Accessing X Applications over the World-Wide Web

Arno Puder and Siddharth Desai

San Francisco State University
Computer Science Department
1600 Holloway Avenue
San Francisco, CA 94132
{arno|sgd1977}@sfsu.edu

Abstract. The X Protocol, an asynchronous network protocol, was developed at MIT amid the need to provide a network transparent graphical user interface primarily for the UNIX Operating System. Current examples of Open Source implementations of the X server, require specific software to be downloaded and installed on the end-user's workstation. To avoid this and other issues involved in the conventional X setup, this paper proposes a new solution by defining a protocol bridge that translates the conventional X Protocol to an HTTP-based one. This approach makes an X application accessible from any web browser. With the goal of leveraging the enormous browser install base, the web-based X server supports multiple web browsers and has been tested to support a number of X clients.

1 Motivation

The staggering rate at which the World-Wide Web has grown over the last decade is evidenced by the number of websites that are accessible over the Internet today. Web browsers, the end-user applications that connect to these websites and display content have also evolved at an impressive rate. Originally based on a document-centric architecture, where a web browser would only display static HTML pages, much work has been done to define extensions that allow for operational interactions. Thus web browsers have generic interfaces for web-based applications.

Although the most popular browsers are freely available and readily downloadable from the Internet, most operating systems today bundle a web browser. As a result, the install base of web browsers has dramatically increased. This has been leveraged by companies that do business online and software providers who have migrated their native client interfaces and made them web-based. The obvious technical benefit is that with all prerequisite software already installed, installation and configuration for such web-based applications is dramatically reduced or even eliminated altogether.

In this paper we introduce a technique that allows to access X applications – based on MIT's X Protocol – through a web browser. This will make X applications that traditionally require an X server to be installed, accessible in a web

browser without having to modify and rebuild those applications. Our proposal is based on a protocol bridge, that translates the X Protocol to an HTTP-based one.

This paper is organized as follows: in Section 2 we give a brief introduction to generic clients. Section 3 will discuss various design alternatives. Section 4 introduces the architecture of our protocol bridge we call XWeb. Section 5 describes our prototype implementation of XWeb as well as some performance measurements. Section 6 discusses related work, and in Section 7 we provide a conclusion and outlook.

2 Generic Clients

In this section we introduce the notion of a *generic client*. A generic client is controlled by an end-user to access a remote application. We use the term generic to denote the fact that the client is not specialized for any particular application, but rather provides access to *a priori* unknown applications. Two examples of generic clients that we will discuss here are X servers and web browsers.

By being a network-aware graphical user interface system, X allows the separation of an application's processing and its output (see [8]). In X Windows, the application is called the X client, whereas the output is rendered at the X server. The X server is therefore located at the side of the end-user (see Figure 1).

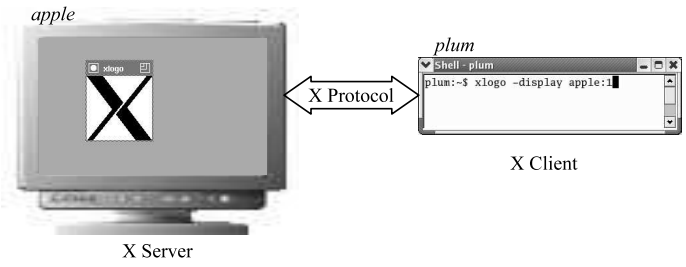

Fig. 1. X Window Protocol.

The X client and X server communicate with each other via the X Protocol. The X Protocol is an asynchronous, binary protocol. The X client sends the window content to be rendered to the X server, whereas the X server sends user input (e.g., keystrokes or mouse movement) to the X client. Interestingly, the X Protocol does not support widgets such as buttons, listboxes, or radio buttons. In X, everything is rendered as a graphical image. If the user interface of an X client requires a button, it needs to be manually drawn by the X client. This

accounts for the fact that many different X applications have a different look-and-feel, since every application can draw its user interface widgets in a different way.

Because the X Protocol is about images and not widgets, another implication is that there is a fair amount of network traffic between an X client and the X server. E.g., every keystroke of the user requires a roundtrip communication with the remote application. The consequence is that the X Protocol works best in low-latency, high-bandwidth networks.

Another example of a generic client is a web browser. Initially meant to render static documents only, the World-Wide Web (WWW) has evolved to support operational interactions. User interfaces are described through HTML (Hyper-Text Markup Language) that are downloaded via HTTP (Hyper-Text Transport Protocol) from a remote web server. HTML allows the description of feature-rich user interfaces supporting widgets such as buttons and listboxes. Since a web browser supports the most common widgets natively, the look-and-feel of web applications is more homogeneous than that of X applications.

Table 1 summarizes the main differences between web browsers and X servers. Whereas an X server effectively only render images, web browsers support complex widgets. Web browsers also feature an execution platform based on Java and JavaScript that allows application-specific code to run on the client side. The X Protocol is an asynchronous protocol, by which we mean that both X client and X server can send Protocol Data Units (PDUs) independent of each other. On the other hand, HTTP is a synchronous protocol where the web browser determines when to interact with the remote web server. The web server cannot send a PDU independently to the web browser.

	X server	**Web browser**
Granularity	Images	Widgets
Execution platform	None	Java, JavaScript
Communication	Low latency	High latency
Protocol	X Protocol (asynchronous, binary)	HTTP (synchronous)
Platforms	WeirdX, Cygwin/X, HummingBird Exceed	Mozilla, Firefox, IE

Table 1. Comparing X servers and web browsers.

The goal of this paper is to introduce an architecture that allows access to X clients from within a web browser. While no one disputes the ubiquity of web browsers that would give almost universal access to X applications, we need to motivate one important assumption: that there are sufficiently many X applications that make this endeavor feasible. Clearly, the well-known X applications such as xclock, xedit, or xcalc would hardly justify our motivation. Upon closer inspection, there are many GUI-libraries that support X Windows. There are several C/C++ based libraries with an X Protocol binding, such as Qt or GTK. On the Java side, Sun Microsystems offers X-based implementations of their

AWT and Swing toolkits. Effectively, every Java application with a GUI is also an X application.

3 Design alternatives

There are several different design alternatives on how to provide universal access to X applications. Before we discuss the various alternatives, we first want to explicitly state the design goals we expect of a general solution:

1. Provides ubiquitous access to any legacy X application.
2. Runs in all major web browsers.
3. Does not require the installation of additional software.
4. Accessible to non-tech savvy users.

Based on these design goals, we discuss various options in designing a possible solution along with their advantages and disadvantages. In the following sections, we discuss the options of installing a local X server, running an X server as an applet, and a HTTP-based solution in detail.

3.1 Local X server

The most obvious solution would be to install a local X server. Several products exist – both commercial and as Open Source – that allow to run an X server on all major platforms including Windows, such as WeirdX, Cygwin/X, or HummingBird Exceed (see [5], [12], [4]). In the most straightforward configuration, the X server is displayed in its own window on the hosting windowing system. However, there are two problems related to this solution: the end-users unwillingness to install additional software as well as firewall issues.

Given a choice, end-users either intentionally or unintentionally choose not to install additional software if they have an already existing solution and the benefits of the alternative are not immediately apparent. As a specific example of the reluctance of end-users to explicitly install software can be seen by the proportion of Windows users who use Internet Explorer, despite security issues compared to other browsers. Internet Explorer currently owns more than 85% of the market share (see [13]), primarily because it is bundled along with Windows. If end-users are reluctant or simply do not bother to use easy-to-install software such as alternate web browsers, they would not be willing to install an X server.

Another issue is related to firewalls that may prevent the X client and X server from communicating with each other, if both are on opposite sides of a firewall. Generally, firewalls block most ports to prevent intruders from accessing and compromising services being provided inside the corporate environment. A few ports that provide essential or popular services (such as HTTP) may be left open. However, the ports over which the X Protocol runs are usually blocked. Also note that the X Protocol essentially reverses the client and server relationship: the X client is establishing a connection to the X server. Since end-users also usually run a firewall on their desktop or DSL-modem, this places an additional burden on running a local X server.

3.2 Applet based solution

Another solution is to implement the X server as a Java applet. This way, the X server can automatically be downloaded into the browser. Once downloaded, the applet functions as an X server that uses the browser's window to render the output of X clients. WeirdX is an Open Source X server that is based on this idea (see [5]). While this approach solves the problem of a users unwillingness to install additional software on his or her local machine, it has some issues of its own.

First of all, firewall issues are not resolved as the applet still acts as a server and would have to communicate in the same manner as a locally installed X server would. In addition, the browser's Virtual Machine (JVM) may prevent the applet from opening ports to listen for connections from X clients, as the JVM would deem this to be a security risk. This can be resolved by configuring some of the JVM parameters but once again, the average end-user would find it unappealing.

Another problem is that the future of applets is unclear with Microsoft phasing out their support for its built-in (and outdated) JVM in Internet Explorer in 2007 (see [7]). End-users would either have to download Java Runtime (JRE) software from Sun Microsystems or use a machine where the JRE is preinstalled. Having to deal with JRE installation and patch updates in addition to the browser's own updates would be an added responsibility for the end-user.

3.3 HTTP-based solution

A solution in which the web browser can view and interact with X clients over HTTP can solve all of the aforementioned issues. The end-user does not need to download any software and does not have to deal with installation or configuration hassles. She can use the existing web browser on her machine or any other machine she uses. Firewalls are usually configured to allow HTTP. Moreover, unlike X servers which are also TCP servers, web browsers are TCP clients. As a result, the client-server relationship is inverted and this essentially resolves the security issues and no additional security configuration is required. Since the end-user is not installing any software, there are no installation restrictions. The solution is not applet based and does not have dependencies on the JVM plug-in. Naturally, this solution introduces some technical challenges, e.g., how to break the symmetry of HTTP in order to support the asynchronous X Protocol. The rest of this paper describes this solution in detail.

4 Architecture

Our solution follows the HTTP-based solution as outlined in the previous section. Figure 2 gives an high-level overview of the architecture. The main component is a protocol bridge we call the *XWeb Broker*. Information that is sent between the X server and X client over the X Protocol is transformed and sent over HTTP

to the web browser by the XWeb broker. Thus, from the X client's point of view, the XWeb broker acts like an X server. Since the X client uses the regular X Protocol to communicate with the XWeb broker, the X client can be used without requiring any modifications to its implementation.

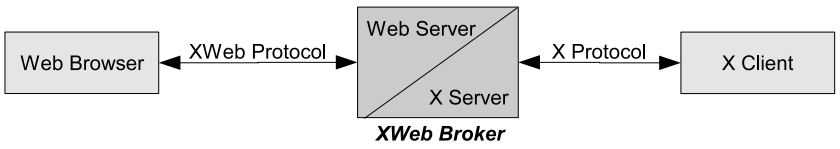

Fig. 2. XWeb overview.

The XWeb broker as shown in Figure 2 acts as a so-called headless X server. This means that while the XWeb broker behaves like an X server, it itself does not render any output and therefore does not require a graphic workstation to run. Instead, the incoming X traffic is transformed and forwarded to the web browser. The web browser is where the visual user interface is rendered. The web browser uses the XWeb protocol to communicate to the XWeb broker. Of course it is not obvious how the XWeb broker can forward information to the web browser, since HTTP only allows the web browser to initiate a request. Our solution to this problem will be discussed in a subsequent section, but first we introduce the informational model.

4.1 Informational Model

The purpose of the informational model is to define a data model that is needed to capture all relevant information for the scope of the XWeb protocol. The informational model of the XWeb protocol is an abstraction of the informational model of the X Protocol. At the core of the X Protocol are *windows* and *panels*. Every X application is contained in its own window. A window can contain multiple panels that form a hierarchy and that are positioned relative to their parent. In Figure 3 the panel `ID_3` is a child of panel `ID_2` which is itself a child of a top-level window. A panel usually represents a widget such as a button where the X client draws the user interface element in its look-and-feel.

The X Protocol offers fine grained drawing operations for lines, rectangles, circles, and other geometric elements. Each of those drawing operations occurs in a panel. Since the X Protocol does not support complex widgets, these have to be drawn manually inside a panel. One option of the XWeb protocol would have been to map the X Protocol elements to equivalent drawing instructions for the web browser. This would result in a one-to-one mapping of X-PDUs to XWeb PDUs. However, this would place a high burden on the client-side protocol engine of the XWeb protocol, since it would need to understand all the different X-PDUs.

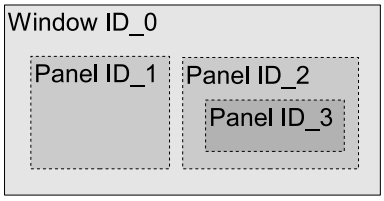

Fig. 3. Window abstraction.

In order to keep the client-side protocol engine as simple as possible (and therefore increase its portability) we limit the informational model to images that will be rendered at the position of their respective panels. This means that the rendering of a panel's content happens inside the XWeb broker, while the web browser only has to load and display the final rendered image. E.g., the X application xlogo draws the X logo using a series of draw instructions. The result will be an image that is captured by the XWeb broker as a PNG (Portable Network Graphics) image.

The informational model also has to include the events that are relevant to the X Protocol. These include mouse events (e.g., mouse move, mouse entered or left a panel, mouse clicked) and key events (e.g., key pressed or released). The informational model has to be mapped to HTML because this is what the web browser is capable of rendering. Windows and panels are mapped to elements which serve as boxing elements in HTML. The hierarchy of panels sown in Figure 3 can be translated to the following HTML:

```
<!-- HTML -->
<div id="ID_0">
  <div id="ID_1" style="position: absolute; top: 10px;left: 5px"
       onMouseOver="..." onKeyPressed="...">
    <img src="..."/>
  </div>
  <div id="ID_2" style="...">
    <img src="..."/>
    <div id="ID_3" style="...">
      <img src="..."/>
    </div>
  </div>
</div>
```

Every element has its own ID as well as further attributes that characterizes it. The image associated with a panel is simply referenced with the <img> tag from HTML. The web browser will automatically load the appropriate image and render it inside the box. We use CSS (Cascading Style Sheets) to position the various elements relative to their parent. Note that despite its name the "`position: absolute`" argument of the `style`-attribute positions

an HTML element relative to its parent. The various events of interests (mouse- and key-events) can be intercepted by registering event-handlers that will invoke appropriate JavaScript functions when they occur.

The HTML shown above only serves as an example on how the informational model is mapped to HTML inside the web browser. Unlike the document-centric usage of HTML where the web browser downloads a complete page and renders it, in XWeb we update the content of the page on a fine-grained basis without loading a complete HTML-page. This can readily be accomplished manipulating the DOM (Document Object Model) representation of the HTML-page using JavaScript. The following JavaScript excerpt shows how a new panel can be created as a child of panel `ID_3`. The various attributes such as CSS-style or event-handlers can be set accordingly using the DOM-API.

```
1 // JavaScript
2 var new_node = document.createElement("div");
3 var parent_node = document.getElementById("ID_3");
4 parent_node.appendChild(new_node);
```

4.2 XWeb Protocol Data Units

The XWeb broker acts as a protocol bridge. While it uses the regular X Protocol to communicate with X applications, we have devised a new protocol for the communication between the XWeb broker and the web browser. We use HTTP as a transport mechanism for the XWeb Protocol Data Units (PDU). The next section will explain how we achieve an asynchronous protocol on top of HTTP, whereas in this section we focus on the structure of the PDUs.

The web browser needs to be able to marshal and unmarshal a PDU in an efficient way. For that reason the XWeb PDUs are based on XML, since all the major web browsers support XML-parsers. Another advantage is that XML-based PDUs can readily be transmitted via HTTP. The following XML shows an XWeb PDU for creating the panel hierarchy shown in Figure 3. This PDU would be sent from the XWeb broker to the web browser:

```
1 <xweb>
2   <create id="ID_0" type="window" />
3   <create id="ID_1" pid="ID_0" type="panel" />
4   <create id="ID_2" pid="ID_0" type="panel" />
5   <create id="ID_3" pid="ID_2" type="panel" />
6 </xweb>
```

An XWeb PDU is an XML-document whose root element is <`xweb`>. This root element can have one or more children. Each of those child elements could have been transported in their own PDU; marshalling several elements into one PDU makes the XWeb protocol more efficient. The client-side protocol engine processes the child elements from top to bottom. In the example shown above,

the four child-elements create one window and three panels. The relationship of IDs (marked by the XML-attribute `id`) to a parent-ID (denoted through the XML-attribute `pid`) determines the nesting of the various panels. Note that up to this point only the hierarchy of the panels has been determined, but not any of their physical attributes such as size and position. The following XWeb PDU demonstrates how these attributes can be defined for panel `ID_1`:

```
<xweb>
  <update id="ID_1">
    <property name="left" value="10" />
    <property name="top" value="10" />
    <property name="width" value="50" />
    <property name="height" value="50" />
  </update>
</xweb>
```

The XML-tag <`update`> allows to alter various attributes of a panel. In the example shown above, the size and position of the panel are set via the <`property`> tag. The reason for using the attributes `name` and `value` (compared to a more compact form such as <`property left="10"/`>) is that new properties can easily be added without requiring to change the schema of an XWeb-PDU. This effectively makes the protocol as well as the protocol engines more robust. Properties can be changed individually at any point in time after the panel has been created. For most applications the size and position will be only set once and then not changed during the lifetime of the application. Defining the geometry of a panel does not say anything about its content. This is done via another property as shown in the following XWeb PDU:

```
<xweb>
  <update id="ID_1">
    <property name="image" value="5.png" />
  </update>
</xweb>
```

The property contains a reference to an image with the name `5.png`. Upon receiving the this PDU, the client-side XWeb protocol engine will create the following HTML: <`img src="/image/5.png"`>. Upon creating this HTML-tag, the browser will then automatically issue another HTTP request to load the image `/image/5.png`. Images are therefore loaded asynchronously which requires the XWeb broker to cache those images. Whenever the content of a panel changes, a new image is created and sent via an update-PDU to the web browser.

The PDUs discussed so far are sent from the XWeb broker to the web browser. Whenever an event arises at the web browser, a PDU has to be sent to the XWeb broker to notify it of the event. With respect to the X Protocol, events of interest are keystrokes and mouse movement. The XWeb protocol features the

XML-tag <event> to describe events. The following XWeb PDU is sent by the web browser to notify the XWeb broker that the mouse has entered the panel with the ID ID_1 at coordinate $(0, 10)$ and while the mouse was inside this panel, the user pressed the key "H":

```
<xweb>
  <event id="ID_1" type="mouseEntered" x="0" y="10"/>
  <event id="ID_1" type="keyPressed" key="H" />
</xweb>
```

Note that some PDUs of the X Protocol have no corresponding PDU in the XWeb protocol. One example is the window-expose-event that is sent every time a part of a window becomes visible. This is necessary in the X Protocol, because the X server does not cache window content. Since web browsers do cache the content of a window, this does not pose a problem and therefore does not require a notification to the XWeb broker — the XWeb protocol has no corresponding PDU. Table 2 gives a summary of all XWeb protocol elements.

PDU	Description
<xweb>	Toplevel XML-tag for every PDU.
<create>	Creates a window or panel.
<update>	Updates one or more properties of a panel.
<delete>	Deletes a window or panel.
<event>	Sends an event to the XWeb broker.
<property>	Sets one property of a panel.

Table 2. XWeb PDUs.

4.3 XWeb Protocol

Figure 4 gives an overview of the XWeb protocol. The interaction begins when the user visits the URL of the XWeb broker (1). Encoded in the URL is also the X application that the user wishes to launch. Upon receiving the initial request, the XWeb broker starts the X application as a separate process (2) and responds to the web browser with the client-side implementation of the XWeb protocol (3). To achieve browser independence, this implementation is based on JavaScript.

After the client-side of the XWeb protocol has been downloaded into the user's web browser, the protocol engine begins pulling updates (4). Upon receiving the request, the XWeb broker defers the reply until there is something to report back to the browser. Meanwhile, the X application will begin to open a window and render its user interface (5). Upon receiving the X window drawing requests from the X client, the XWeb broker converts them into images to be

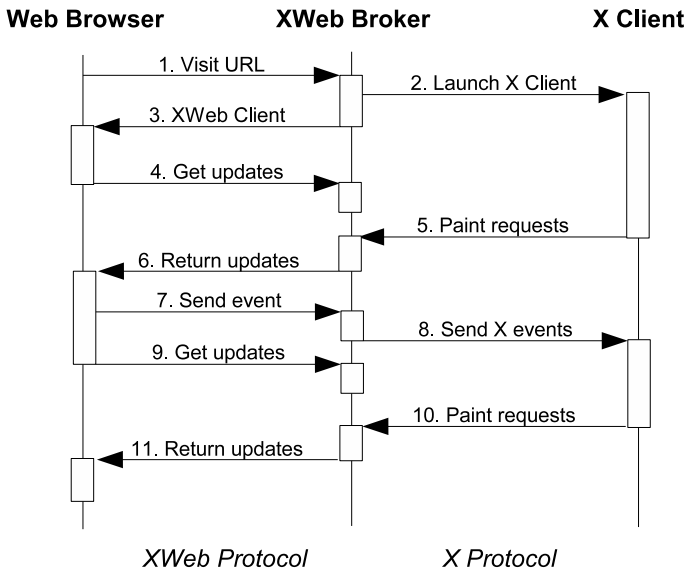

Fig. 4. XWeb Protocol.

sent back to the web browser. It is only then that the XWeb broker sends a response (6).

This technique is referred to as a *deferred response*. The response (6) to a request (4) is deferred until the XWeb broker has some information to send to the browser. This technique is necessary to allow asynchronous updates: the X client can update its user interface at any point in time, but because of the client/server relationship of HTTP, the XWeb broker itself cannot forward those updates to the browser. Those updates are piggy-backed onto the HTTP responses. Note that because the HTTP response is deferred until there is a PDU to be sent to the web browser, this model does not result in busy waiting.

The main event loop of the client-side implementation of the XWeb broker therefore constantly pulls for updates from the XWeb broker (e.g., (4) and (9) in Figure 4). Since the response is deferred until there is something to be sent back, this technique does not revert to busy waiting. In Figure 4, the PDUs 1, 4, 7, and 9 represent HTTP requests, whereas PDUs 3, 6, and 11 represent HTTP responses.

Whenever the user creates an event (such as pressing a key), the XWeb broker is notified of the event (7). In order reduce the number of mouse events, the web browser can be configured not to send every mouse event. Upon receiving an event, the XWeb broker simply forwards it via the appropriate X-PDU to the X client (8). The X client will react to the event by updating the content of a panel (10). These updates are sent back to the web browser via another deferred response (11).

5 Prototype Implementation

We have done a prototype implementation of XWeb. By implementing the XWeb broker, we had to implement the X Window Protocol as well as the XWeb-specific protocol. In order to leverage Open Source tools as much as possible, we use several freely available Open Source packages.

As explained in the previous section, the XWeb-broker acts as an X server towards the X clients connected to it. We use the aforementioned WeirdX. Although WeirdX implements an X server in Java, we had the problem that WeirdX is supposed to run on a desktop and it consequently opens a window wherever it runs. In order to suppress this window and to be able to interface with the output, we have implemented our own AWT Toolkit. By implementing the abstract base class `java.awt.Toolkit`, one can intercept the opening of Windows and the creation of widgets. Setting the property `awt.toolkit` during startup of the Java virtual machine, one can override the build-in AWT toolkit. Luckily WeirdX only makes use of very few AWT classes, so that this solution is both simple and elegant. This allowed us to use WeirdX without any modifications.

The communication between XWeb and the web browser is based on HTTP, which means that the XWeb broker acts as a web server towards the browser. We use the Open Source HTTP engine called Simple (see [2]). As the name implies, Simple is an easy to use, thin web server implementation. The API is modeled after the Servlet specification. Simple is shipped as one JAR file that can easily be linked to other applications such as the XWeb broker in our case.

The client-side of the XWeb protocol is completely implemented in portable JavaScript that runs in all popular browsers. The JavaScript code is loaded into the browser when first visiting the main page served by XWeb through Simple. The user can specify the application to be run via parameters encoded in the URL. E.g., visiting `http://xweb-host.com/XWeb?APP=xcalc` assumes that the XWeb broker is installed on host `xweb-host.com` and it would launch the X application xcalc (an X calculator). Figure 5 shows a screenshot of xcalc running inside Internet Explorer after performing the computation $2*21$. The input focus still shows the mouse on the =-key that was clicked last.

We have conducted some performance measurements to estimate the overhead introduced by XWeb. We have used xedit, a popular X editor, for our experiments. The experiments consisted of three phases: starting xedit, typing "Hello World" once xedit has started, and then exiting xedit. Table 3 shows the total number of transferred bytes and the number of PDUs exchanged for each of the three phases. As can be seen, XWeb does incur a significant overhead. This will become apparent when considering how for example a rectangle is drawn.

The X Protocol has a special instruction for drawing rectangles. This instruction contains the geometry of the rectangle and further attributes such as fill color. The corresponding X-PDU is therefore only a few bytes in size. On the XWeb side, the rectangle is actually transferred as a PNG image, which obviously requires more bandwidth. But despite the higher protocol overhead of XWeb compared to the X Protocol, we found that X applications that run inside a browser via XWeb are sufficiently interactive to be usable. Because of

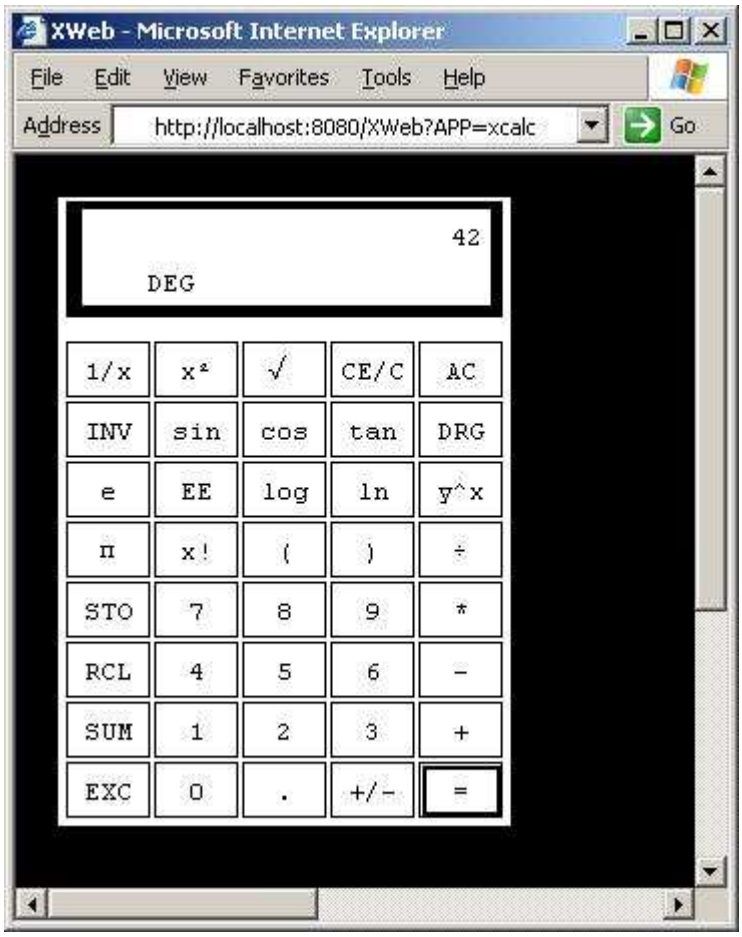

Fig. 5. Running xcalc inside Internet Explorer.

the high-bandwidth of most networks one does not notice any significant delays and we were able to operate a complex IDE this way.

	Total size (in bytes)		Number of PDUs	
	X	**XWeb**	**X**	**XWeb**
Starting xedit	13.388	90.002	56	168
Typing "Hello World"	7.422	61.935	53	197
Exit xedit	0	16.574	0	35

Table 3. Comparison of protocol overhead.

6 Related work

We are not aware of any other project that has built a X-to-web protocol bridge. However, there is a different way how our work can be interpreted, which has to do with the way we integrated WeirdX. Recall from the previous section that we leverage the Open Source X server implementation WeirdX by providing our own AWT Toolkit. The benefit of this approach is that we do not need to make any modifications to WeirdX which certainly has benefits whenever a new version of WeirdX is released. What facilitated our approach is the fact that WeirdX only makes use of two AWT container classes to render its output: `java.awt.Window` and `java.awt.Panel` for which our AWT toolkit provides a replacement implementation.

The consequence is that any Java AWT application that only uses those two classes could be exposed as a web application via XWeb. WeirdX could just be seen as one of those applications. Of course, as soon as an application uses a different AWT class such as `java.awt.Button`, our current XWeb prototype would not work. But it is conceivable to extend the XWeb protocol in such a way that it also supports other widgets such as buttons or listboxes. The web browser as a generic client could provide native support for these widgets. The XWeb protocol would thus support different widget types such as the following:

```
1 <xweb>
2   <create id="ID_2" pid="ID_1" type="button" />
3   <create id="ID_3" pid="ID_1" type="listbox" />
4 </xweb>
```

This path leads to a general application migration framework where any AWT or Swing application could be exposed as a web application (see Figure 6). Several projects – commercial and Open Source – exist that aim at providing an easy migration path for legacy Java applications to web applications. WebCream is a commercial product by a company called CreamTec (see [1]). They have specialized in providing AWT and Swing replacements that render

the interface of the Java application inside of a web browser. WebCream makes use of proprietary features of Microsoft's Internet Explorer and therefore only runs inside this browser.

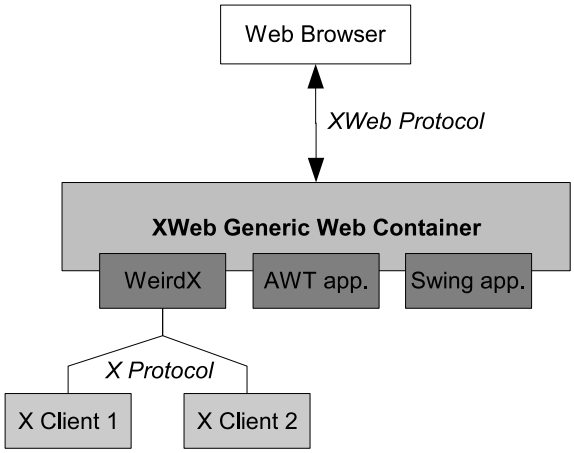

Fig. 6. XWeb Generic Web Container.

Two Open Source projects, both hosted at SourceForge, follow the same idea of exposing Java desktop applications as web applications. The first one is called WebOnSwing (see [9]). Unlike WebCream, this project is not tailored for a particular browser. One feature offered by WebOnSwing are templates that allow to change the look-and-feel of the application that is rendered inside the browser. Another project with similar features, but not quite as mature, is SwingWeb (see [6]).

Following our earlier argument, it should be possible to run WeirdX as an application in those three projects and achieve the same results as with our XWeb. Our experiments however revealed that this is not possible because none of those projects supports asynchronous updates. In all cases, updates only happen when the user interacts with the user interface, e.g., by pressing a button. An application, such as xclock, that produces asynchronous updates, do not update the user interface in any of the three projects mentioned in this section.

7 Conclusions and Outlook

The X Protocol, developed in 1984, has led to a wealth of X applications. E.g., through the Motif library, every Java application is effectively also an X application. While web browsers are ubiquitous nowadays, the same cannot be said of X servers. This paper introduces a migration path that allows access to any

X application via any web browser. This can be accomplished without any special browser plugins. Our prototype implementation leverages Open Source tools where possible to implement the protocol bridge efficiently.

XWeb can be considered as a generic web container that could potentially allow one to host any Java Swing or AWT application. With additional work, a comprehensive set of implementations for the other Swing and AWT widgets could be provided. This would make any Java application accessible from any web browser again without the need for special browser plugins. It might also be worth-while to consider different end-user devices such as PDAs. A PDA would thus become the generic client introduced earlier in the paper. The XWeb protocol would eventually become a client/server protocol decoupling different technologies in the same way that the X Protocol decouples the X server from the X client. We have begun this work as outlined in [11].

This framework leads to support for Ajax-applications. Ajax (Asynchronous JavaScript and XML, see [3]) applications run inside a web browser and achieve an interactiveness that resembles that of desktop applications. An example of an Ajax application is Google Maps that is entirely implemented in JavaScript. Ajax applications execute part of the application inside the web browser. We are currently investigating to add a code migration framework to the XWeb protocol that is capable of translating Java-byte code instructions to JavaScript (see [10]).

References

1. CreamTec, LLC. *WebCream*. http://www.creamtec.com/webcream/.
2. Niall Gallagher. *Simple - A Java HTTP engine*. http://sourceforge.net/projects/simpleweb/.
3. Jesse Garrett. *Ajax: A New Approach to Web Applications*. http://www.adaptivepath.com/publications/essays/archives/000385.php.
4. Hummingbird Ltd. *Exceed - X Server Support for Microsoft Windows*. http://www.hummingbird.com/products/nc/exceed/.
5. JCraft. *WeirdX - Pure Java X Window System Server*. http://www.jcraft.com/weirdx/.
6. Tiong Hiang Lee. *SwingWeb*. http://swingweb.sourceforge.net/swingweb/.
7. Microsoft Corporation. *Microsoft Java Virtual Machine Support*. http://www.microsoft.com/mscorp/java/.
8. The Open Group. *X Window System (X11R6) Protocol*, 1999.
9. Fernando Petrola. *WebOnSwing*. http://webonswing.sourceforge.net/xoops/.
10. Arno Puder. An XML-based Cross-Language Framework. In *DOA*, LNCS, Agia Napa, Cyprus, 2005. Springer.
11. Arno Puder. XML11 - An Abstract Windowing Protocol. *PPPJ*, 2005.
12. RedHat. *Cygwin/X - A Free X Server for Microsoft Windows*. http://www.cygwin.com/xfree/.
13. WebSideStory. *U.S. Browser Usage Share*. http://www.websidestory.com/.