# Extending Desktop Applications to the Web

Arno Puder

San Francisco State University
Computer Science Department
1600 Holloway Avenue
San Francisco, CA 94132
arno@sfsu.edu

**Abstract.** Web applications have become the major means to allow ubiquitous access to backend systems via a web browser. Several technologies such as JSP, ASP.NET, or Java Server Faces exist today that help in developing web applications. These technologies do not support the migration of existing legacy desktop applications written with a GUI class library such as Swing, Qt, or GTK to web applications. The framework presented in this paper allows the programmer to expose arbitrary desktop applications as web applications without requiring any changes in the source code of that application. Dialogs are rendered using HTML and JavaScript and a flexible event model transparently forwards user interaction at a web browser to the application running at the backend. By employing a cross-language compiler it is also possible to transparently execute parts of the application on the client side inside the web browser.

## 1 Motivation

Web browsers are generic clients that allow access to arbitrary services. Initially, the standards of the World-Wide Web were document-centric and web browsers were mainly used to view documents. With the rise of eCommerce, these standards were extended to allow for operational interaction. The main protocol for a web browser to invoke an operation at a web server is via the Common Gateway Interface (CGI). The least common denominator for writing web applications are HTML/HTTP, JavaScript and CGI. Even those few standards have different implementation in different browsers and the developer needs to be careful to create portable web applications. All in all, it is difficult to develop portable web applications. Different technologies exist today to help with the development of portable web applications.

A different domain for Graphical User Interfaces (GUI) is the desktop. Several different class libraries exist that ease the development of desktop applications. Among the more popular class libraries are Swing, Qt, GTK, and MFC. Some libraries target a specific platform (e.g., MFC targets the Windows environment) while others allow the development of cross-platform applications (e.g., Qt is available for Unix and Windows environments).

The novel idea proposed in this paper is to bridge the chasm between desktop and web applications by introducing a framework that allows to expose desktop applications as web applications. This can be achieved without any modifications to the desktop application. The benefit of this approach is that it is easy to port legacy applications to web applications. It also allows the development of new web applications based on well understood paradigms, thereby making it unnecessary to understand web technologies.

Section 2 gives a brief overview of existing web application development technologies. Section 3 introduces a simple example showing how to implement a desktop application based on Swing. Section 4 introduces the framework. Finally, Section 5 provides and outlook for future work.

## 2 Web–Application Methodologies

This section gives a short overview of existing methodologies that facilitate the construction of web applications. By the term “web application” we mean interactive applications that can be accessed through a web browser. There are three different basic approaches to develop web applications:

1. HTML centric: the HTML page that is rendered by the browser is created by a program.
2. Template centric: a template that contains HTML and program code, creates the HTML page that is rendered by the browser.
3. Library centric: a library encapsulates the creation of an HTML page.

An example of a technology that uses the first approach are CGI scripts and servlets. The problem with that approach is that there is no clear separation between flow of control and the user interface as mandated by the MVC (Model View Controller) paradigm (see [2]). An example of the second approach is JSP (Java Server Pages). An HTML page is interspersed with Java code that allow the creation of dynamically generated HTML pages. Here again, there is no clear separation between flow of control and user interface.

Both of these approaches require intimate knowledge of HTML as well as a programming language that is used for the dynamic aspects. The third approach for developing web applications is through class libraries. In some sense, it is a specialization of the HTML centric approach. The main difference is that with the library centric approach knowledge of HTML is encapsulated and thereby hidden in the class library which is why we view it as a separate approach. One example of a technology that uses the library centric approach is ASP.NET. The interesting fact is that ASP.NET contains a specific class library specially for web applications. It is not possible to expose a legacy MFC (Microsoft Foundation Classes) application as a web application.

There are many different class libraries that support the development of desktop applications: MFC, GTK, Qt, Swing are only a few of them. The framework introduced in this paper makes it possible to expose those applications as web applications without having to change a line of code of the legacy application.

## 3 Sample Swing application

To demonstrate our approach, we have chosen a simple Swing application from Sun’s own Swing tutorial (see [1]). The application allows the conversion from degrees Celsius to Fahrenheit. The following listing is an excerpt from the original source code of the tutorial:

```
 1:  import java.awt.*;
 2:  import java.awt.event.*;
 3:  import javax.swing.*;
 4:
 5:  public class CelsiusConverter implements ActionListener {
 6:    JFrame converterFrame;
 7:    JTextField tempCelsius;
 8:    JLabel celsiusLabel, fahrenheitLabel;
 9:    JButton convertTemp;
10:
11:    public CelsiusConverter() {
12:      converterFrame = new JFrame("Convert Celsius to Fahrenheit");
13:      // ...
14:      tempCelsius = new JTextField(2);
15:      fahrenheitLabel = new JLabel("Fahrenheit", SwingConstants.LEFT);
16:      celsiusLabel = new JLabel("Celsius", SwingConstants.LEFT);
17:      convertTemp = new JButton("Convert...");
18:      convertTemp.addActionListener(this);
19:      // ...
20:    }
21:
22:    public void actionPerformed (ActionEvent event) {
23:      int tempFahr = (int)((Double.parseDouble(tempCelsius.getText()))
24:                           * 1.8 + 32);
25:      fahrenheitLabel.setText(tempFahr + " Fahrenheit");
26:    }
27:
28:    public static void main(String[] args) {
29:      CelsiusConverter converter = new CelsiusConverter();
30:    }
31:  }
```

All Swing applications import packages that provide the basic functionality [Lines 1–3]. If an application needs to respond to events such as the pressing of a button, it has to implement the `ActionListener` interface [Line 5]. The constructor of the application defines a frame [Line 12] and several widgets such as labels, entry fields and buttons [Lines 14–17]. Note that all the classes such as `JButton` used in this code come the Swing library. By associating the button with the action listener [Line 18], the method `actionPerformed` [Line 22] will be called every time the button is clicked. In this event, the Celsius value is read from the entry field [Line 23]. After the conversion to Fahrenheit, the value is written to the appropriate label [Line 25]. The main function simply creates an instance of the application [Line 29].

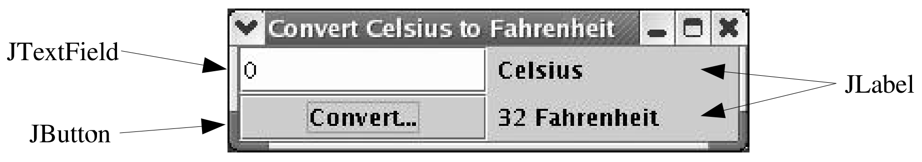

**Fig. 1.** Native Swing application.

Figure 1 shows the Celsius converter application running as a native Swing desktop application. The screenshot shows the conversion of 0 degree Celsius to 32 Fahrenheit. The user entered 0 in the input field. After clicking on "Convert..." the label in the lower right corner changes to "32 Fahrenheit". The framework introduced in the following section will allow to run the same application inside a web browser. This will be accomplished without changing a line of code in the original Celsius converter application.

## 4 Framework

This section introduces the framework that allows easy migration of existing desktop applications to web applications. Section 4.1 outlines the design goals we want to achieve with our framework. Section 4.2 introduces the general architecture of the framework and Section 4.3 makes some comments regarding our reference implementation.

### 4.1 Design goals

A GUI application is usually structured as shown in Figure 2. The layering depicts the dependencies between the different components. The GUI library typically uses the functionality offered by the operating system while the application depends on the functionality of the operating system and the GUI library. The horizontal lines denote API (Application Programming Interfaces). A GUI library exposes its functionality through a specific API to the application.

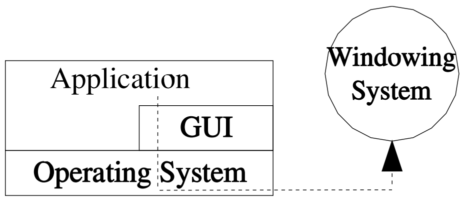

**Fig. 2.** Structure of a GUI application.

Based on this background, our framework is designed to achieve the following goals:

**Language independence:** we do not impose a particular programming language since legacy applications can be written in many different languages.
**GUI library independence:** we want to support a wide variety of different GUI class libraries.
**Browser independence:** the web applications should run on a wide variety of popular browsers.

Since there exist several different GUI-libraries for different programming languages, the framework should offer support for all of these programming languages. The same applies to the GUI-libraries themselves. Since there are many different programming languages and GUI class libraries, a complete implementation of the framework will require significant efforts. Our last goal is browser independence. The idea is to make as few assumptions about the browser's capabilities as possible. For that reason, we do not assume any special plugins such as Java Virtual Machine. The only capabilities we assume are subsets of HTML and JavaScript that are known to be supported by all major browsers.

## 4.2 Architecture

Figure 3 shows the architecture of our framework. The Application shown here is unchanged from its desktop version. This implies that the GUI-proxy layer has to offer the identical API as the GUI-library shown in Figure 2. Since there are several GUI-libraries, it is beneficial to structure the GUI layer into a proxy and a Unified GUI (UGUI) library. The purpose of the latter is to provide a unified library while the proxy offers the functionality of the UGUI in the way the application expects it. This way it is possible to reuse significant portions of the implementation and thereby facilitates to support other GUI libraries.

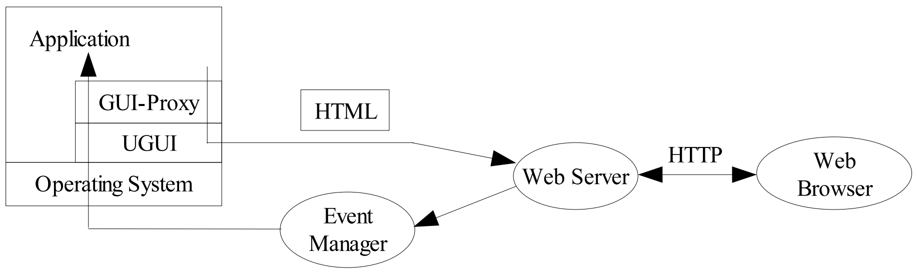

**Fig. 3.** Architecture.

The UGUI layer emits the content of a window in HTML that the browser subsequently renders. The user can then populate input fields displayed in the form. Every user action, that would result in a callback to the application, will initiate an HTTP POST request. This will initiate an interaction with the web server that will forward the POST request to an event manager. The event manager relays the user action to the UGUI. The user action is then delivered to the application via some GUI-library specific mechanism. After the application has handled the event, an updated version of the window content is created via the translator and send back to the browser.

Depending on the kind of user interaction this might result in a very inefficient implementation. E.g., a `MouseListener` would trigger a page reload every time the user moves the mouse. While it would be possible to mimic this behavior inside the browser through an appropriate JavaScript function, it would clearly result in very inefficient runtime behavior. One way to deal with these situations will be outlined in the last section where we discuss future work.

### 4.3 Reference implementation

We have implemented a prototype based on the framework introduced in the previous section. As mentioned earlier, a complete implementation would require support for several programming languages as well as GUI class libraries. While this would be possible with appropriate resources, we have limited our prototype to support only one programming language and a subset of one GUI class library. Our goal was to run the Celsius Converter program introduced in Section 3 as a web application. For that purpose the current prototype supports Java applications using the Swing GUI-class library. Because of the GUI-proxy, the Celsius Converter application does not need to be modified from its original version. The application can be accessed from any web browser. User interaction like pressing the "Convert..." button results in transparently calling the `actionPerformed()` method of the Celsius Converter application.

## 5 Conclusions and Outlook

Easy migration of existing desktop applications to web applications has huge benefits with regards to investment savings. Instead of manually developing a web-based user interface, this task can be automated by the framework introduced in this paper. The framework is general enough to be applied to different GUI class libraries. We intend to verify our statements by supporting other libraries. However, we have not yet studied the question if all aspects of a GUI class library can be turned into a web application. Some of the more exotic capabilities might be difficult; if not impossible to support.

In the current version of our framework the user interface is passive; the application logic is completely located on the server side. This incurs a high overhead for fine grained operations that are triggered by user interaction such as the aforementioned mouse events. We plan to extend our framework by migrating application code from the client to the server side. Methods of the application can be labeled as client side methods through JavaDoc style comments. The source code of those methods will be translated to JavaScript in order to be able to execute it inside a browser. This opens up a field of interesting research questions. E.g., if the client side method calls a server side method, this requires a middleware to transport the actual parameters.

## References

1. C.S. Horstmann and G. Cornell. *Core Java 2, Volume I: Fundamentals.* Prentice/Hall International, sixth edition, 2002.
2. G. E. Krasner and S. T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August 1988.