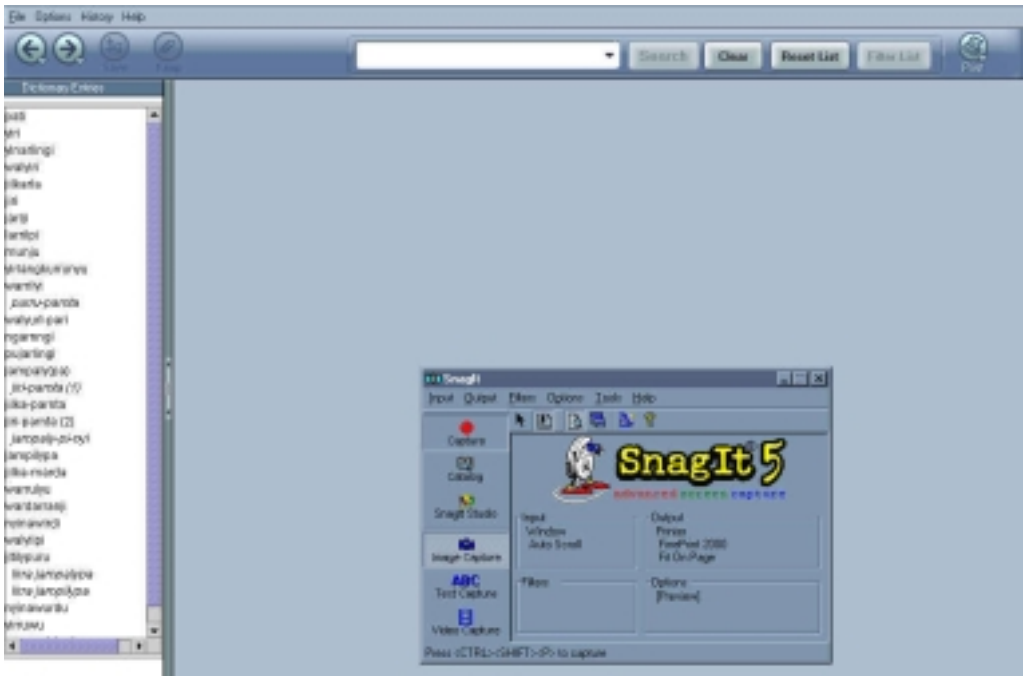


Figure 4.6b The part of GUI that is hided can be overlapped with another application windows.

Print button has it own history so that user may view what they have printed before. Some of the thing that are strenuous to implement in XUL are remained in Java applet, the main difficulties is that, more applets means more loading time needed by the browser, need more synchronization and more difficult to control the communication one to another.

Word arrangement form Menu list of Option to arrange the word with sub-words, multiple, alphabetical order, with rhyme, with according to frequency, and the plain arrangement are remain the same as the old application (figure 4.7)

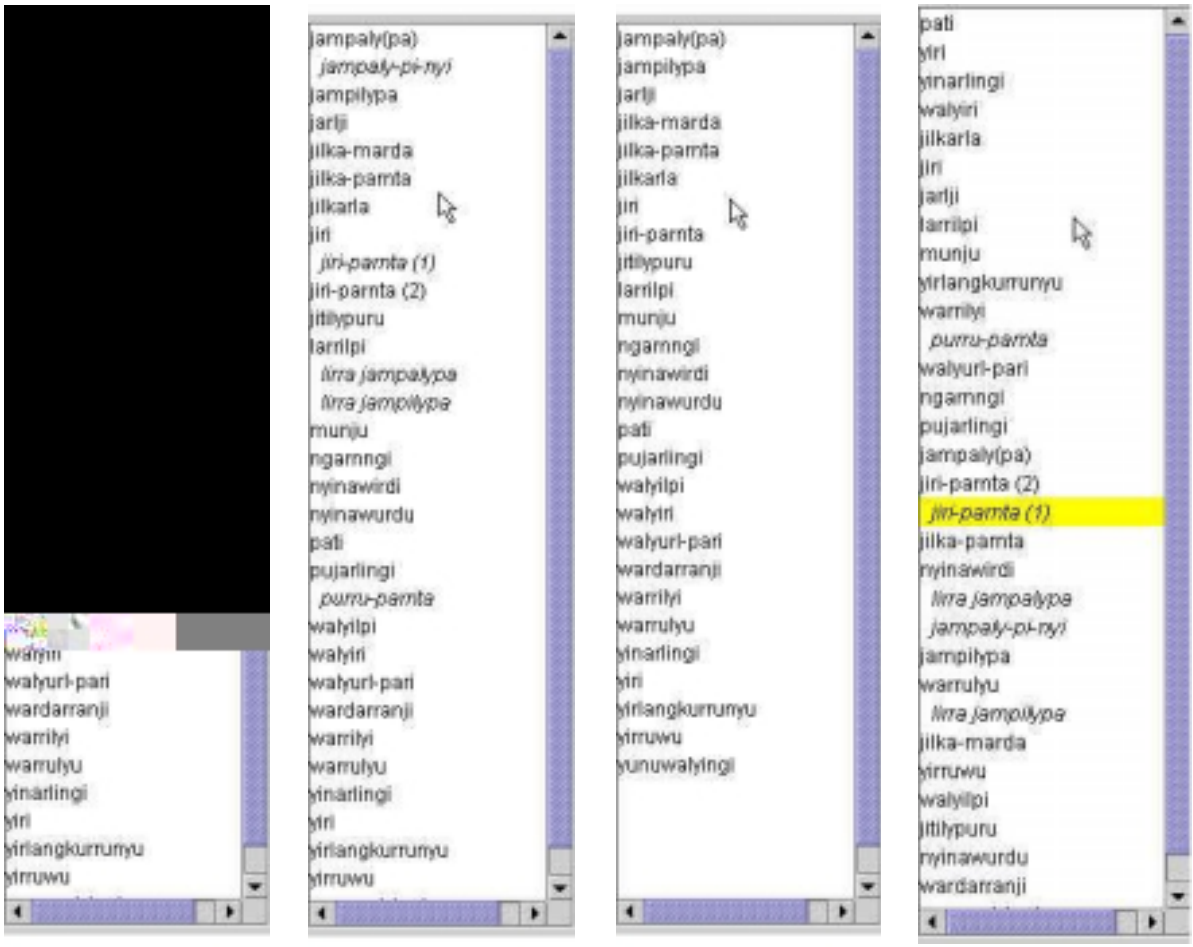


Figure 4.7 from left to right. Display of headwords on plain, rhyme with number of multiples, non subwords list and according to frequency and multiples list.

4.2 Help Modules

The earliest application has Java help application with its finished help set and help broker. In this Mozilla application, all of java help component are eliminated and change to all XUL files with numbers of frame.

The left hand content box refers to the content of the help itself. As the user travels to the table of content, the image frame and description frame will swap accordingly. This is moderately done in XUL with a little help of java script to link every action of the selection made. Iframe and tree tags from XUL are being employed as well. The details of the tree can be found is chapter 3.

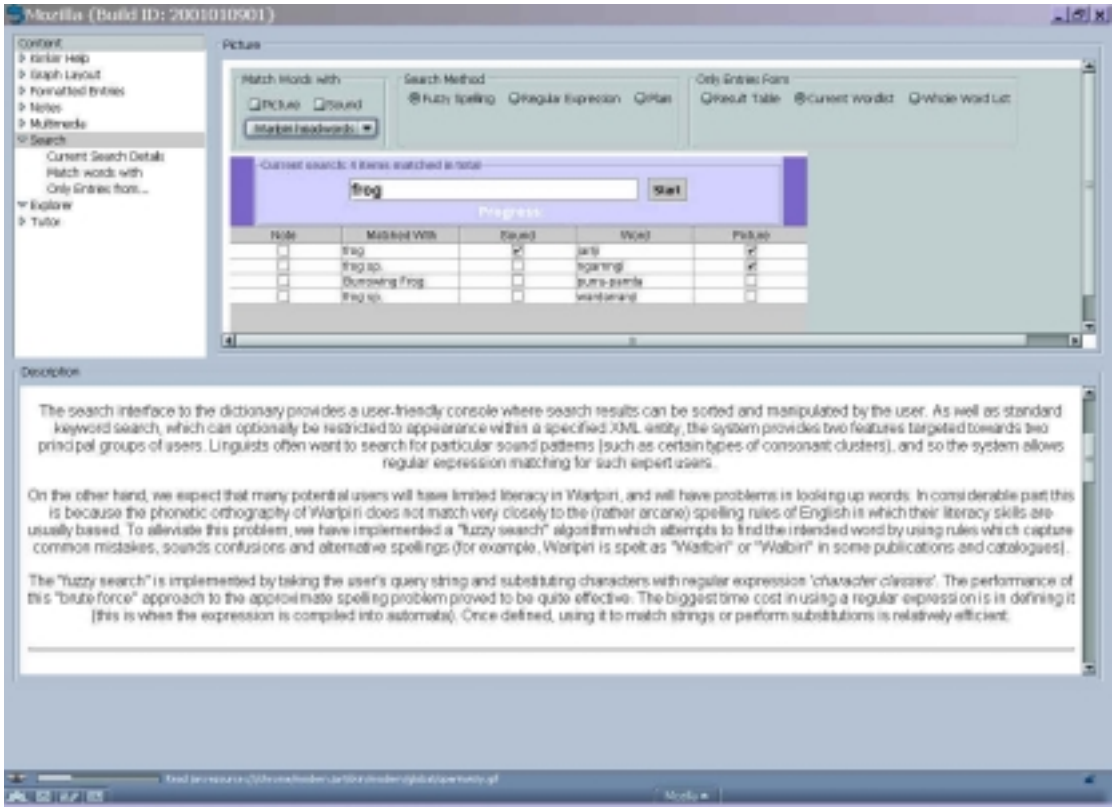


Figure 4.8 Kirrkirr menu → Help → Contents..

On the main window (Figure 4.9), the brief description of every panel is positioned on the bottom pane of the window. XUL panel is more straightforward to be maintained for upcoming expansion as it shows before. But the development (adding more panel) is restricted to the width of the window.

We can hide the bottom panel part as well as the headwords (dictionary entry on the left most side of the main window) as shown on figure 4.6a. This has simplified the java coding since the part of the grip is not being used and instead XUL GUI has substituted it. Last time, to conceal and reveal the bottom panel, users need to select on Menu toolbar under Option menu. But now they just need to click on the grip/arrow on the sidebar. Therefore the Option→show bottom can be removed from menu options.

The bottom frames of main window (the help panels) are sourced from HTML file. Therefore, user may do a right click and open this option on a new window if they wish to (Figure 4.10).

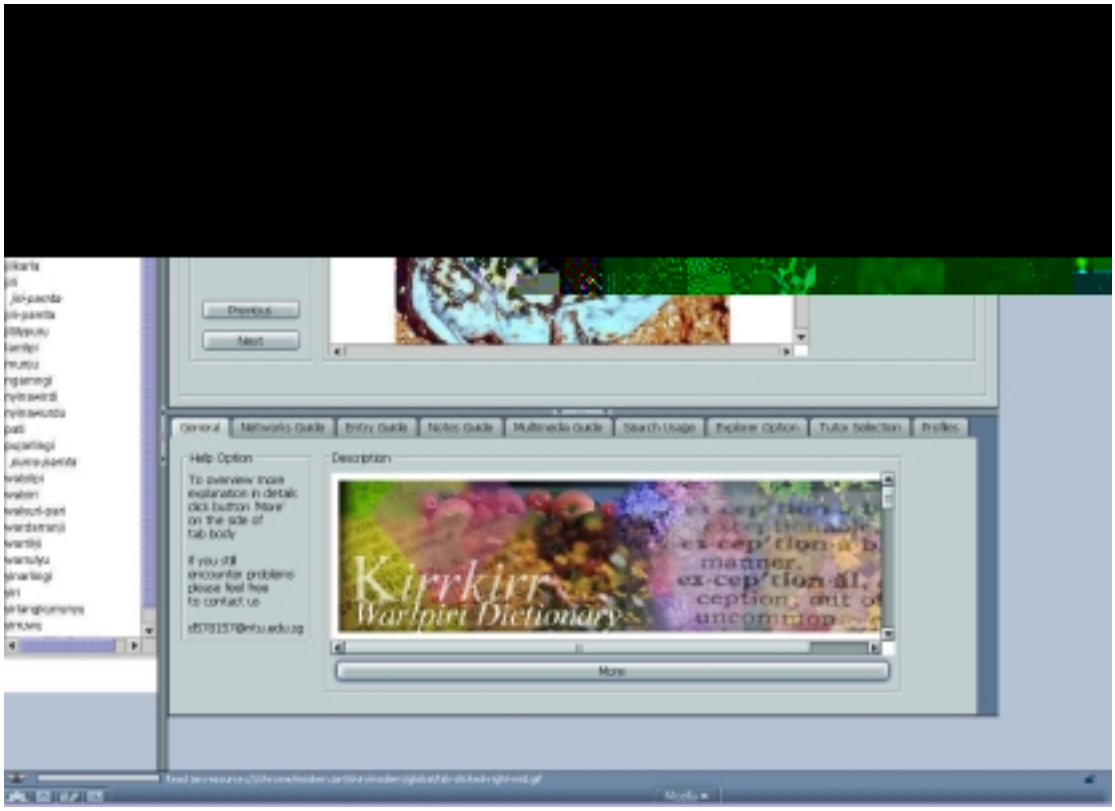


Figure 4.9 Kirrkirr brief help placed on the bottom part of the main window

Those help panels the application has: general (to explain how to use help and Kirrkirr in general), network guide, entry guide, notes guide, multimedia guide, search guide, explorer option, tutor selection and profiles.

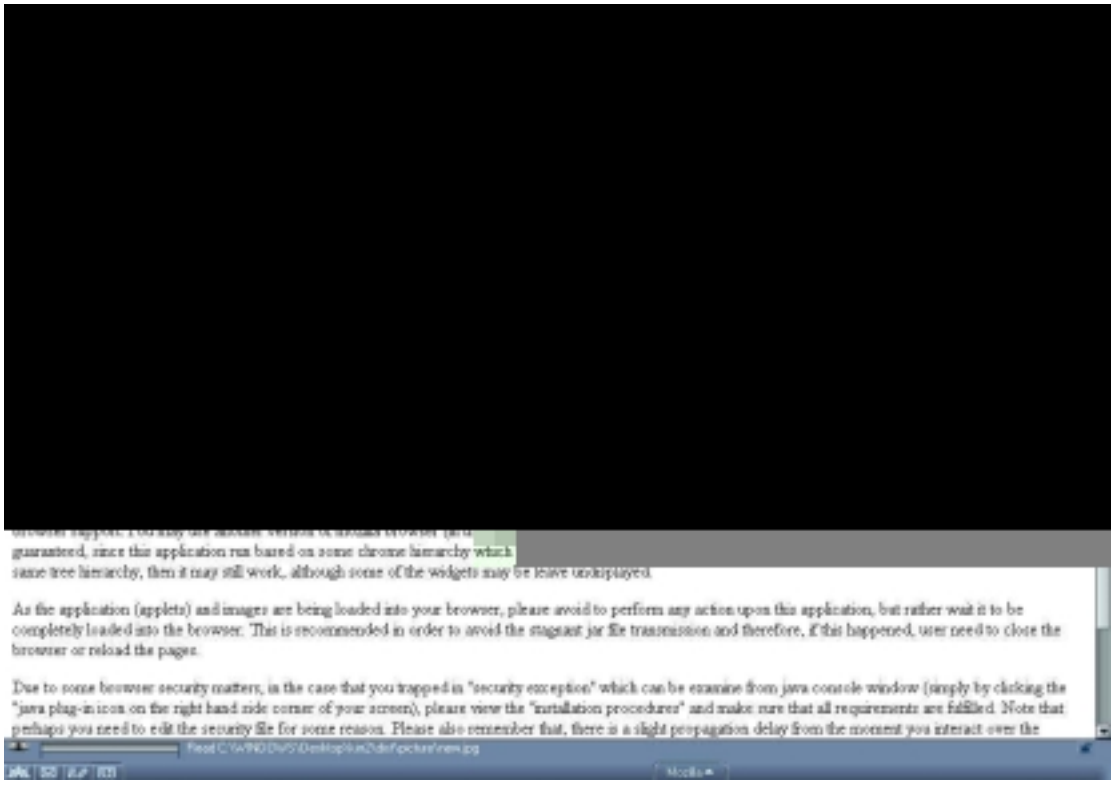


Figure 4.10 Do The right click on the bottom panel, and select "Open on new browser window" will open the panel on separated window frame.

General Help offers the description of general use of Kirrkirr and link to main menu, toolbar and main screen brief help. Button "More" is to link up to main Help pages, which basically describes the module in more particulars. It enclose the same thing as Menulist Help→Contents.. option but different way of representation.

Figure 4.11 General Help in Kirrkirr

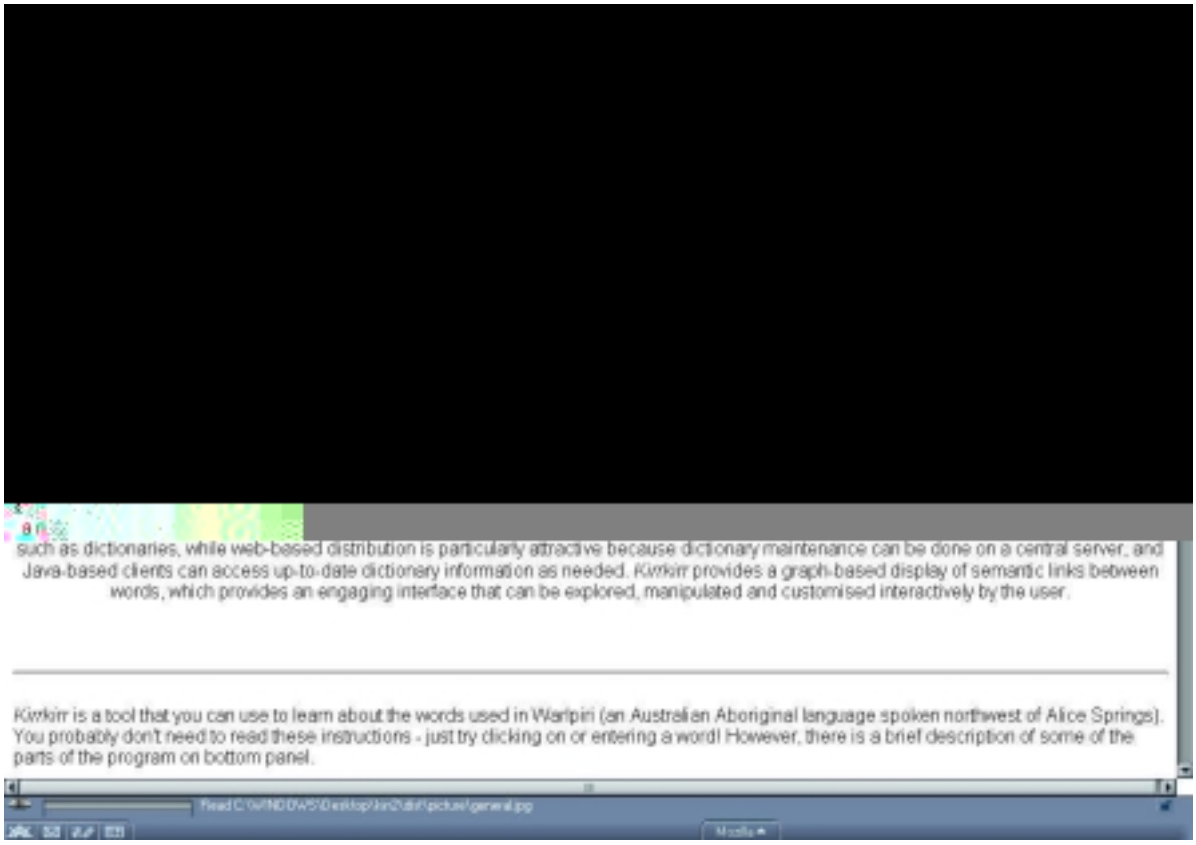


Figure 4.12 "More" on the details of General Help in Kirrkirr figure 4.10

The other Help short cut is from Help menu list on the main toolbar. As mention in chapter 3, this feature has replaced Java Help on previous application. The design are taking the benefit of XUL tree and framing widgets. This is simple to be preformed in XUL rather than in Java or HTML.

In Java, we need to do more programming and create help broker and help set besides some HTML files which presenting the help database itself. In fact, we just need to create one XUL files and use the old HTML file as the source of help database. The programming part is more straightforward and separated from Java class, so modification can be done without involving compilation and Java coding.

Figure 4.13 Mozilla Help module replacing Java Help

Hence, with XUL, we are trying to separate the GUI part and control part to enhance maintainability, modularity, flexibility and readability of the program. In this module, we link up every selection with corresponding picture and explanation (refer to chapter 3) as demonstrated on figure 4.14.

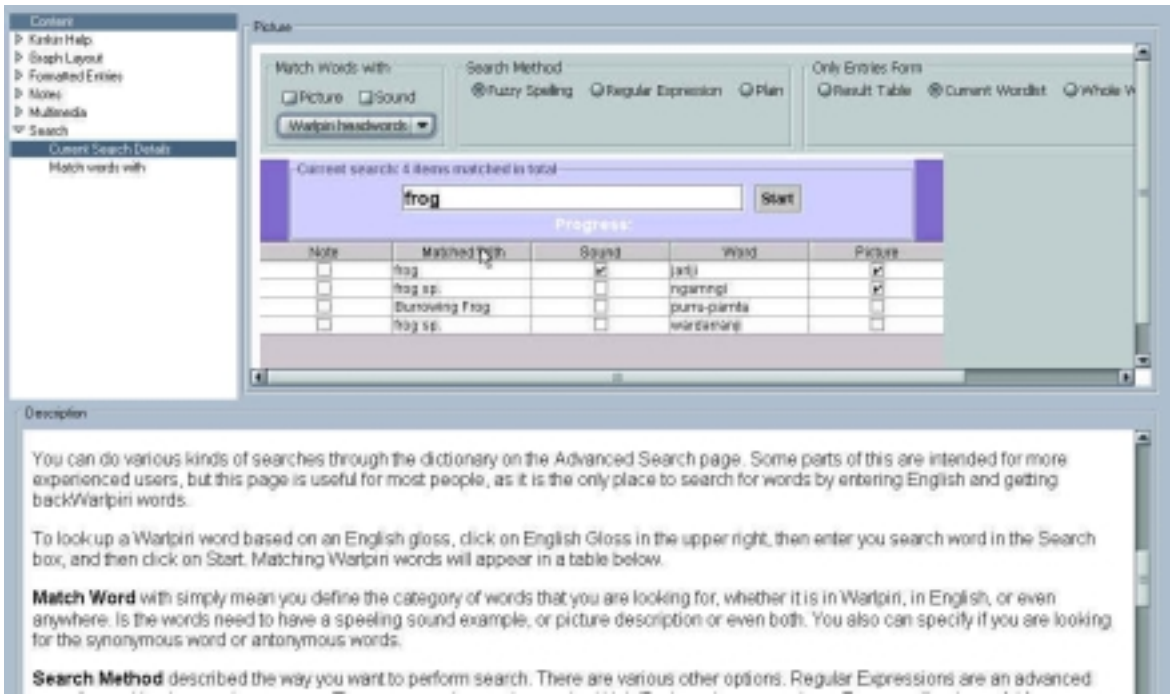


Figure 4.14a Demo of Mozilla Help being used on "Current Search Details"

Figure 4.14 Demo of Mozilla Help being used on "Graphical Layout"

Figure 4.15 Display of each panel of the bottom panel

4.3 Main Panel

Ideally, it is superior to have all parts of the application shape in java script and XUL only without involving any java components. But unfortunately, some parts of the dictionary are demanding to be implemented in java script or XUL, for example dynamic box on network panel, explorer window and etc. Hence, the panel formed partly, for some convoluted algorithm the application remain as java applet and for interface the application make use of XUL.

The idea is to have one main applet controlling one panel and be in touch with other applet and headword applet. This will help to take back to first principles of the coding a lot since the programmers do not need to be concerned about previous functionality of buttons and windows. All they need to do is to link up every new button with the method stated in Java class included inside the applet.

From the previous application, the top panel can occupied the entire window or splits into 2-mirrored panel. Because generating panel is easier in XUL rather in Java, then the java pane is removed and changed into XUL pane. So now there are 7 panels for the main panel: network, notes, multimedia, formatted entry, explorer, semantic search and tutor panel.

Then to have 2 mirrored pane meaning that the Java code should be changed to adjust the concurrent action of all 14 applets, which will be very complicated and complex (since we need to consider how the bottom part and top part to communicate and sharing the same cache and resource).

Another issue is to reduce or enlarge the applet size, programmers can define new size of the height and width since, the design pass 5 parameters to the applets (including height and width)

```
<html:applet name = "GraphDemo"
code ="GraphPanelDemo.class"
archive = "demo.jar, gnu.jar, oro.jar, jh.jar, parser.jar, xt_old.jar"
width = "600" height = "250">

<html:param name="xml_file" value="Wrl.xml"/>
<html:param name="index_file" value="Wrl.clk"/>
<html:param name="html_folder" value="html"/>
<html:param name="applet_width" value="600"/>
<html:param name="applet_height" value="250"/>
<html:param name="applet_num" value="0"/>
</html:applet>
```

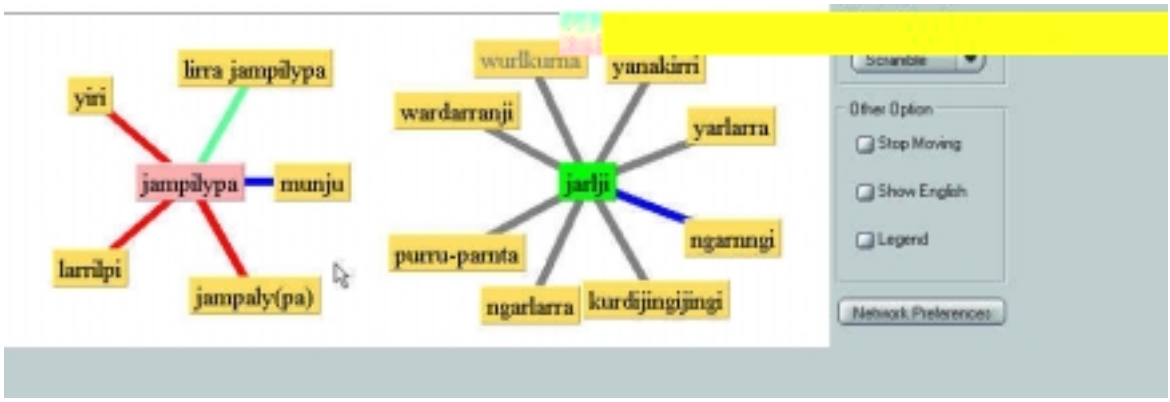


Figure 4.16 Network Panel

Network Panel

Motion Selection as it on the application before: Scramble, Shake, and Clear. Popup menu is there so that less area is to be used. Popup menu also denotes there will be only one assortment at a time. Other Option: Stop Moving, Show English and Legend can all be selected on the same time (multiple selection). And Network Preferences to display the addition option to change the display of Graph Panel.

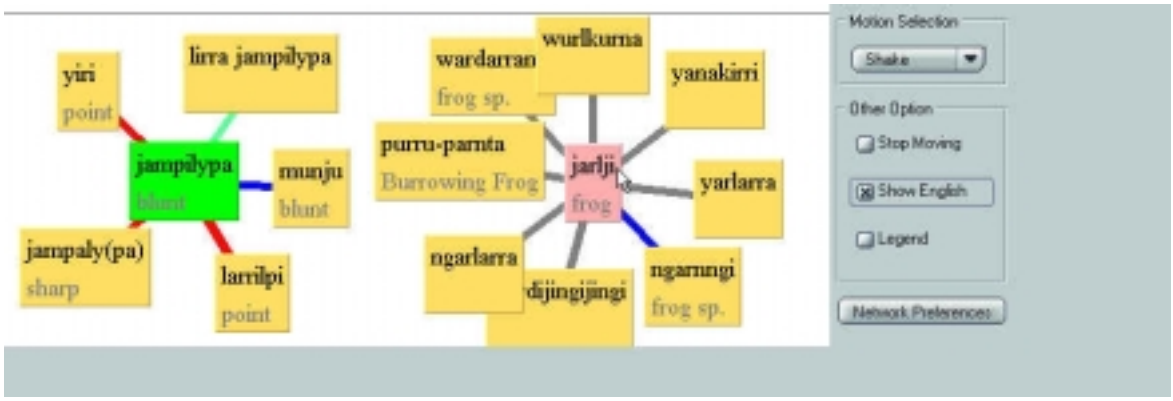


Figure 4.17a Show English

The right click mouse button on the graph Layout area can brought the user to 8 selection, they are : find in list, sprout, collapse, see definition, delete, anchor down, release, and option. So, the user may be able to open the network preferences by doing mouse right click.

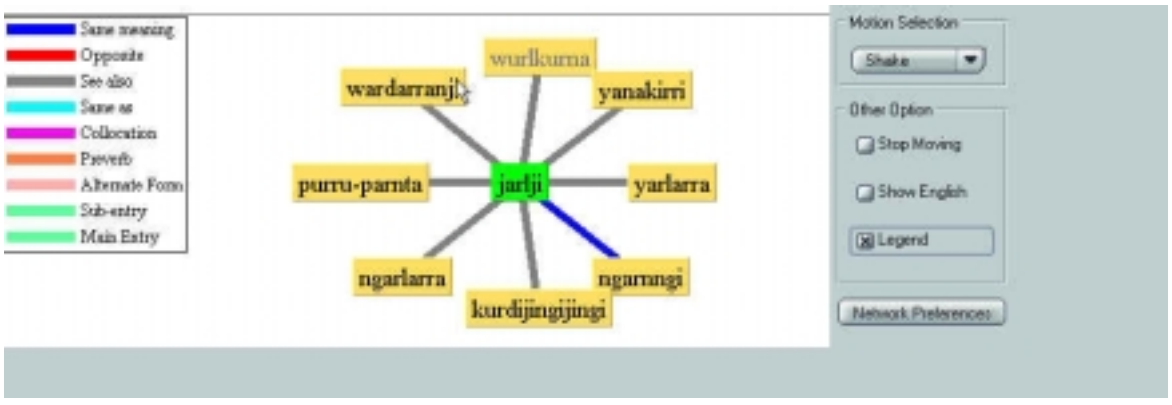


Figure 4.17b Show Legend

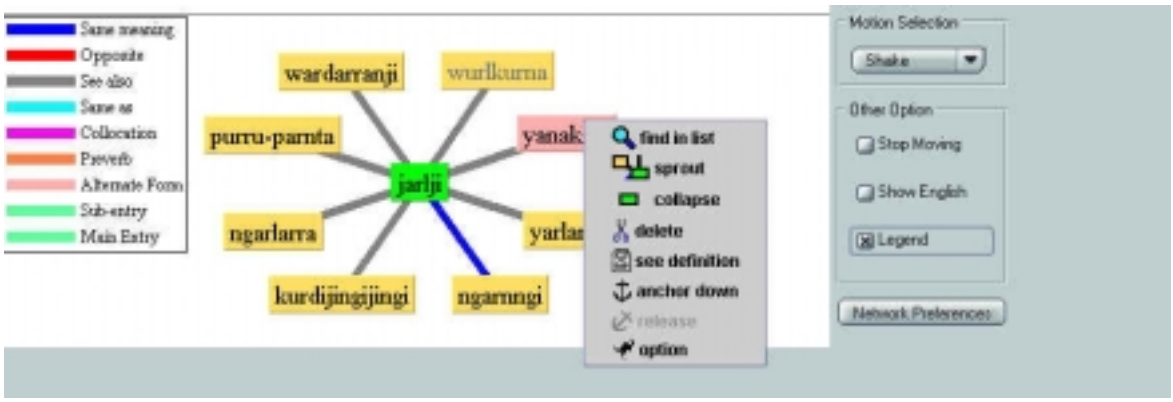


Figure 4.18 Right click of Graph Panel Window

Formatted Entry Panel

Formatted entry works the same as in the java application except at the moment; the design is to exhibit the full entry of each word instead of only the definition of the word. In later java application, user may prefer whether they want details, description or full words entry. But here, the option of changing display is static or cannot be changed. This is because there are some exceptions raised if we force to select different display option after the page is loaded.

The restriction is because of this version of application can not act upon read, write or execute file from outside that are not secure for the system. Meanwhile, performing selection to adjust the display option (e.g. from "only definition" into "full entry") the browser needs to read input from file and parse it then display it. So, we only able do the parsing and reading from file once before it is loaded into the browser, therefore this selection cannot be varied.

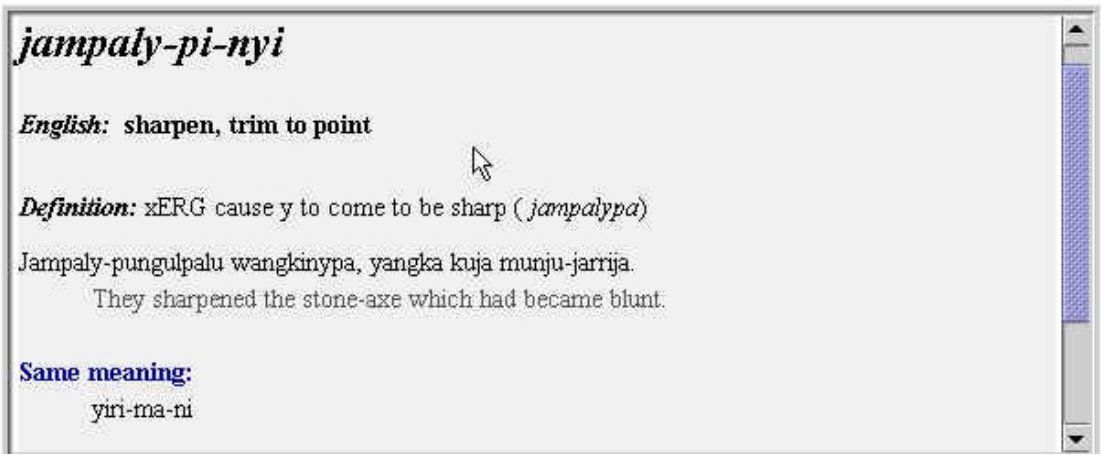


Figure 4.19a Display of the full entry format (two screen shot, before scrolling down)

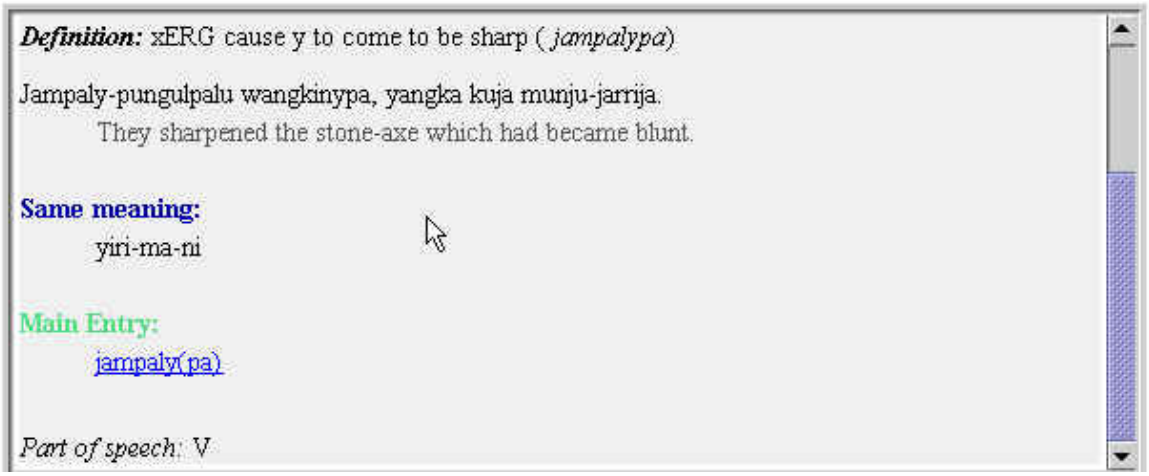


Figure 4.19b Display of the full entry format (two screen shot, after scrolling down)

Actually, we can improve this part by using HTML format instead of applet. But this may involve transforming the entry into HTML with XSL that are not available on Netscape. With HTML we can avoid the reading by parsing every input on before the files being loaded and keep it as form of HTML files, then as users change they selection, straightforwardly make out frame pointing to different HTML previously created.

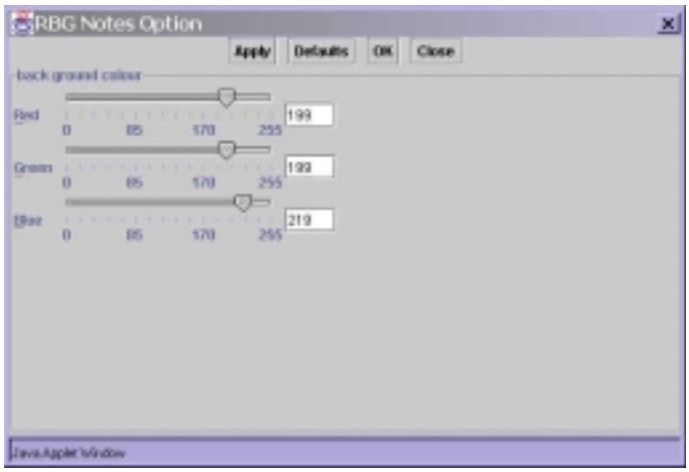
Figure 4.19c Following the hyperlink on figure 4.19b will bring you here

Notes Panel

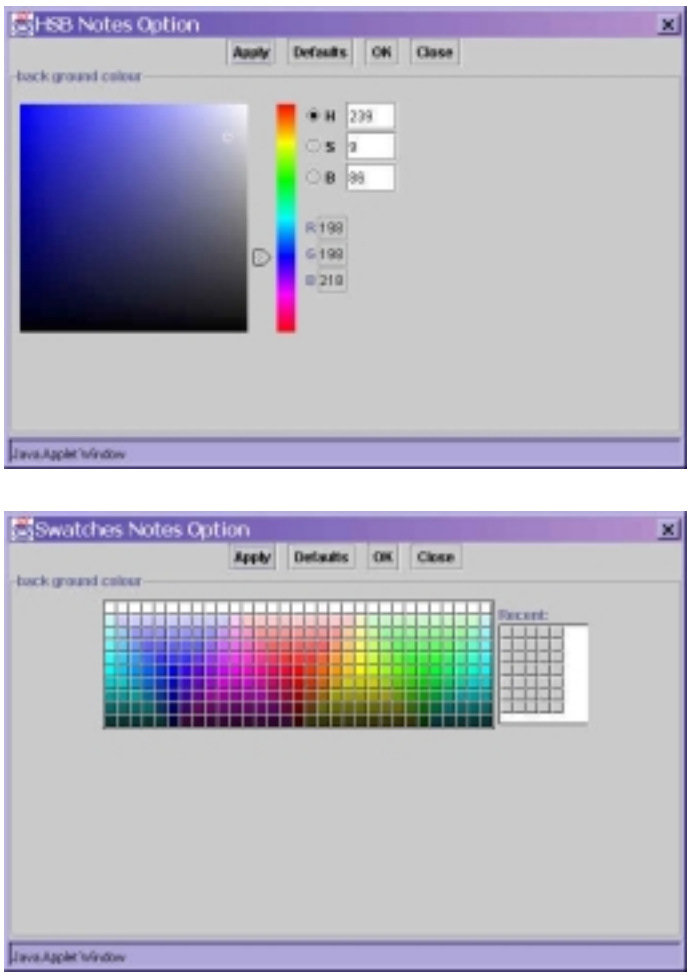
Notes panel add three more button to show notes display preferences. On the previous application, the path Option→Preferences→Notes, that are 3 another tab panel: RGB, HSB, and Swatches.

These portion remnants as an applet since it is strenuous to implement in XUL. To attach every preferences only with its field also make the controlling part easier. The idea is to reduce the number on applet so that the loading time and the parameter passing (communication among applets) lots simpler. This applet (NotesDemo) and note preferences is one applet.

Figure 4.20 Notes Panel



As mention before in the GUI design guidelines, determination of whether the module should be appear as window or dialog or another form. Here the display is dialog since this is the presentation of ongoing task.



Notice that the title of the dialog has been changed and titled as the same as the source button and panel.

This is to make the user able to trace back from where they have commenced or open the dialog (mentioned in one of the GUI guidelines above).

Figure 4.21 Various User preferences on Notes Panel

Multimedia Panel

Listen to words : Play words, Stop, Previous and Next. The applets still included java scroll bar so that if the selected headwords contains bigger images, the scrollbar (horizontal and vertical) can be used.

Why XUL scrollbar is not used? It formerly understood that XUL interface is more compact and simple, so why is not implemented in XUL instead? In this occurrence, XUL scrollbar only can acknowledge applet size. See the code below; imagine if the italics code is replaced by “<iframe…>” tag which pointing to MultimediaDemo applet on separated file.

```
<titledbox align="vertical" id="Fourth"><!--multimedia-->

<box align="horizontal">
```

```
<!--Listen to words titledbox contain of play words and stop button-->
<titledbox orient="vertical">
	<title><text value="Listen to Words"/></title>
	<button id="search-button" class="dialog" value="Play Words" />
	<button id="search-button" class="dialog" value="Stop"/>
	<spring flex="1"/>
	<button id="search-button" class="dialog" value="Previous" />
	<button id="search-button" class="dialog" value="Next"/>
</titledbox>

<!--titled button see illustration contains the picture as description of
the object-->
<titledbox orient="vertical" flex="100%">
	<title><text value="See Illustration"/></title>
	<box orient="horizontal" flex="100%">
		<box align="horizontal">
			<html:applet name = "MultimediaDemo"
			code ="MultimediaDemo.class"
			archive = "demo.jar, gnu.jar, oro.jar, jh.jar,
			parser.jar, xt_old.jar"
			width = "500" height = "200">

			<html:param name="xml_file" value="Wrl.xml"/>
			<html:param name="index_file" value="Wrl.clk"/>
			<html:param name="html_folder" value="html"/>
			<html:param name="applet_width" value="500"/>
			<html:param name="applet_height" value="200"/>
			<html:param name="applet_num" value="4"/>
			</html:applet>


		</box>
	</box>
</titledbox>
</box>
</titledbox><!--Multimedia-->
```

Then, it is known there is various sizes of images to be put on viewed. If the size of applets is not dynamic (remains the same all time, since the parameter is static as well), then XUL scroll bar cannot distinguish the discrepancy between bigger image and smaller image.

For this reason, the applet still includes java scrollbar instead on XUL scrollbar that is the applets size is static but the images size is dynamic.

Figure 4.22 Multimedia Panel

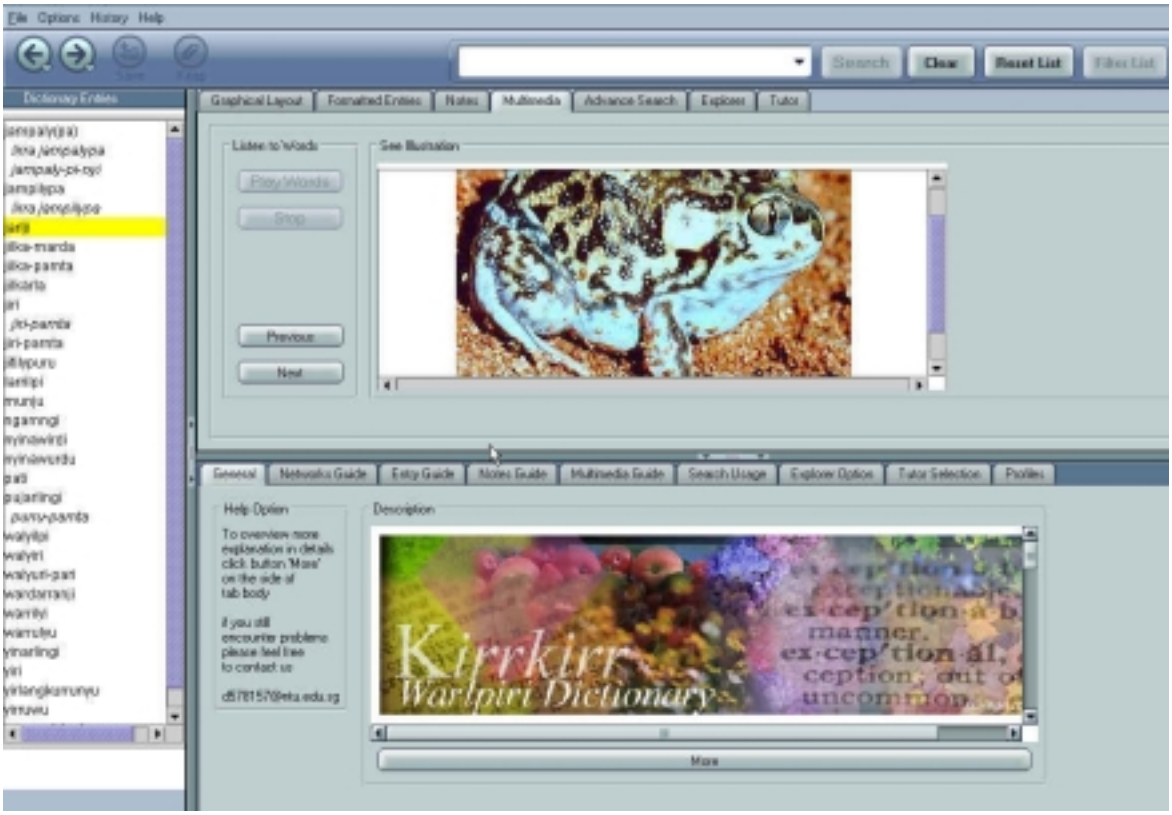


Figure 4.23 One word may have some pictures on multimedia panel, for example word "jarlji" has two representatives, one of it is shown in figure 4.22

Advance Search Panel

Since most of the contents of previous Advance Search Panel was buttons, so that most of its parts can be simplify by converting to XUL buttons. The only part that remain as java is the result table since Java brings ability to drag and drop effortlessly, moreover the java applet holds the search functionality (which will be difficult to perform using only java script). There are some problem doing the text field and start button in XUL since the JavaScript can not correctly call the java method (there are exception raised), therefore this part is implemented as a part of the applet. The progress bar has been removed since we already have one on the main window.

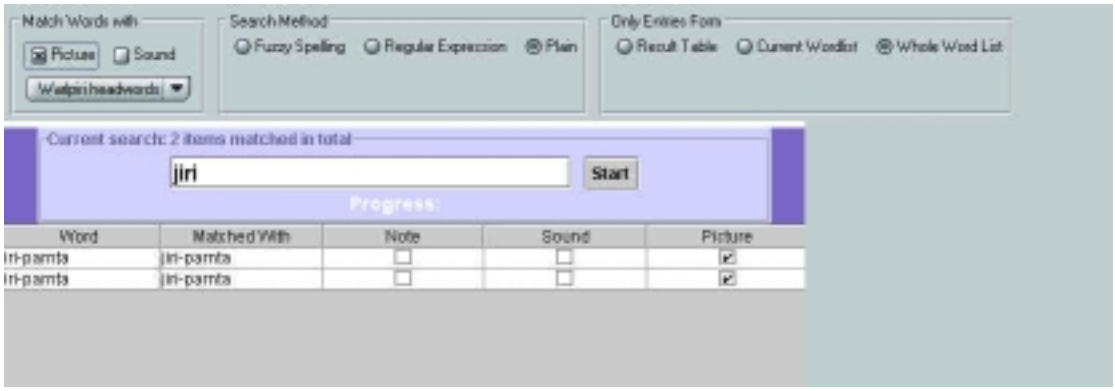


Figure 4.24a Search for word "jiri"

The above figure is the result of searching the word "jiri" for matched word in Warpliri, search method is : fuzzy spelling (word with similar pronunciation will be included), and the source are from the current word list. While figure 4.24b is finding word "jiry" which pronounced the same as jiri. With fuzzy spelling, word "jirry", "jiry", "jirri" will result on the same words collection, since those three are pronounced the same.

Regular Expression is a wide used method in searching. Yet has its own syntax for example the expession "^…I$" means "search for words with 4 character enclosed with 'I'. Therefore, it shall return : jiri, yiri, and pati. Yet Kirrkirr search is able to define some other filters, for example, search those that have picture representation, or search from the result table instead of from the entire database.

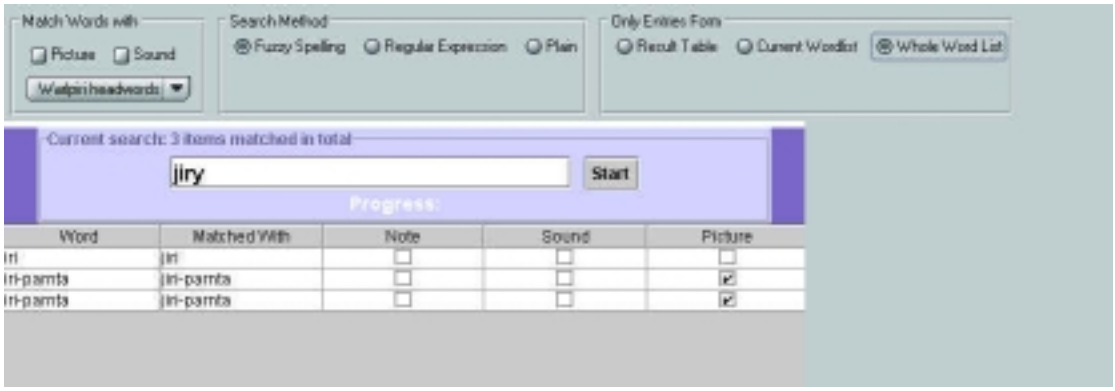


Figure 4.24b Searching for word "jiri" with fuzzy spelling

Field drag and drop are still available as it previously designed on java application. There are a lot of features of this search. For example, users may act upon search base on the current word on the list (sometimes if the sub-words are hided, the number of the current word list may be less), the current result of the search displayed on the table, or base on the entire word list.

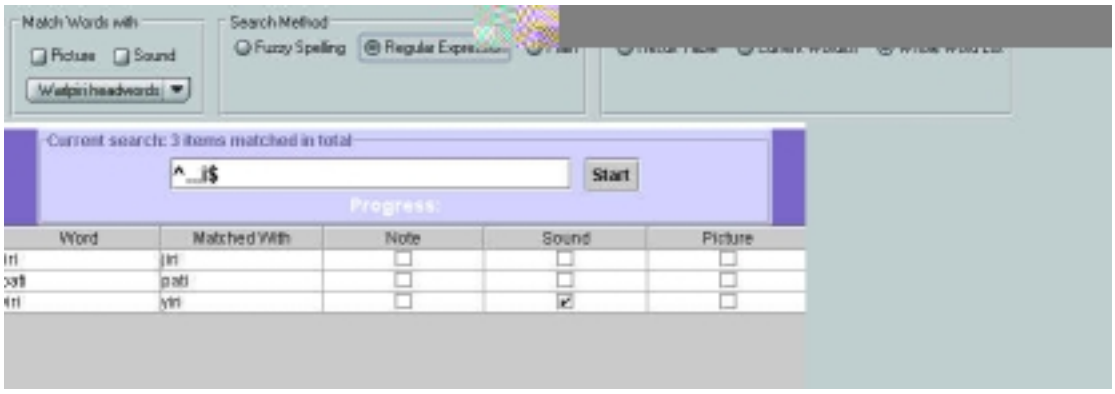


Figure 4.24c Searching for word "^…I$" with Regular Expression

User is able to specify the match requirement of input, such as the words opposite to the input, the words that are has semantic relation, the collection of words, which are synonymous to the input and many others. All of this ability is fully supported on former Java Application.

Explorer Panel

As multimedia panel, this applet also includes its scrollbar for the same reason. Those three buttons on the right side are taken from previous application. Clear is to clear the screen,

collapse is to shrink the tree while expand is to display the entire branch of the selected word (the opposite of collapse).

Figure 4.25 Explorer Panel

Tutor Panel

Tutor panel is simply a tutoring module that will read as input tutor file and display it on the applet. On the worksheet level titled box, you may see there are number of units ready to be loaded. In fact, it is better for us to implemented as drop down menu instead of buttons, but in this circumstance since we have a lot of space and not many tutorial files available, creating buttons make not much different. These units' buttons are remaining as applet since it is need to perform reading unto input file keyed in by the user (note that java script can not link up XUL action and java method which performing read, write and execute).

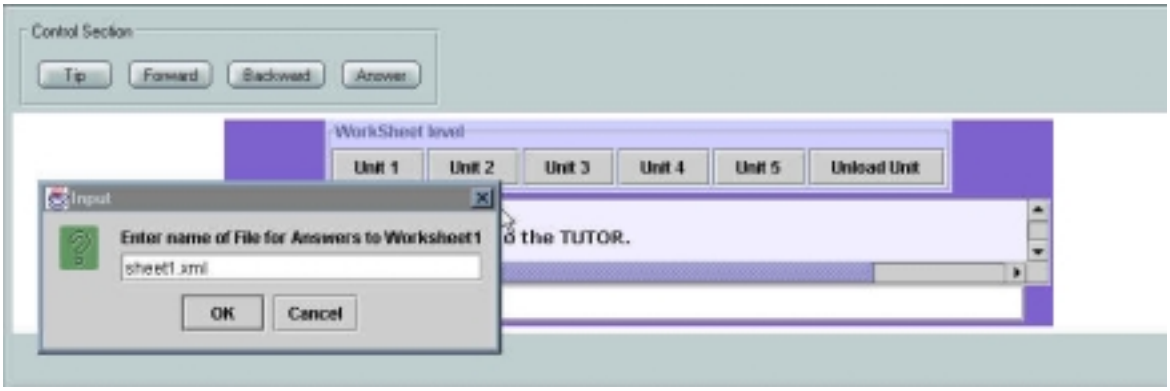


Figure 4.26 Tutor Panel loading tutorial file

The other control button are those : Tip, Backward, Forward and Answer. Backward and Forward is for user to be able to scroll up and down from the first question to another. We can assume this as Tutor history control. While Tip and Answer merely give user hint and answer.

Figure 4.27a Starting a tutorial

Figure 4.27b Browsing through questions

Chapter 5 :
Speed, User Preference and Future Expansion

The advantages of on browser application is : separation on GUI part and control part where by, in this project, XUL is in charge of the GUI part and Java is controlling the behavior of the application. This will make the maintenance and programming in the future easier. The other thing is the change in theme is possible via browser window.

Some section work better in browser since not much memory being used, such as animation on graph panel move faster and smoother in browser better than in previous Java application. But the much of time wasted on loading applets and initiating the fist execution of JavaScript on every applets.

On browser implementation, GUI has make use of browser features that make it possible to remove some of the menu or option on the application. This has created more space and stores more space for ease of future expansion. Some design such as relevant picture is added into the application to enhance the overall looks.

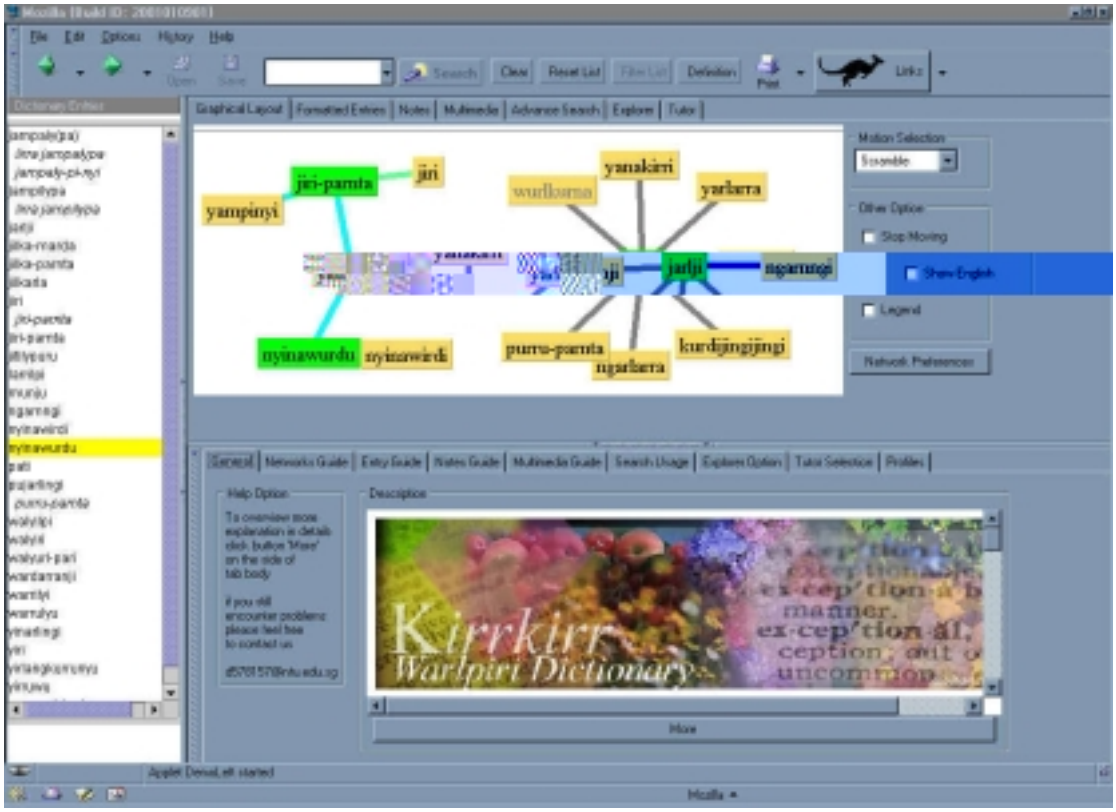


Figure 5.1 Kirrkirr on Classic Skin

The Design has certain limitation. As revealed before, in order for this program to response faster and loaded faster into the browser, ideally the dictionary should only construct over XUL and JavaScript or any other component that are suitable in browser environment. But since the dictionary consists of some complex and complicated algorithms, ended up with its unbearable to be instigated with XUL and JavaScript alone (for example: search function), therefore we still need to insert the applets components here and there.

Another problem is that, some of the applet cannot be tainted into XUL interfaces. For example, the floating boxes on Graphical Layout. So, for the same reason some of the window still includes applets inside.

Some of the XUL tag cannot be presented on certain theme. For example “alfred” theme wasn’t able to comprehend with large images pasted on the toolbar. On figure 5.2, the kangaroo logo seemed inappropriate.

Figure 5.2 Kirrkirr in "Alfred" skin. Notice that the "links" logo (kangaroo) cannot be displayed correctly

Future Expansion on GUI part is merely possible, since the widgets can be replaced and added without the needs of compilation. Moreover, besides switching theme, it is also possible to switch among languages (as long as the jar/source file are provided)

Chapter 6 :
Conclusion and Recommendation

After all the steps through out the entire project, it proves that yet extensible mark up language technology is very useful in certain extend.

- Scriptable, that is supported in changing UI without recompilation. Application development cycle is dramatically reduced.
- Cross-platform. The same sets of XUL files are portable across platforms including Unix, Mac, and Windows 32.
- Customisable as mentioned on introduction of "Configurable Chrome", all components of the chrome can be configured either statically in the chrome registry or dynamically on the fly.

But there are still a lot of works to be done to modify certain application in order to enforce it to make the full use of browser ability itself. The overall concept of interaction between interfaces (build over XUL) and application is shown below.

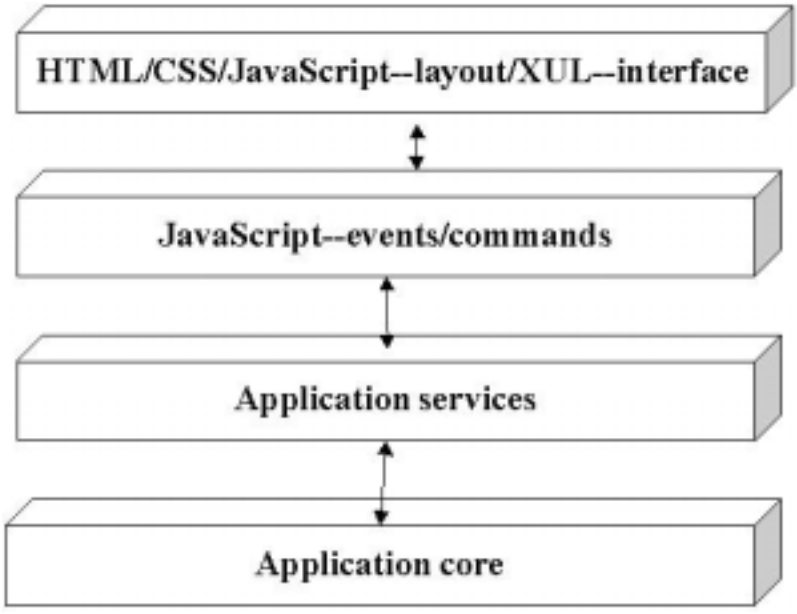


Figure 6.1 XUL and Application Core Communication

XUL is a very good start to begin an application with wide possibility of future expansion. The .css can be customised easily without affecting any other potion of code, so that user

may experience different surroundings. DTD and RDF files help overall application more organized, structured and can be traced comfortably. XUL file can be separated into numbers of independent window for the ease of debugging.

The XUL code itself can provide basic interface (widgets) and their basic behaviour, and programmers just need to know the syntax of the widgets. XUL is easier to understand than Java or C++ in term of creating interface. The drawback of using XUL is that only few browsers may be able to display it (Netscape and Mozilla family). But, note that Mozilla has several versions beside Windows OS (for example Mac and Linux) as well as Java, which is a cross platform language, make it possible for user to operate this application on various platform.

Our recommendation is to re build this application on XML, XPCom, JavaScript or any other language that can communicate well and robust with XUL. Other is to expand the language within the application so that it may represent the GUI in many languages (besides English-Warlpiri). Removing Java applet from application design will give user a lot of advantage beside faster loading time as well as no more security problem, need not to be anxious about the memory usage (as java stack dump a lot of memory space) and more flexibility. The existence of Java Applet also makes it not achievable to change the language preferences of the entire application.

This project tried to follow as close as possible to Mark up language Guidelines as well as GUI language Guidelines, but still due to the time constrains as well as trying to maximise of reusing the code, make this project doesn’t put into practice properly according to the guidelines. Yet, it is concluded that GUI design on browser application can be achieve in regard of the complexity of the application core.

References

[Arlov, L]
Arlov, L.*GUI Design for Dummies.* IDG Books: Foster City, CA, 1997.

[Bain, S]
Bain, S. and Gray, D. *Looking Good Online: The Ultimate Resource to Creating Effective Web Designs.* Ventana: Research Triangle Park, NC, 1996.

[Bergeron, Jaques]
Bergeron, Jaques.*Ant Manual*.http://jakarta.apache.org/ant/, 2000.

[Browne, D]
Browne, D. *STUDIO: Structured User-interface Design for Interaction Optimization.* Prentice Hall: New York, NY, 1994.

[Carey, J]
Carey, J. (Ed.) *Human Factors in Information Systems: The Relationship Between User Interface Design and Human Performance.* Ablex: Greenwich, CN, 1997.

[Cheng, Tao]
Cheng, Tao. *XUL Coding Style Guidelines.*
http://www.mozilla.org/projects/l10n/xul-styleguide.html, 2000.

[Cooper, A]
Cooper, A. *About Face: The Essentials of User Interface Design.* IDG Books Worldwide: Foster City, CA, 1995.

[Cover]
Cover,Robin. *The XML Cover Pages : Extensible User Interface Language.*
http://www.oasis-open.org/cover/xul.html, 1999.

[Connoly]
Dan Connoly. *XML: Principles, Tools, and Techniques*. O'Reilly and Associates, 1997.

[Deakin]
Deakin, Neil.*XUL Element and Script Reference.*
http://www.xulplanet.com/tutorials/xultu/, 1999.

[Dumbill]
Dumbill Edd.*Fooling with XUL*. XUL.com, 2000.

[Eberts, R.E.]
Eberts, R. E. *User Interface Design.* Prentice-Hall: Englewood Cliffs, NJ, 1995.

[Eich, Brendan]
Brendan Eich. *Mozilla Development Readmap.* Mozilla Organization.
http://www.mozilla.org/roadmap.html, 2000.

[Fleming, J]
Fleming, J. *Web Navigation: Designing the User Experience.* O'Reilly & Associates: Sebastopol, CA, 1998.

[Fosythe, C]
Fosythe, C., Grose, E., and Ratner, J. (Eds.) *Human Factors and Web Development.* Lawrence Earlbaum: Mahwah, NJ, 1998.

[Fowler, S]
Fowler, S. *GUI Design Handbook.* McGraw-Hill: New York, NY, 1998.

[Fowler, S.L.]
Fowler, S. L. and Stanwick, V. R. *The GUI Style Guide.* AP Professional: Boston, MA, 1995.

[Galitz, W.O.]
Galitz, W. O. *The Essential Guide to User Interface Design.* Wiley: New York, NY, 1997.

[Gulbransen, David]
Gulbransen, David, Kenrick Rawlings, and John December. *Creating Web Applets With Java*. Sams Publishing, 1996.

[Hackos, J. T.]
Hackos, J. T., and Redish, J. C. *User and Task Analysis for Interface Design.* Wiley: New York, 1998.

[Hix, D.]
Hix, D. and Hartson, H. R. *Developing User Interfaces: Ensuring Usability Through Product & Process.* Wiley: New York, NY, 1993.

[Hyatt, David]
Hyatt, David .*Writing Skinnable XUL and CSS*. Mozilla Organization. http://www.mozilla.org/xpfe/skins.html, 1999.

[Jansz, Kevin]
Jansz, Kevin, Nitin Indurkhya and Christopher D. Manning. "Kirrkirr: Interactive Visualisation And Multimedia From A Structured Warlpiri Dictionary". *Proceedings of AusWeb99, the Fifth Australian World Wide Web Conference*, pp. 302-316., 1999.

[Jansz, Kevin]
Jansz, Kevin. *Intelligent Processing, storage and visualisation of dictionary Information., 1998.*

[Jirat, Jiri]
Jirat, Jiri.*CSS2 References*. Free Software Foundation, 2000.

[Java Look and Feel Design Guidelines]
Java Look and Feel Design Guidelines. Sun Microsystems. http://java.sun.com/products/jlf/dg/higa.htm, 1999.

[Laurel, Brenda]
Laurel, Brenda, ed. *Art of Human-Computer Interface Design*. Addison-Wesley. 1990.

[Krock,Eric]
Krock, Eric. *Introduces XUL*. O'Reilly and Associates, CA, 2000.

[Mandel, T]
Mandel, T. *The Elements of User Interface Design.* Wiley: New York, NY,1997.

[Manning, Christoper]
Manning, Christoper. *Kirrkirr: Experiences with a flexible software interface to indigenous dictionaries*. http://www.ldc.upenn.edu/exploration/LSA/manning/,1999.

[Martin, A.]
Martin, A. and Eastman, D. *The User Interface Design Book for the Applications Programmer*. Wiley: Chichester, UK, 1996.

[McGrath, Sean]
McGrath, Sean. *XML by Example : A Web Master Guide*. Prentice Hall, 1998.

[Miloslav, Nic]
Miloslav, Nic.*XML Schema*.Free Software Foundation, 2000.

[Mozilla.org]
Mozilla.org. *Projects-Seamonkey*, 2000.

[Oeschger, Ian]
Oeschger, Ian. *XUL Genealogy : What Does XUL Have To Do With XML?*, 2000.

[Oeschger, Ian]
Oeschger, Ian. *XUL Programmer's Reference Manual*.
http://www.mozilla.org/xpfe/xulref/XUL_Reference.html ,2000.

[Powers, Shelly]
Powers, Shelly.*Introduction To Mozilla/Navigator 6.0*.
http://www.yasd.com/samples/xul/page12.htm, 2000.

[R cube]
R cube. *Java Applet Design*.
http://www.r-cube.co.uk/applet.html, 1995.

[Redmond-Pyle, D]
Redmond-Pyle, D. and Moore, A. *Graphical User Interface Design and Evaluation (GUIDE): A Practical Process*. Prentice Hall: London, UK, 1995.

[Robets, D]
Roberts, D., Berry, D., Isensee, S. and Mullaly, J. *Designing for the User with OVID: Bridging User Interface Design and Software Engineering*. Macmillan Technical Publishing: Indianapolis, IN, 1998.

[Shelly]
Shelly.*Digital Play Dough : Designing application with XUL*. Web Technique, Boston, 1999.

[Shneiderman, Ben]
Shneiderman, Ben.*Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 3d ed. Addison-Wesley, 1997.

[Spencer, Paul]
Spencer, Paul. *XML Design and Implementation*. Wrox, 1999.

[St. Laurent, Simon]
St. Laurent, Simon. *XML Elements of Style*. New York: McGraw-Hill, 2000.

[Trudell, Peter]
Trudell Peter. *XPToolkit Project.*
http://www.mozilla.org/xpfe/, 1999.

[WaterSon, Chris]
Waterson, Chris. *XUL Template Reference.*
http://www.mozilla.org/rdf/doc/xul-template-reference.html, 2000.

[Wood L.E.]
Wood, L. E. (Ed.) *User Interface Design: Bridging the Gap from User Requirements to Design.* CRC Press: Boca Raton, FL, 1998.

[Yong, Chua Way, Evan]
Yong, Chua Way, Evan. *Tools for Inforrmation Modelling*.Nanyang Technological University, School of Applied Science., 1999.