Digital Business Ecosystem

Contract n° 507953

Workpackage 20: User Interface

Deliverable D20.6: User Interface Specification

Information Society Technologies

Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: 20.6

Due dates: 06/2006

Delivery Date: 01/2007

Short Description: To put the existing GUI frameworks into the context of the DBE system by identifying the requirements of *User Oriented Services*, that is, services developed for consumption by human users, and by finding suitable GUI Frameworks which meet these requirements.

Author: Sun Microsystems

Partners contributed: Intel, TCD, WIT

Made available to: Public

Versioning

Version	Date	Author, Organisation
0.1	14th Oct, 2004	Juanjo Aparicio, Sun Microsystems
0.2	9th Dec, 2004	Juanjo Aparicio, Sun Microsystems Paul Malone, Waterford Institute of Technology Joe Butler, Intel Ireland
0.3	10th Dec, 2004	Juanjo Aparicio, Sun Microsystems Paul Malone, Waterford Institute of Technology Joe Butler, Intel Ireland Dominik Dahlem, Trinity College Dublin
0.4	30th Nov, 2004	Juanjo Aparicio, Sun Microsystems Jordi Sánchez, Sun Microsystems Paul Malone, Waterford Institute of Technology Joe Butler, Intel Ireland Dominik Dahlem, Trinity College Dublin
0.5	26th Apr, 2005	Juanjo Aparicio, Sun Microsystems Jordi Sánchez, Sun Microsystems Rubén González, Sun Microsystems
0.6	31th May, 2005	John M Kennedy, Intel Rubén González, Sun Microsystems
0.7	18th Nov, 2005	Juanjo Aparicio, Sun Microsystems
1.0	10th Jul, 2006	Juanjo Aparicio, Sun Microsystems John M Kennedy, Intel

Quality check:

1st Internal Reviewer : Kennedy, John M - Intel

2nd Internal Reviewer:

Table of Contents

1.Introduction

The word “interface” is defined as the point, area, or surface along which two substances or other qualitatively different things meet. In computer science, the interface is a specification of those properties of a software component that other components may rely upon. In a more specific sense, a user interface is the functional and sensorial attributes of a system (appliance, software, vehicle, etc.) that are relevant to its operation by users.

Put in other words, in order to allow human users to consume computer-based services, a user interface is needed. In the first era of computation, there was little or no interface. In the 1945-1968 period the dominant type was the batch interface. The interaction was limited at providing input before the computation took place, and retrieving the results when the computation finished. In the 1969-1983 period, user interfaces allowed a greater degree of interactivity between the program and the user. The dominant type were the command-line interfaces, in which the user typed interactively a series of text-based commands.

In 1984, with the appearance of the Apple Macintosh, graphical user interfaces (GUI) helped introduce the personal computer onto the desktop of non-technical people. However, graphical user interfaces date from the 60s[1] and are based on earlier attempts of investigating how humans learn. With the appearance of personal computers and their widespread adoption by the consumer, GUIs have now become the norm for computer/user interaction, ranging from PC desktops to workstations to mobile phones.

In order to do easier the GUI development, GUI Frameworks have appeared. GUI Frameworks are a set of classes, libraries and other resources that can be used to implement GUI applications in a guided way.

This document attempts to put the existing GUI frameworks into the context of the DBE system by identifying the requirements of *User Oriented Services*, that is, services oriented at their consumption by human users, to find a suitable GUI Framework which meets these requirements.

1 http://en.wikipedia.org/wiki/History_of_the_graphical_user_interface

2.GUIs

Design and implementation of GUIs is a time and resource consuming activity, with a small return on investment. The main reason why this is so is because of GUI complexity. Even a simple-looking GUI is composed of many small components: windows, buttons, text fields, menus. The GUI computer model has to represent all these components, plus the relationships between them. Usually this is accomplished by writing programs in one of the available languages, with the aid of some toolkit and supporting libraries. But the level of abstraction of these toolkits is often too fine grained, and their use requires a great deal of boilerplate code.

Coarser grained UI toolkits would allow the programmer to concentrate on the business logic and divert time and thought from the UI design and implementation. This usually comes with an associated cost in UI performance. One size doesn't fit all, and the UI abstractions that fit one UI style or application are often not suitable for other styles or applications. For example, the graphical elements available to HTML authors allow them to represent data in an efficient way, but allow little interaction between the user and the application.

Furthermore, UI design and implementation is a multidisciplinary activity: it requires a designer with experience of ergonomics and graphical design in order that the UI be intuitive and easy to use, and to look appealing; and it requires a programmer to implement the required functionality in an efficient way, a process that must be driven by the requirements of the application the UI is designed for.

But good programmers often make bad designers, and vice-versa. A plausible solution to this problem is to decouple the UI design from the UI interactions with the application's computational model. This has been a design pattern since the days of SmallTalk 80: the MVC[2] pattern (short for Model-View-Controller). The intent of the MVC pattern is to facilitate reuse of GUI modules as well as separating the three main aspects of a GUI application in order to allow easier maintenance with minimum side effects of modifications.

By adding an additional layer of abstraction the UI execution environment is decoupled from the specific computing platform. For example, consider how the development of the UNIX operating system was accelerated by the development of the C programming language, which served initially as a macro language over heavily repeated constructs. C served as a means to reduce the amount of boilerplate code to be explicitly typed in, and also as an abstraction over the machine code the OS was being developed for. Ultimately, with the promotion of C to an ANSI standard, a program developed in the C programming language could be more easily ported to platforms other than the original supporting platform, that is, where the program was initially developed and expected to run on.

The same notion of a simplified syntax expressing many things has been done with regard to UIs in the field of the world wide web, where a few constructs suffice to provide a good-looking interface to an application. Documents in the world wide web are usually defined in HyperText Markup Language (HTML)[3]. A web page looks essentially the same (or at least it should) on different client machines, with different operating systems and web browsers. Most notably, the author of the web page is never concerned with the details of the machine where the page will be viewed. This is possible due to the additional level of abstraction: the web page describes in a declarative language the elements of the page. The actual rendering of the page takes place on the client side machine, by the virtue of a client program called the web browser. This client program provides the coupling from the abstraction layer that the web page is to the actual implementation details of the client machine architecture. HTML allows the definition of forms for data input that are submitted to a remote server.

However, HTML forms are mostly constrained to a client-server model, in which all the processing is performed by the server side. In order to free the server side of having to compute everything the solution is to implement rich clients. In fact, web browsers are rich clients themselves, only

2 http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

3 http://www.w3.org/MarkUp/

constrained to understand an abstraction layer mainly directed towards producing visual output, with very limited user interaction, as defined by HTML.

Graphical user interfaces can be classified into two main groups: imperative and declarative.

2.1.Imperative user interfaces

Imperative user interfaces are defined in an imperative programming language such as C, Java, Python, etc. They consist of one or more graphics libraries, which interact with the underlying platform that will ultimately display the graphics, and the graphics API made available to the language.

Imperative languages, such as C++ or Java, compile the structure and content of the user interface into a binary encoding that is executed in a runtime environment on the user platform. In contrast, declarative languages, such as XHTML or WML, describe the structure and content to a renderer (XHTML to a Web browser, for example, WML to a WAP Browser) that progressively interprets the markup code and renders the user interface accordingly.

Imperative languages require the programmer to explicitly specify how to perform each task; declarative languages require the programmer to only specify what tasks to perform.

Whilst imperative user interfaces offer significant advantages in areas such as flexibility, control and scalability, clear disadvantages are also associated with them. In particular, imperative user interfaces can only be invoked on platforms on which the imperative language (and relavent libraries) are supported.

Most user interfaces are written in an imperative language with a graphical toolkit (e.g., C++/MFC, C/Motif, Java/JFC). These toolkits enable programmers to quickly create complex user interfaces, without having to deal with the intricate details of implementing graphics using the underlying operating system.

Some languages, such as Java, introduce a new layer of abstraction, such as Java/JFC, permitting the development of interfaces whose appearance is identical irrespective of the operating system on which it is running.

2.2.Declarative user interfaces

Declarative user interfaces describe the structure and content of the UI. The description is used by a renderer that progressively interprets the markup code and renders the user interface accordingly.

A reason not to have UIs implemented in imperative languages is that by doing so often means that the platform of choice of the developer is enforced onto the customers of the UI. For example, a programmer may provide a UI application implemented in Visual Basic. By doing so, any customer not running a Windows operating system on a compatible machine can not execute the application

The more usual declarative user interfaces use markup languages (often XML).

There are several well-known XML user interface markup languages: XHTML (W3C recommendation), UIML (developed by OASIS), XFORMS (W3C recommendation), XUL (developed by Mozilla), Thinlet (Open Source), MXML (developed by Macromedia), GladeXML (developed by GNOME) and XAML (developed by Microsoft).

3.GUI Frameworks for the DBE

There exist several GUI frameworks at different levels of standardisation which provide different levels of abstraction.

In the case of the DBE, where widespread adoption is a primary goal, standardisation, as provided by standards bodies, is not a guarantee of ease of deployment. Rather than simply choose a GUI framework standard, a standard must be selected for which implementations are available. For example, SVG has been a W3C recommendation since January 2003. However, not may browsers support SVG graphics natively. PNG is also a W3C recommendation, this time since October 1996. But the most widely used web browser[4] is notorious for its flawed support of PNG image files[5].

In order to better evaluate potential GUI frameworks for the DBE, a series of criteria has been developed:

- Reliability: Probability that the GUI framework functions without failure. The probability increases in direct proportion with the maturity level.
- Continuity: Probability that the GUI framework will be actively maintained in the future. The continuity may be promoted mainly by open source communities when the GUI framework is open source, or by enterprises when the GUI framework is not open source.
- Portability: Be easily redistributable, in terms of redistribution policies, installed size and OS/library dependencies. Rationale: the DBE is aimed at minimizing the digital divide. Therefore the user base that benefits from the overall project must be maximised among all existing SMEs in Europe. The environments in which the DBE is to be deployed are not known in advance, and can not be constrained arbitrarily without the risk of ruling out of the DBE those SMEs that would benefit the most from it. Therefore, portability is a key issue in the overall architecture, and in the GUI arena in particular.
- Versatility: Be powerful enough to allow extensibility to build interactive applications, as well as simple client/server forms. This criterion is somewhat lax. In this document UI frameworks will be evaluated for their power to be used for different tasks. However, this does not mean that the most versatile framework will be the most fit for the task of providing DBE-wide UIs, because more versatile frameworks usually come with an attached increase in memory footprint and demanding CPU capacities. A compromise is needed between the most versatile framework and one that is barely able to perform the tasks the DBE needs the UI for.
- Freedom: Not suffer from vendor lock-in. Vendor lock-in is seen as a threat, because it effectively ties the consumer with a provider (in this case, the UI framework provider). Should new requirements arise on the part of the consumer SMEs, the vendor may decide not to follow the new requirements. There must be a way for the consumer to switch to another vendor that provides the same solution, or to ask someone else to make the appropriate changes, if they are needed.
- Internationalisation: Easy support for internationalisation. The European Union spans 25 countries (as of 2004[6]) and 20 official languages[7]. In order to make all users of the DBE equals when it comes to DBE usage, the DBE itself should be internationalised and support localisation of the application to specific languages.
- Pervasiveness: Easy access to the User Oriented Service created with the GUI Framework. The final user should not need to install any software every time they access a User Oriented Service. Nevertheless, the final user may need to install some generic software framework to access all the User Interface Services in the same way; examples of that software are browsers, runtime libraries,

4 http://www.pcworld.com/news/article/0,aid,116848,00.asp

5 http://www.libpng.org/pub/png/pngapbr.html#msie-win-unix

6 http://europa.eu.int/abc/index_en.htm

7 http://europa.eu.int/comm/education/policies/lang/languages/index_en.html

etc.

- Learning curve: SME Providers must be able to learn to use the GUI Framework quickly and with low cost. Factors that contribute to this criteria are:
    - Inherent Complexity of the GUI Framework: How easy it is to understand the main abstractions of the GUI model, how easy it is to make these abstractions interact. Note that this criteria is highly impacted by the mainstream programmer culture. For example, the MVC pattern was something novel in the early eighties, but is a fairly well-known pattern nowadays.
    - Learning resources: Have good and diverse resources to learn how to use the GUI Framework. This resources can be on-line books, on-line tutorials, printed media, etc...
    - Helping tools: Existence of tools that help developers to use the GUI Framework. Normally the graphical front-ends are more valuable than text front-ends. The most valuable tools are the IDEs (Integrated Development Environment), that normally have source code editors, pre visualization, a compiler, build-automation tools and a debugger.

In the following chapters different GUI Frameworks have been evaluated with the previous criteria. The GUI Frameworks evaluated are:

- XForms
- XUL
- Thinlet
- SWING/JFC
- SWT/JFACE

3.1.XForms

XForms is a W3C recommendation since 14th October, 2003. It provides a layer of separation between the form presentation and the form model.

The W3C specifies (http://www.w3.org/MarkUp/Forms/) the key goals of XForms:

- Support for handheld, television, and desktop browsers, plus printers and scanners
- Richer user interface to meet the needs of business, consumer and device control applications
- Decoupled data, logic and presentation
- Improved internationalisation
- Support for structured form data
- Advanced form logic
- Multiple forms per page, and pages per form
- Suspend and Resume support
- Seamless integration with other XML tag sets

In spite of all this, XForms is still aimed at providing rich interfaces to client-server applications. That is, the processing is still done on the server side. The DBE needs user interfaces for client-server applications, but also for rich client applications, where as much processing as possible is being carried out by the client side.

There are no widely used web browsers supporting XForms. There is an implementation of an XForms client in Java, denominated XSmiles[8], but currently it is at beta stage.

3.1.1.Evaluation

- Reliability: The W3C has listed three fully compliant implementations[9] as of July 8th, 2003, as well as another list of known implementations[10] as of September 1st, 2004, but at the time of writing of this document there are no mature client implementations.
- Continuity: The continuity depends on the W3C. At the time of writing (April 2005) the W3C seems committed to maintaining the specification, and has already issued an XForms 1.1 Working Draft.
- Portability: Being a declarative language expressed in XML, the portability of XForms is assured.
- Versatility: XForms provides a conceptual separation between the view of the form and the data model behind it. Apart from that, the framework is still aimed at providing input/output forms for client/server applications. Little computation is performed on the client side[11]. Interactions that involve complex computations are constrained by the server response time plus two round trip times. A possible performance optimisation could be to have the server side as a client side application which can receive XForms submissions. In that case the front-end of the application is still the same as for remote applications, but the "server side" is local to the client network. Therefore, the round trip times are diminished to local network connection times.
- Freedom: As XForms is a W3C standard, any vendor can implement it.

8 http://www.xsmiles.org/

9 http://www.w3.org/MarkUp/Forms/Test/ImplementationReport.html

10 http://www.w3.org/MarkUp/Forms/#implementations

11 http://www.idealliance.org/papers/dx_xml03/papers/03-05-04/03-05-04.html#s2.2.3

- Internationalisation: The XForms standard has explicit support for internationalised applications[12].
- Pervasiveness: An XForms-enabled browser is needed on the client side. If in the future this technology is integrated in widely used web browsers, then the pervasiveness will be assured.
- Learning curve: XForms have a medium complexity. The standard can be accessed on-line in the W3C site. An on-line tutorial named "XForms for HTML Authors"[13] also exists on the W3C site. Some books have been published. There are no free or commercial IDEs available.

12 http://www.idealliance.org/papers/dx_xml03/papers/03-05-04/03-05-04.html#s2.1.5

13 http://www.w3.org/MarkUp/Forms/2003/xforms-for-html-authors

3.1.2.Screenshots

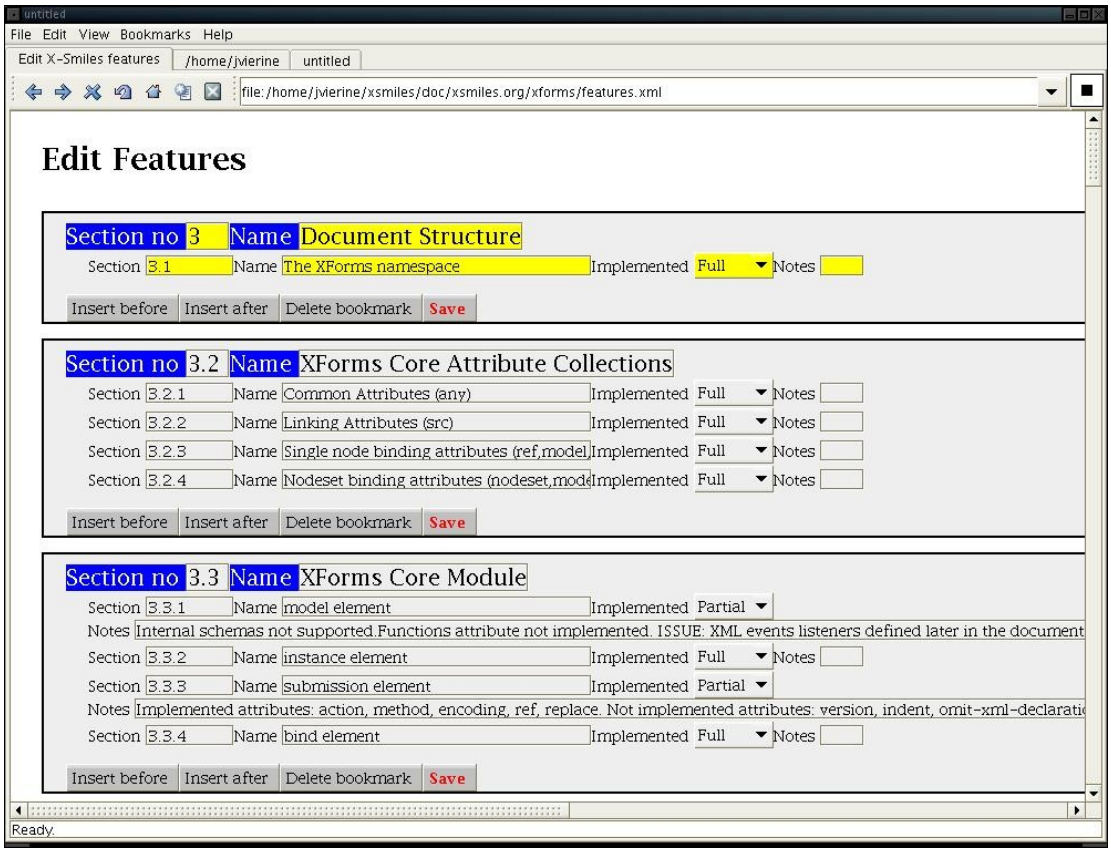


Figure 1: The X-SmilesXForms browser, editing form data...

Figure 2: ... on a PDA

Figure 3: ...and on a mobile phone

3.2.XUL

XUL[14] is an acroynm for XML User interface Language. It is an XML based framework for the declarative description of GUIs. Fully-fledged applications can be constructed with the aid of XUL. XUL plays a role in the development of Mozilla applications similar to the role of the C programming language in the development of the UNIX operating system[15]. For an example of the power of XUL, see any of the Mozilla applications[16].

XUL, however, is not a standard[17], but a product of mozilla.org, built for their own purposes. The FireFox web browser and the ThunderBird mail user agent are both XUL applications, demonstrating what XUL is capable of. The XUL Alliance[18] project contains a list of XUL implementations for several platforms.

Another problem of XUL is that the platform needed to execute XUL applications requires several megabytes of software to be installed in each client. Mozilla is in the process of releasing their Gecko Runtime Environment to allow the easy installation and deployment of XUL-based applications[19]. In the meantime, if you want to use XUL, you either have to have one of the mozilla.org applications installed, or select one of the open source implementations.

3.2.1.Evaluation

- Reliability: The Mozilla suite (browser, email client, chat application, ...) is a robust and quality product implemented with XUL. This fact suggests that XUL is reliable, but the lack of existent applications apart from the Mozilla suite may be interpreted as a warning with respect to reliability.
- Continuity: The continuity depends on two facts, the existence of the Mozilla Foundation[20] and by the existence of the the Mozilla Community[21].
- Portability: The originator of the XUL specification is the Mozilla project. The main execution environment for XUL-specified GUIs is the Gecko Runtime Environment (GRE), created by Mozilla. The GRE is not yet available as a standalone download, but is the core of the Mozilla suite, the Firefox web browser and the Thunderbird mail/news reader. The Mozilla suite is available for Windows, Linux and Mac OS X.
- Versatility: The Mozilla suite is proof of the versatility of the language regarding the definition of user interfaces. The Mozilla suite is composed of the XUL engine plus the supporting libraries for HTML rendering, JavaScript engine, communications with email backends, etc.
- Freedom: The Gecko Runtime Environment is a product of Mozilla.org. It is the main implementation that supports the full XUL. The Mozilla suite, the Firefox web browser and the

14 http://www.mozilla.org/projects/xul/

15 http://cm.bell-labs.com/cm/cs/who/dmr/chist.html

16 http://www.mozilla.org

17 http://www.xml.com/pub/a/2001/09/05/xforms.html?page=last

18 http://xul.sourceforge.net

19 http://www.mozilla.org/projects/embedding/GRE.html#mozTocId334331

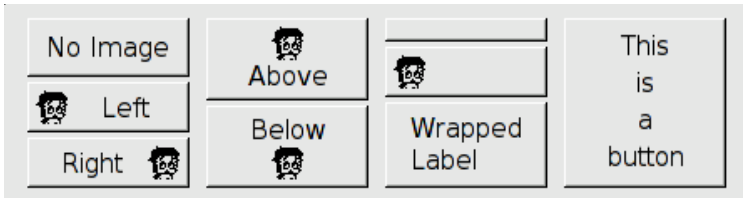
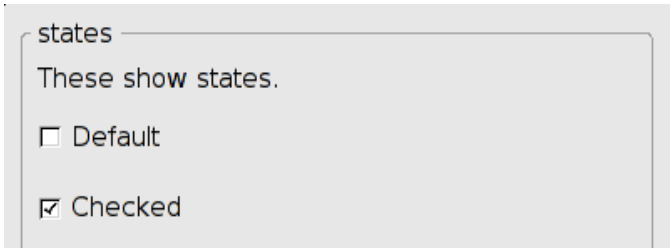
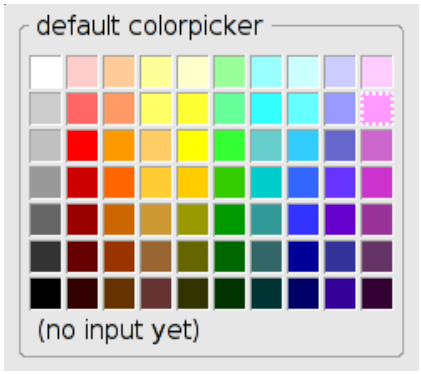
20 http://www.mozilla.org/foundation/

21 http://www.mozilla.org/community/

Thunderbird mail/news reader are released under the Mozilla Public License[22].

- Internationalisation: Mozilla XUL supports internationalisation of applications through the use of XML entities[23].
- Pervasiveness: Using the Mozilla Firefox browser it is possible to provide access to the User Oriented Services without the need for local deployments of additional software.
- Learning curve: XUL has a medium complexity. In the "XML User Interface Language (XUL)"[24] page, there are references to the following on-line resources: XUL 1.0 specification, XUL examples, RE Project (XUL Runtime Environment), XUL Tutorial and XUL Programmer's Reference Manual. There are few commercial books. There are no mature IDEs.

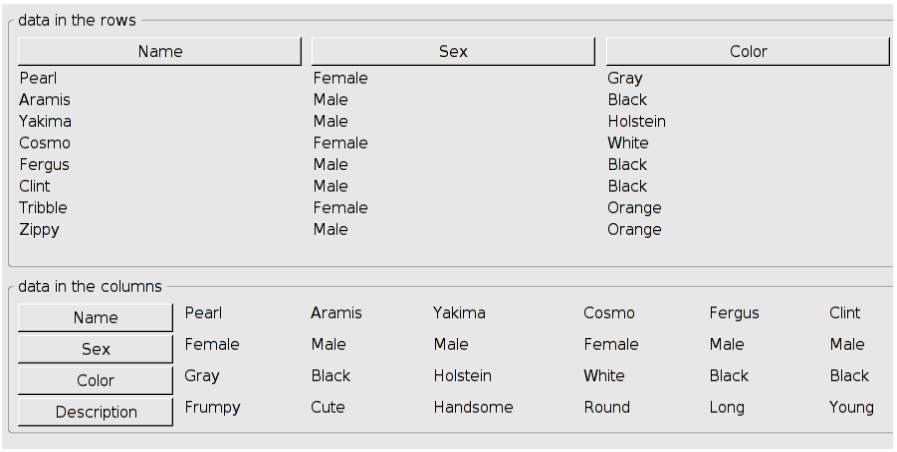
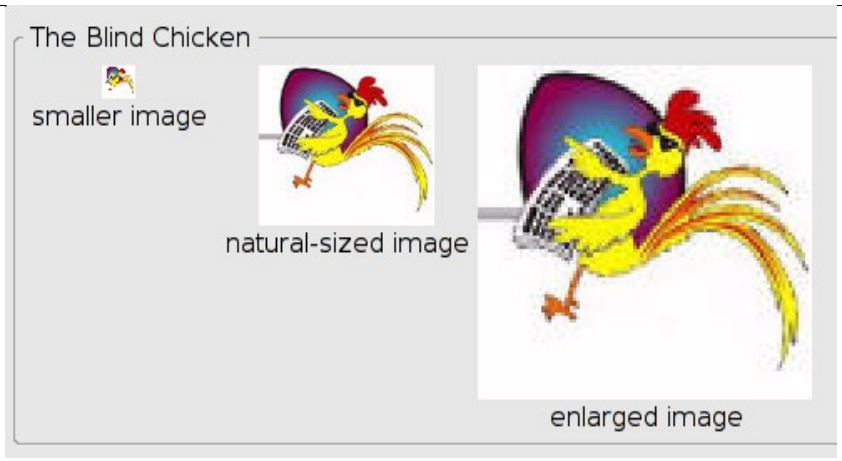
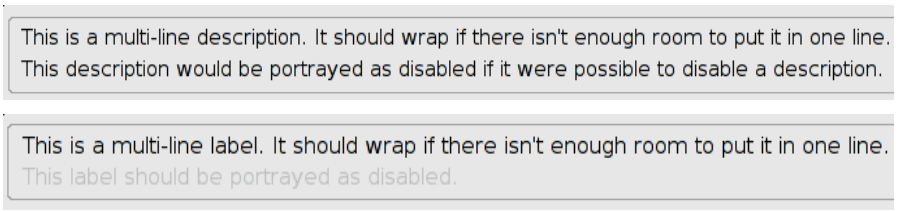
3.2.2.Widgets

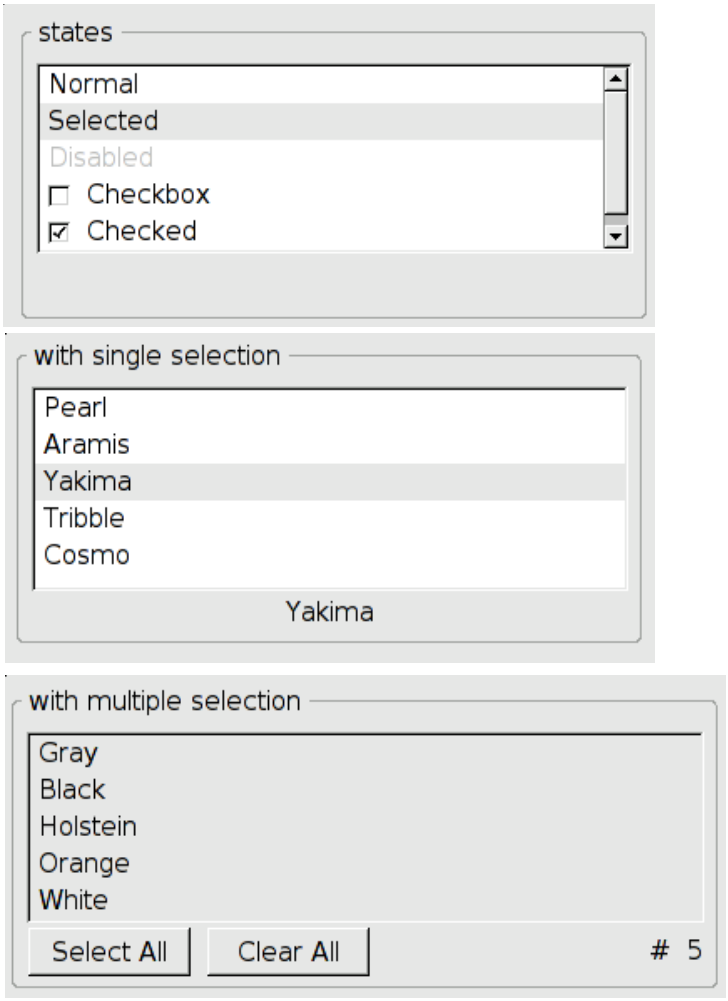
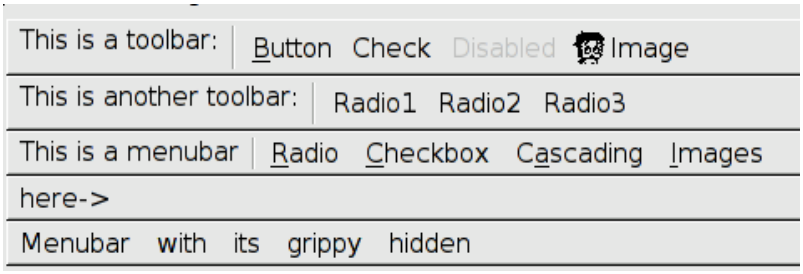
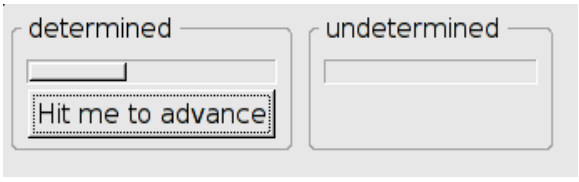
Buttons	
Checkboxes	
Colorpicker	

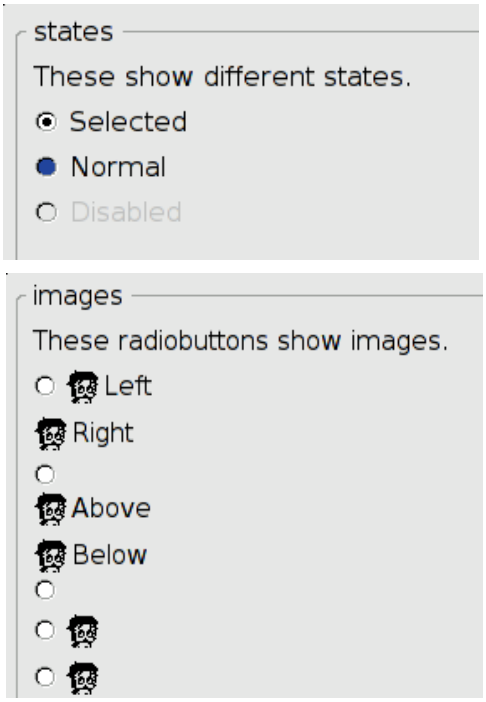
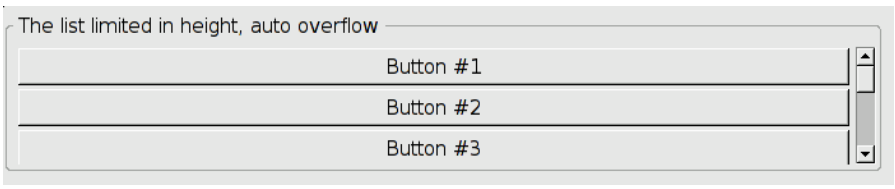
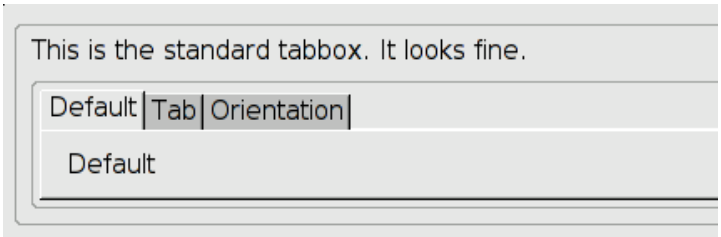
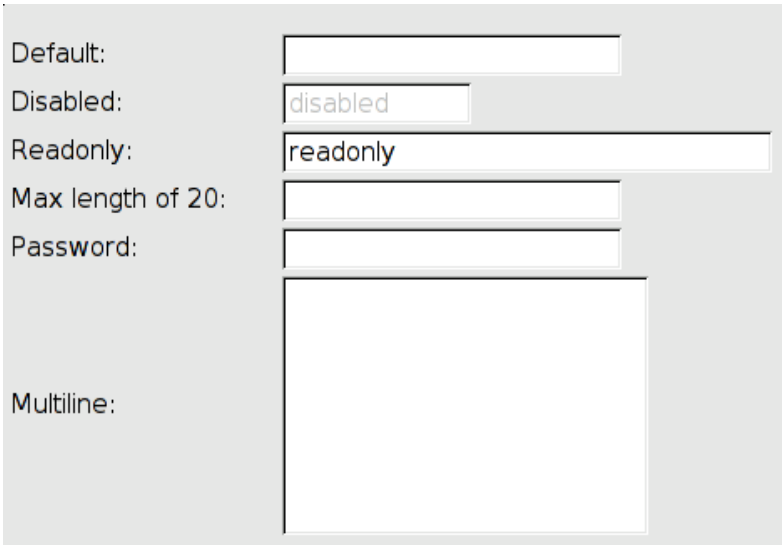
22 http://www.mozilla.org/MPL/

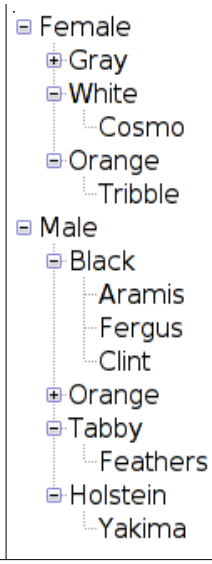
23 http://xulplanet.com/tutorials/xultu/locale.html

24 http://www.mozilla.org/projects/xul/

Grids	
Images	
Labels	

Lists	
Toolbars and Menubars	
Progressmeters	

Radiobuttons	
Scrolling	
Tabs	
Textboxes	

Trees	 Female Gray White Cosmo Orange Tribble Male Black Aramis Fergus Clint Orange Tabby Feathers Holstein Yakima

3.3.Thinlet

XUL is not consistently implemented across vendors, and several subsets exist. One of the most attractive of these is Thinlet[25]. Thinlet is an implementation of XUL in 38 KBytes of Java classes.

The main characteristics of Thinlet are:

- Small footprint.
- Inherits the good traits of XUL, that is, the UI is expressed declaratively in an XML document.
- The implementation of the actions of the UI is done through Java classes, allowing applications to be run on any platform to which Java has been ported.
- Thinlet runs on Java Runtime Environment 1.1 or above.
- Thinlet applications can be embedded as applets.
- Thinlet is licensed under the Lesser GNU Public License[26].
- The XUL documents Thinlet understands are not mozilla.org XUL documents. This implies that Thinlet cannot be used to execute mozilla.org XUL applications.

3.3.1.Evaluation

- Reliability: Thinlet is an open source project developed and maintained by only one person, and it is not widely used. Despite this, it seems to have reached a certain degree of maturity, having appeared in 2002 yet only having 6 opened bugs to date.
- Continuity: The fact that Thinlet has been developed and is maintained by only one person, and the fact that it is not in widespread use, suggests a low probability of it surviving into the future.
- Portability: Being a Java application the portability is guaranteed onto all those platforms for which a Java platform exists. Thinlet requires a modest Java Runtime Environment version 1.1. Swing is not required. Personal Java[27] can make use of it. J2ME Personal Profile[28] can use it as well. Although it can be argued that Java is a platform in itself and, therefore, choosing Java as the programming platform equals ruling out other alternatives and risking a vendor lock-in in the long run, the facts are that any vendor can provide a Java implementation, and that the language and the platform themselves are governed by a community process, although it is true that Sun Microsystems has the ultimate control over it.
- Versatility: Though much more limited in power than fully-fledged XUL implementations like the Gecko Runtime Environment, Thinlet still provides GUIs rich enough for most applications. The main website offers samples[29] in the form of applets. The code to develop a Thinlet applet is the same as for a Thinlet application, only the packaging details differ. Behavioural extensions are implemented in the Java programming language.
- Freedom: Thinlet is a product licensed under the LGPL. This means that it can be embedded into any application without the need to change the whole application license, which may be closed source, except for the Thinlet engine itself.

25 http://xul.sourceforge.net/thinlet.html

26 http://www.gnu.org/licenses/lgpl.html

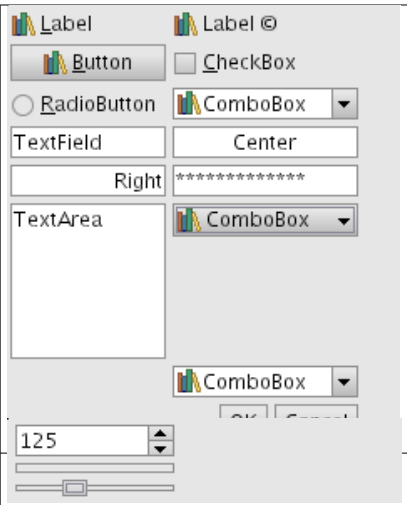
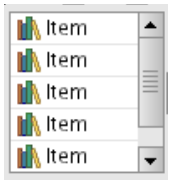
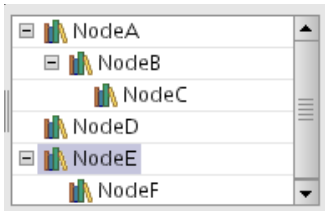
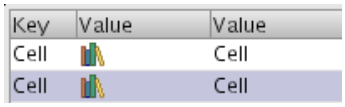
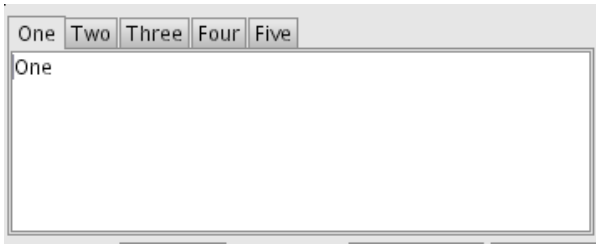
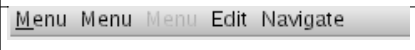
27 http://java.sun.com/products/personaljava/index.jsp

28 http://java.sun.com/products/personalprofile/index.jsp

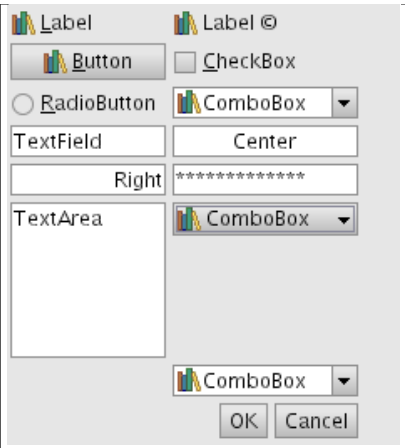
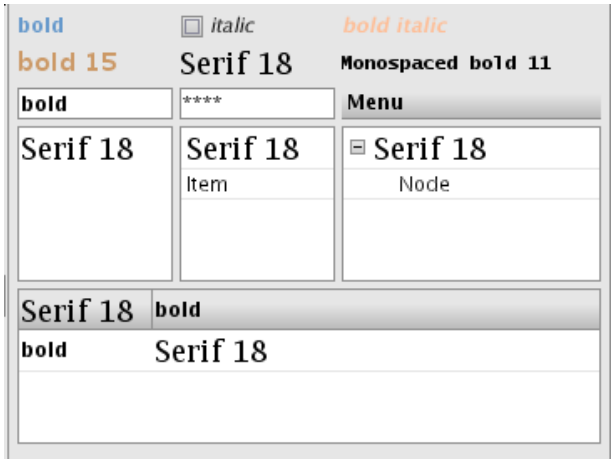
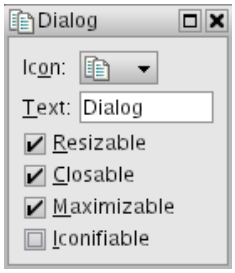
29 http://thinlet.sourceforge.net/demo.html

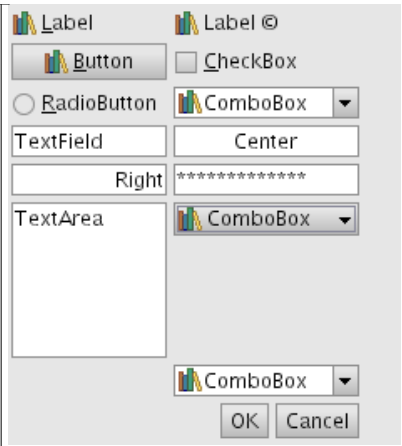
- Internationalisation: Thinlet supports internationalised applications by means of resource files containing appropriate messages.
- Pervasiveness: Easy access to User Oriented Services could be provided by using Java Applets, without the need for local deployments of additional software.
- Learning curve: Thinlet has a low complexity. In the project home page there are some examples and a small reference manual. There are no commercial books available, and there are no free (free as in ``free beer'') mature IDEs. There is a free visual GUI design editor for Thinlets, named ThinG[30], but it is in beta stage.

3.3.2.Widgets

Labels, Buttons, TerxFields, TextArea, ComboBox	
Spinner, Progress	
List	
Tree	
Table	
TabbedPane	
Menu	

30 http://thing.sourceforge.net/

Labels, Buttons, TerxFields, TextArea, ComboBox	
Text	
ToolBar	
Dialog	

Labels, Buttons, TerxFields, TextArea, ComboBox	

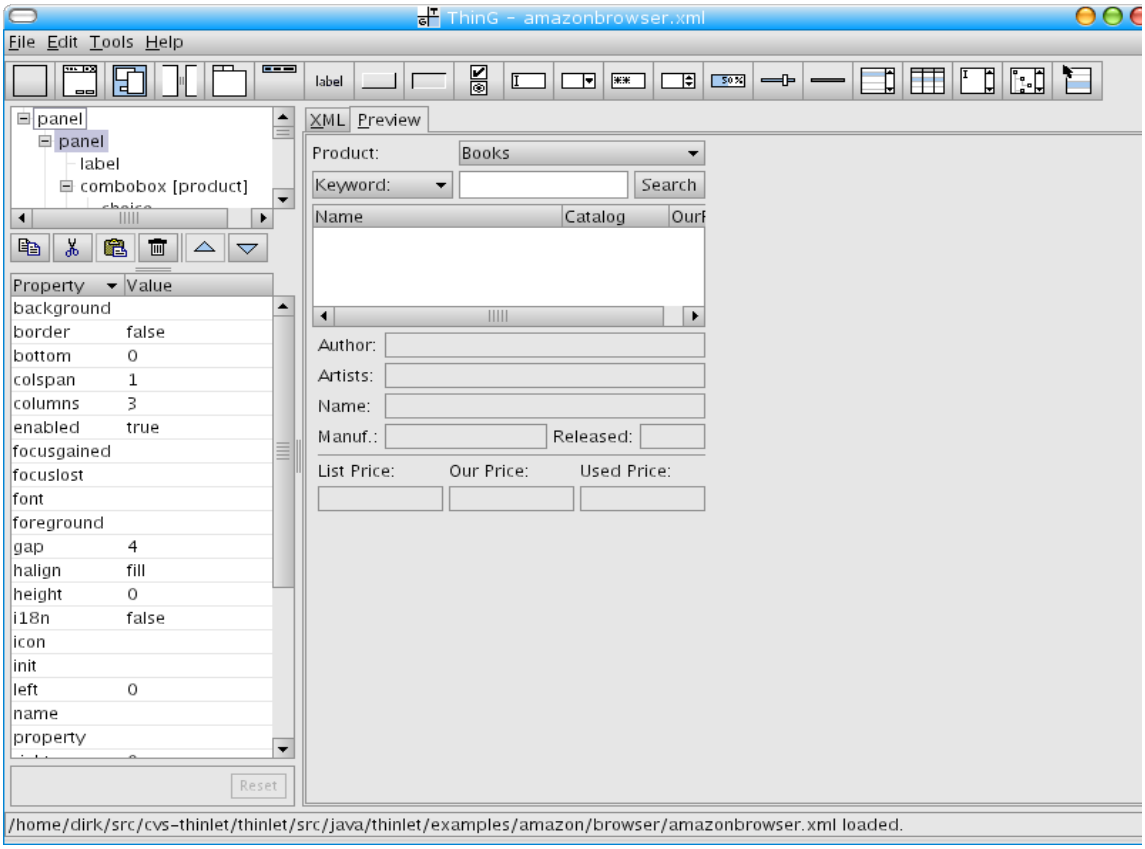


Figure4: ThinG, a Thinlet editor that uses Thinlet itself

3.4.SWING/JFC

SUN released the Swing library in 1998. Swing is Java's user interface toolkit, written in pure Java, capable of building platform-independent user interfaces. Swing is the project code name for the lightweight GUI components in Java Foundation Classes[31] (JFC). These components are included in the JFC, a set of Java class libraries provided as part of Java 2 Platform, Standard Edition (J2SE), to support building graphical user interfaces (GUI) and graphics functionality for client applications that will run on popular platforms such as Microsoft Windows, Linux, and Mac OSX.

Swing contains all the components that may be expected to be seen in a modern UI, from buttons that

31 http://java.sun.com/products/jfc/

contain pictures to trees and tables. The MVC design pattern is used as the basis for each individual Swing user interface component. Swing supports a feature known as 'pluggable look and feel'. This enables the Swing GUI components to support different look-and-feels while maintaining a consistent API for the component, allowing developers to code their application GUI once, yet run it with multiple look-and-feels. In the J2SE platform, Swing provides a Java technology look and feel that is consistent across platforms (the default), as well as native-platform look-and-feels for Microsoft Windows and Motif (Linux/GTK is in progress). Apple provides their own look and feel for Mac OSX. Applications can choose between running the Java technology look and feel or picking up the appropriate platform-native look and feel at runtime. The Swing Pluggable look and feel API is designed to be extended so that developers can create custom look-and-feels.

Swing applications can run as standalone applications, or embedded within HTML pages as applets.

The choice of Java as the implementation language forces a choice on the execution environment, but fortunately the Java platform has been ported to all the major hardware platforms, and some of the minor ones as well (PDAs, mobile phones).

3.4.1.Evaluation

- Reliability: JFC is core to the Java platform. It was first released by SUN in 1998, and since then has reached a high degree of maturity. The proof of its maturity is the existence of powerful, robust and widespread applications. An extensive list of them is maintained on the “Swing Sightings Index”[32] web page. They include:
  - NetBeans, a Java IDE from SUN
  - IntelliJ, a Java IDE from IDEA
  - Poseidon, a UML editor from Gentleware
- Continuity: Continuity can be assummed, based on the fact that JFC is core to the Java Platform.
- Portability: Being a Java toolkit that is part of the Java 2 Platform Standard Edition, the portability is guaranteed to all those platforms for which a Java 2 Platform exists.
- Versatility: JFC is a Java technology which provides seamless access to all the Java libraries. JFC is a fully-featured and mature toolkit. There are many applications developed using this toolkit, including large commercial applications like Borland's JBuilder and IntelliJ's Idea. See the paragraph titled “Reliability” above.
- Freedom: JFC is part of the Java 2 Platform, Standard Edition (J2SE), and therefore is licensed under the same terms as J2SE. J2SE is licensed under Binary Code Licenses[33],[34]. This means basically that Sun does not charge for the use or distribution of JFC.
- Internationalisation: Swing supports internationalised applications by means of resource files containing appropriate messages.
- Pervasiveness: Easy access to User Oriented Services could be provided by using Java Applets, without the need for local deployments of additional software.
- Learning curve: JFC has a medium complexity. In the “Java Foundation Classes”[35] page there are links to API Specifications, Documentation and FAQs . There are many commercial books. There is a free (free as in ``free beer'') mature IDE, named NetBeans.

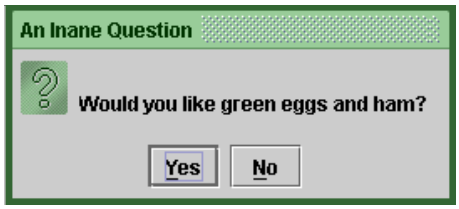
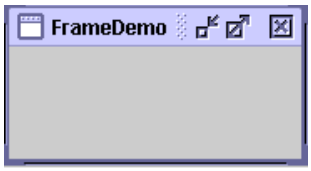
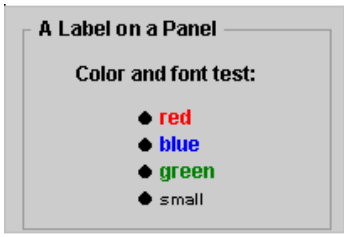
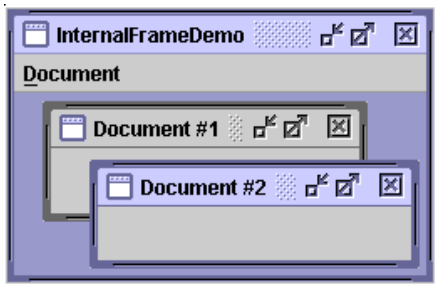
32 http://java.sun.com/products/jfc/tsc/sightings/index.html

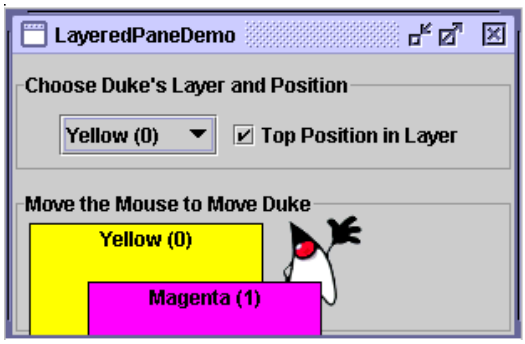
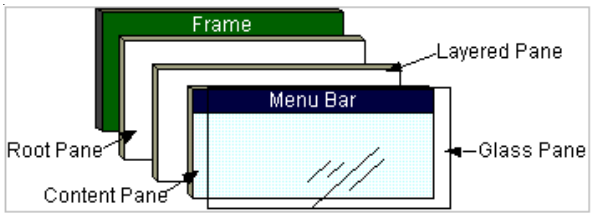
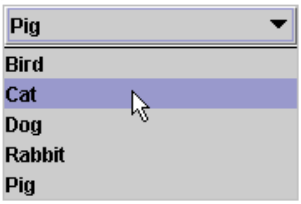
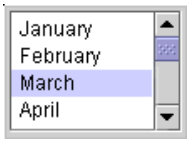
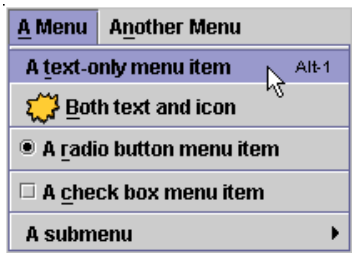
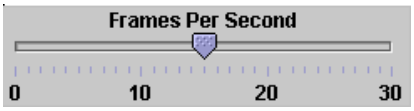
33 http://java.sun.com/j2se/1.4.2/j2re-1_4_2_08-license.txt

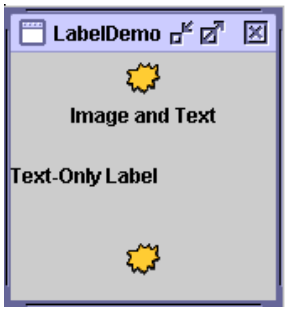
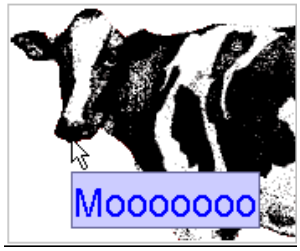
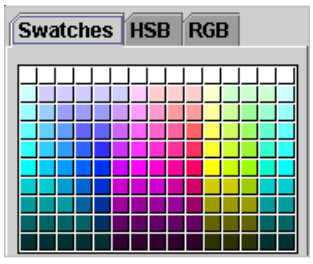
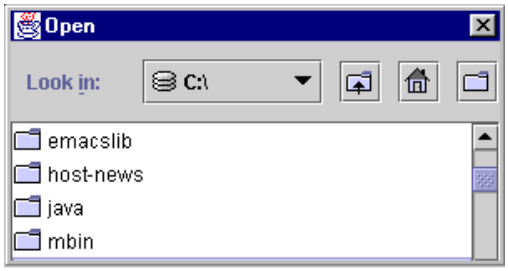
34 http://java.sun.com/j2se/1.5.0/jre-1_5_0_02-license.txt

35 http://java.sun.com/products/jfc/index.jsp

3.4.2.Widgets

Dialog	
Frame	
Panel	
Scroll pane	
Split pane	
Tabbed pane	
Tool bar	
Internal frame	

Layered pane	
Root pane	
Buttons	
Combo box	
List	
Menu	
Slider	
Spinner	
Text field	

Label	
Progress bar	
Tool tip	
Color chooser	
File chooser	
Table	
Text	

Tree	

3.5.SWT/JFACE

SWT (Standard Widget Toolkit)[36] is a user interface toolkit released in November 2001. It was actually developed by the Eclipse Foundation[37] for the Eclipse IDE. It uses explicitly standard Java, with the approved Java Native Interface, but the toolkit cannot be described as "pure" because of the reliance on a platform-native library for every computer on which it runs. Currently, implementations of SWT are available for all major platforms.

SWT contains all the components that may be expected to be seen in a modern UI, from buttons that contain pictures to trees and tables. SWT has a higher-level JFace API, layered on top of SWT but without "hiding" it, to provide classes and functions to handle common tasks associated with programming using SWT.

The widgets that the SWT component provides are those that are provided by the operating systems on which SWT runs, implying that the look and feel of applications built with SWT will vary on each operating system.

The SWT applications are mainly developed to be executed in the Eclipse IDE. Alternatively it is possible to develop applications to be executed in the form of standalone applications. The SWT applications cannot be executed as Java Applets.

3.5.1.Evaluation

- Reliability: SWT is core to the Eclipse platform. It was first released by IBM in 2001, and since then has reached a high degree of maturity. The proof of this maturity is the existence of the powerful, robust and widespread Eclipse IDE and associated plug-ins. In the “open source projects and plug-ins”[38] and “commercial projects and plug-ins”[39] web-pages there are extensive lists of plug-ins.
- Continuity: Continuity is assummed, based on the fact that SWT is core to the Eclipse Platform.
- Portability: The SWT Standard Widget Toolkit home page[40] provides an explanation of SWT portability. This page claims that portability is supported by ensuring that all *but the lowest-level*, direct interface to the operating system is written in Java. This “lowest level” implies that SWT needs to be ported separately for every Operating System. To run an SWT application it is necessary to have a Java 2 Platform Standard Edition (J2SE) installed, as well as a ported SWT library.
- Versatility: SWT/JFC is a fully-featured toolkit, and the best proof of its versatility is the existence of the Eclipse Project[41]. SWT/JFC is preferentially used in the context of the Eclipse Project, and

36 http://www.eclipse.org/swt/

37 http://www.eclipse.org/org/

38 http://www.eclipse.org/community/osplugins.html

39 http://www.eclipse.org/community/commercialplugins.html

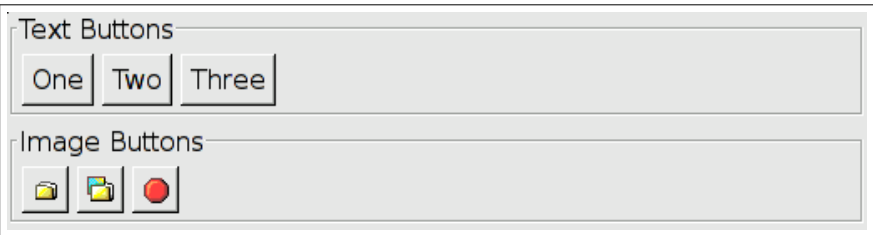
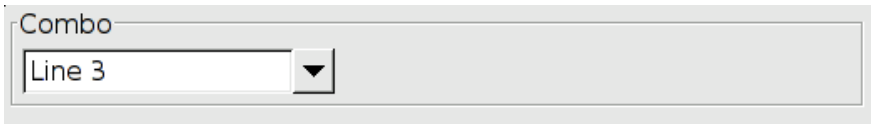
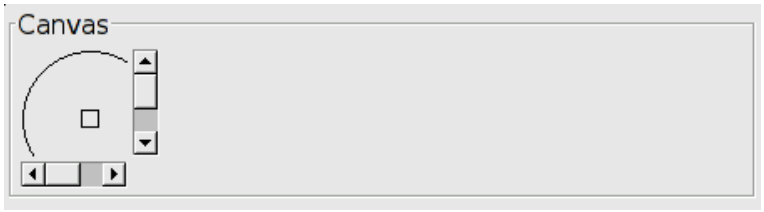
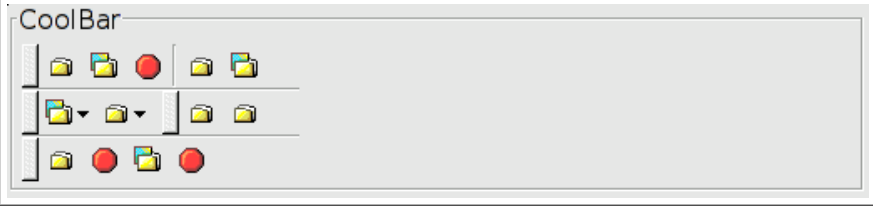
40 http://www.eclipse.org/swt/

41 http://www.eclipse.org/eclipse/

there are not many standalone applications outside of this context. Some standalone examples are listed in the SWT Standard Widget Toolkit - Community Page[42].

- Freedom: This toolkit is provided under the terms and conditions of the Common Public License Version 1.0 ("CPL")[43]. The CPL is a copyleft license, broadly similar to the GNU General Public License[44] in its terms. The main addition is a patent clause designed to prevent unscrupulous contributors from contributing code which infringes on their patents, and then attempting to charge royalties.
- Internationalisation: SWT/JFACE supports internationalised applications by means of resource files containing appropriate messages.
- Pervasiveness: SWT/JFACE User Interfaces cannot be downloaded using Java Applets, therefore there is no access to the User Oriented Services without local deployment of software.
- Learning curve: SWT/JFACE have a medium complexity. In the “SWT Standard Widget Toolkit - Development Resources”[45] page there are links to API Specifications, Documentation, FAQs and other information. There are some commercial books. There is a free plug-in (free as in ``free beer'') named “Visual Editor” that may be plugged into the Eclipse IDE, but it doesn't seem to be mature as it has many opened bugs.

3.5.2.Widgets

Buttons	
Combo	
Canvas	
CoolBar	

42 http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-swt-home/external.html

43 http://www.eclipse.org/legal/cpl-v10.html

44 http://www.gnu.org/licenses/gpl.html

45 http://java.sun.com/products/jfc/index.jsp

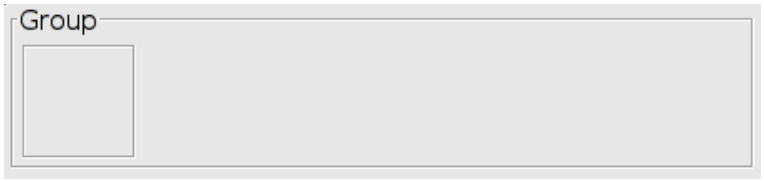
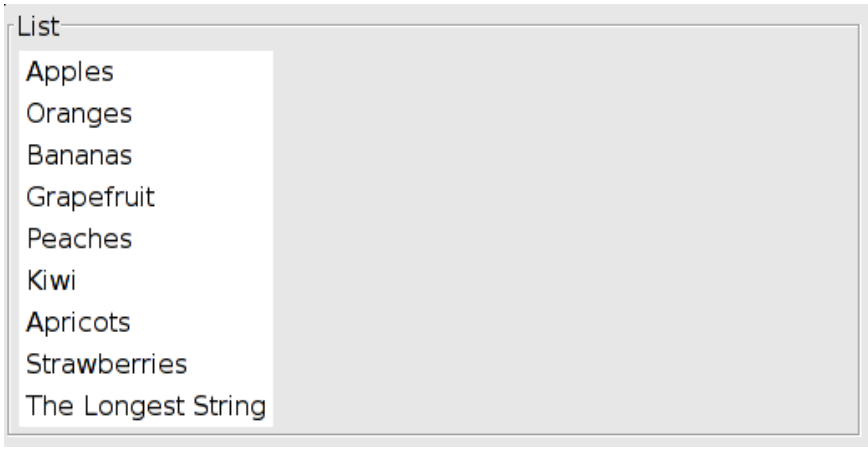
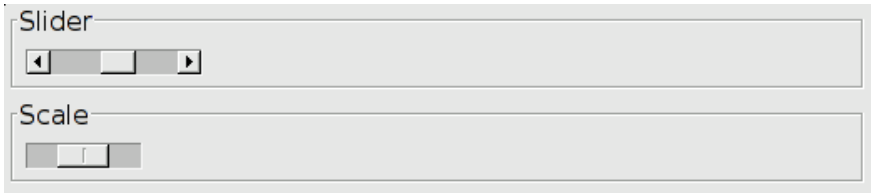
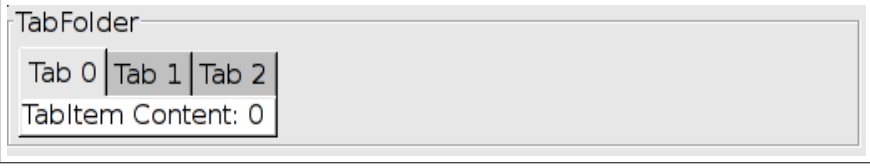
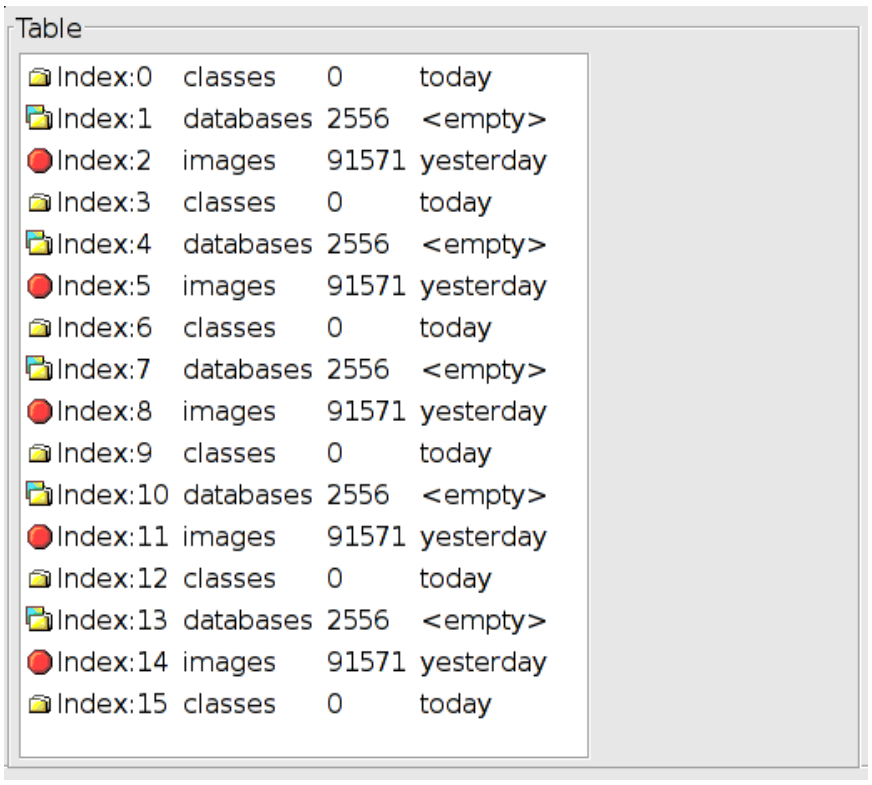
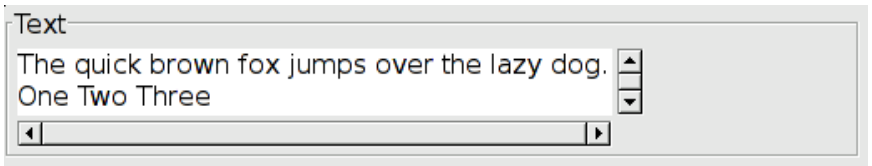
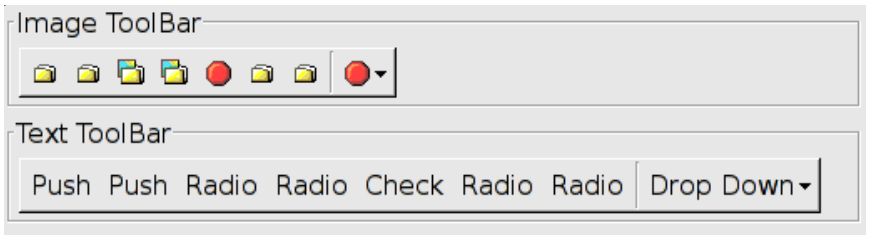
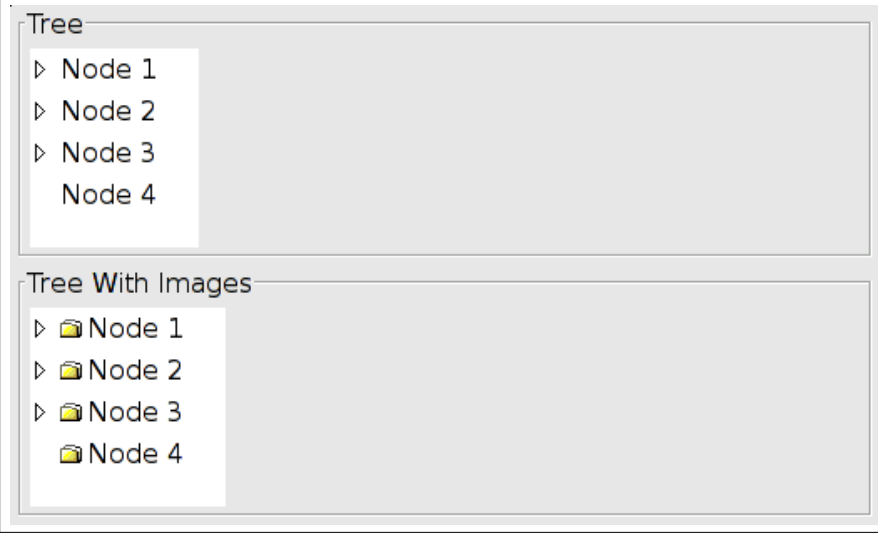
Dialog	
Group	
Labels	
List	
ProgressBar	
Slider/Scale	
TabFolder	

Table	
Text	
ToolBar	
Tree	

4.AJAX

AJAX stands for “Asynchronous JavaScript language and XML”. It's not a framework for UI creation, but rather a combination of technologies that are used to create UIs to be consumed by a web browser that supports Javascript. The term was first used by Jesse James Garrett in February 18th 2005[46].

The following technologies comprised make up the AJAX framework:

- XHTML (or just HTML) and CSS for the rendering of information in the web browser.
- The DOM (Document Object Model) of the page is manipulated through JavaScript. This technique is also used in DHTML[47].
- The XMLHttpRequest object to exchange data with the web server.

Any web application that uses these three techniques is said to be written in AJAX.

4.1.Evaluation

- Reliability: Not being a technology in itself, there is not much to say about reliability. The technologies in which AJAX is based are open standards, and supported by web browsers, so the reliability is subject to the reliability of web browsers themselves, and their conformance to standards.
- Continuity: Similarly, the continuity of AJAX is subject to the continuity of the support of browser vendors to open standards.
- Portability: As long as browsers support the involved technologies, AJAX applications are readily portable, mainly because they are not ported, but rather executed from different environments.
- Versatility: Through the use of Javascript the DOM that defines the structure of the (X)HTML document may be changed at runtime. There is a limit in what can be done through this technique, but it allows a great deal of creativity to be expressed.
- Freedom: Again, this question does not belong to something that is not a realization of a technology itself. AJAX is as free as browsers are: as long as there is freedom of choice for the web browser, an AJAX application can be executed.
- Internationalisation: The internationalisation is left entirely to the writer of the AJAX application.
- Pervasiveness: Except in very limited environments, JavaScript capable browsers are available anywhere in the world. The pervasiveness of an AJAX application is almost guaranteed.
- Learning curve: This is the tougher point. To develop an AJAX application a developer must learn how to use three different technologies.

46http://www.adaptivepath.com/publications/essays/archives/000385.php

47http://www.w3.org/DOM/faq.html#DHTML-DOM

5.OpenLaszlo

It is a product of Laszlo Systems. OpenLaszlo is basically a Java web application that is able to turn a file with the description of the UI into an swf (MacroMedia Flash) file, that is then loaded and executed by the web browser, provided that it has installed the proper plugin. The latest version is able to generate DHTML as well, so interoperability has been enhanced.

It comes with a variety of widgets that are represented in a workspace/desktop within a web page. These widgets are drawn by the flash application entirely, so everything happens in the web browser. Only when submissions of forms or data retrieval is to be performed the flash application communicates with the servlet engine, which holds the OpenLaszlo servlet and other libraries.

Communication with external entities is mainly performed by the servlet itself, and the results are sent back to the flash application in the browser using an internal protocol. OpenLaszlo applications can also invoke directly web services without passing through the OpenLaszlo servlet.

5.1.Evaluation

- Reliability: OpenLaszlo is the open source version of the Laszlo platform. It went open source under the CPL license in October 2004. It seems to be mature enough. To run, it needs a web browser with Macromedia Flash support. Macromedia Flash was a rebranding performed in December 1996 of an earlier product of Macromedia, called FutureSplash Animator, which in turn was based on another Macromedia product, called FuturePecoraro Animator. The technology has been around for several years now, and the latest release is Flash 8 Pro. The OpenLaszlo engine may produce a flash file suitable for versions 6, 7 or 8 of the Macromedia Flash player, a behaviour configurable in the OpenLaszlo engine itself. However, it is not clear how future versions of the player will affect actual versions of the OpenLaszlo web application. It must be noted that the latest version offers the possibility to generate DHTML instead of an swf file, therefore widening its user base, as DHTML is supported by more browsers than swf files. The usability is not handicapped by chossing the DHTML format, the same widgets are available.
- Continuity: Nothing can be said about the continuity of companies or their products, but Macromedia doesn't seem inclined to go out of business in the near future. Although they have opened their swf format to the open source community, they forbid the creation of swf players[48]. The implication is that should they stop providing the player for free, or go out of business, it is not likely that anyone else is authorized to continue providing players for the swf format. But the same can be said about Sun Microsystems and the Java platform. In this respect it plays more or less like Java Applets or Swing rich clients. Due to the support for DHTML, which has been around for some years now, and the fact that it is based on open standards, the continuity has been much improved.
- Portability: As the OpenLaszlo technology is based in a Java web application that doesn't run in the client, the portability issue here is bound to the portability of the swf player. Macromedia provides the player for free for Windows, Linux, Solaris and MacOS X, and maybe others (their download page doesn't allow to see the complete list easily[49]). Again, the DHTML interface that can be generated with the last version of OpenLaszlo ensures a much higher portability.
- Versatility: The widgets available with OpenLaszlo include the usual widgets available in an HTML page, plus some of its own, including windows that live in the desktop provided by OpenLaszlo, playing sound files or video streams. The platform may perform remote procedure calls in XML-RPC, SOAP or by calling Java classes. Version 3.1 of the OpenLaszlo framework provides an AJAX API. This seems to be more than enough for most applications. The DHTML format seems to cover the same set of widgets.

48 http://www.macromedia.com/licensing/developer/

49 http://www.macromedia.com/go/getflashplayer

- Freedom: Given the dependency on the Macromedia Flash player, the freedom is quite limited. There is, basically, only one provider. The web browser choice is less constrained, as long as there is a version of the Flash player available for it. If the DHTML format is chosen, there is much more freedom, because most web browsers support DHTML, a technology that is not exactly new.
- Internationalisation: There doesn't seem to be support (yet) for externalizable strings in OpenLaszlo, unless some tricks are used, which require a rather deep knowledge of the platform[50]. Unicode is supported.
- Pervasiveness: It's true that OpenLaszlo was tightly bound with the Macromedia flash player. But it is also true that the flash player is widely deployed (600 million Internet-connected desktop and mobile devices, according to the MX Developers Journal[51]). The DHTML interface makes it much more pervasive.
- Learning curve: Designing a UI in OpenLaszlo seems to be an easy affair. The description of the UI is done through an xml-based file. The source of documentation seems to be the OpenLaszlo home site, which is a kind of single point of failure regarding documentation. The documentation, however, comes along with the OpenLaszlo downloadable bundle. Also there is already an Eclipse plugin for the creation of OpenLaszlo-based UIs[52].

50 http://www.laszlosystems.com/developers/community/forums/showthread.php?s=740ff373a87066181d115cfeaa740cd6&threadid=2769&highlight=i18n

51 http://mxdj.sys-con.com/read/110966.htm

52 http://eclipse.org/laszlo/

5.2.Screenshots

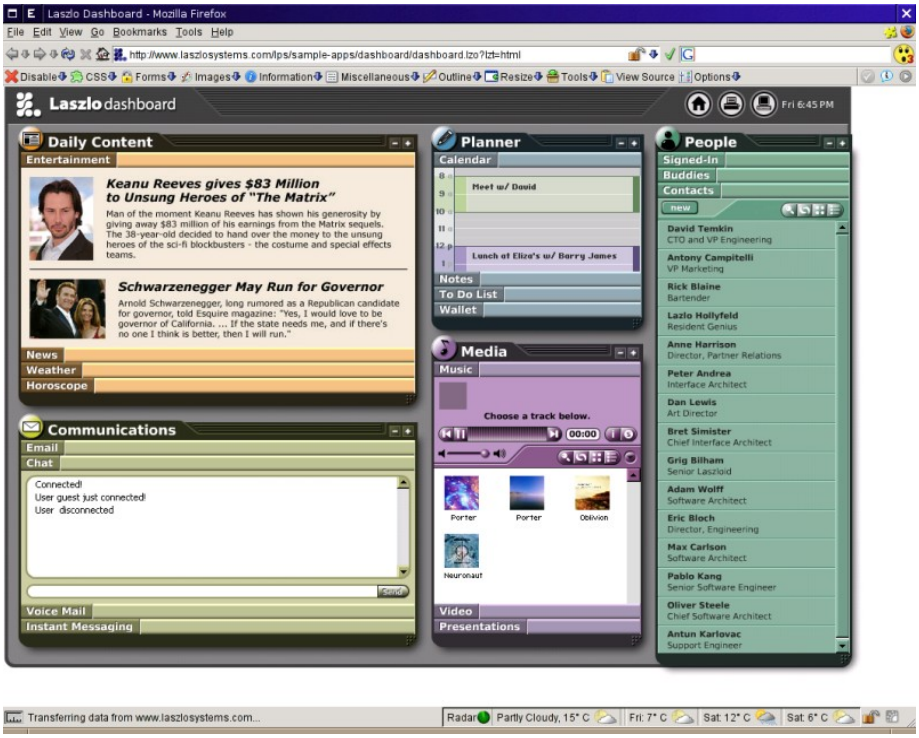


Illustration 5: SWF (flash) based UI

Illustration 6: DHTML based UI

6.Conclusions

Different GUI Frameworks have been evaluated in the previous chapters. This chapter is a summary, focused on the coverage of the different criteria, in order to see which frameworks are candidates for use in the implemention of User Oriented Services.

XForms passes the Continuity, Portability, Versatility, Freedom, Internationalisation, and Pervasiveness, but fails to cover the Reliability and Learning curve criteria. Reliability is a very critical criterion, for this reason XForms should be discarded.

XUL passes the Continuity, Portability, Versatility, Freedom, Internationalisation, and Pervasiveness criteria, but don't pass the Learning curve criteria. It is better than XForms in reliability terms. This is proven by the Mozilla suite, but the lack of applications outside of the Mozilla suite may be interpreted as a warning with respect to the Reliability. The doubts with the Reliability criterion and the fact that it doesn't pass the Learning curve criterion suggest the XUL framework should be discarded.

Thinlet passes the Portability, Versatility, Freedom, Internationalisation and Pervasiveness criteria, but not the Reliability, Continuity and Learning curve. For small applications, not passing the Learning curve criterion may not be critical. Reliability and Continuity are very critical criteria, therefore Thinlet should be discarded.

SWING/JFC passes the Reliability, Continuity, Portability, Versatility, Freedom, Internationalisation, Pervasiveness and Learning curve criteria. To pass the Pervasiveness criteria Java Applets must be used. This framework passes all the criteria, therefore is a candidate framework.

SWT/JFACE passes the Reliability, Continuity, Portability, Versatility, Freedom, Internationalisation and Learning curve criteria but don't pass the Pervasiveness criterion. Pervasiveness criteria is a critical criterion, therefore SWT/JFACE should be discarded.

AJAX is not a technology in itself, but rather a way to do things. As such, it basically leaves the writers of the UI on their own, because it doesn't even dictate or guide *how* things should be done. Put in other words, it fails the Learning Curve criterion. For these reasons we can not recommend AJAX as a technology for UI creation.

OpenLaszlo seems to be easy to use, powerful and appealing to the eye. Furthermore, the computing team of the DBE seems to be rather enthusiastic about it, and are already using it for some demo services. The available editor for the Eclipse platform and its support for DHTML-based interfaces make it more easily usable by developers and consumer.

Therefore, scoring quite high in the evaluation parameters, and being already accepted by the DBE computing team, OpenLaszlo seems to be an acceptable choice.