Thin Clients Via Shadow Objects

A.K.M. Ashikur Rahman
Department of Computing Science, University of Alberta
Edmonton, Alberta, CANADA T6G 2E8
{ashikur}@cs.ualberta.ca

Abstract

In this paper, an approach of distributed presentation architecture of thin client computing has been proposed. This approach tries to achieve its goal via object reconstruction. Each widget will have two instances, one is its shadow on the server side, which does not have any graphical presentation and the other is on the client side with full graphical representation. This approach tries to combine two other traditional approaches, one is web-based approach and the other is remote frame buffer approach. In our proposed approach, it is expected that the network traffic will be less than the remote frame buffer approach but higher than the web-based approach. For small wireless devices, this approach may be considered as a viable solution to thin client computing.

1 Introduction

Thin clients are zero-administration or low-administration devices that depend on the network to provide most or all of the processing. The fundamental idea of thin client computing is, instead of running applications locally in traditional manner, applications run centrally on the server and only the user interface is displayed on the client side. The client side and the server side applications interact each other with minimal possible bandwidth requirement. The server may be connected with other data servers, mail servers etc. to provide extraneous services to the client side. In its simplest form, the thin client would consist of a means of input and a display that is updated from the wireless/wired network.

In computer's early days, there were the thinnest clients. Computer terminals were heavy in terms of weight, yet light in locally resident software. Early phones were semi portable devices that were heavy in weight but light in functionality. Today, phones and computers are lightweight, but heavy in terms of software and functionality, no longer they are considered as thin clients. The Personal Digital Assistant (PDA) is considered to be thin but carries 8MB to 24 MB storage to hold phone numbers, appointments and notepads [5]. The cell phone is physically thin but functionally thick. It may include more than one line, web accessibility, caller ID and possibly even interactive map display [5].

Although the thin client model is some sort of client/server model, but the traditional client-server model is based on reliable networks that is not directly applicable for thin clients. Several extensions have been introduced to the client/server model. One of possibility is from the proxy approach, i.e., adding a proxy between the client and the server on either side or both [8]. Also another solution to weak connectivity is proposed in [10] using Coda file system. Other approaches considered using mobile agents as an adaptive computing model and a disconnected-friendly transport. However most of these extensions are either application specific, requiring modifications to pre-existing applications or needs adding powerful caching capabilities thereby making thicker clients.

2 Thin Client Architecture

2.1 Classifications of thin Client Architecture

Based on functionalities, any traditional application may be decomposed into three layers:

`Presentation Layer` The purpose of this layer is to present user interface [1] of the application to the user and mostly to handle user interactions.

`Application Layer` This layer is responsible for the application's computations and logic implementations.

`Data Layer` This layer is responsible for maintaining and supplying the required data of the application.

Each of these layers is then assigned to a machine or a set of machines. A machine or set of machines, which is responsible for a specific layer implementation, constitutes a tier of the architecture. Most popular architecture is a two-tier architecture. In a typical two-tier architecture for example, presentation and application layers might be assigned to the client and the data layer is assigned to a central database server.

Most of the thin client architectures are based on this two-tier client/server architecture, composed by one

[1] mainly graphical user interface

client tier and one server tier. The layers of an application can be assigned either to the client tier or to the server tier. This results in five base architectures:

`Distributed Presentation:` In this architecture, the task of application and data layer is assigned to the server. The task of the presentation layer is distributed between the client tier and the server tier. Most of the processing is done on the server side. The client is merely responsible for displaying graphical user interface (GUI) image of the application and forwards user interactions with the GUI such as keystrokes and mouse clicks to the server. This architecture causes a little processing power to be incorporated on the client side. On the other hand, this causes ample network traffic between client and server. Examples of this architecture are windows based terminals connected to a terminal server or X window.

`Remote Presentation:` In this architecture the application and data layer is assigned to the server. The presentation layer as a whole assigned to the client. Client is capable of managing the entire user interface, as a result all user interactions such as mouse move, keystroke or syntactical checking of data input are handled in the client side and need not to be transferred to the server. Certainly, this architecture requires higher client resources, capabilities and less amount of network traffic then the distributed presentation. Examples of remote presentation are web browsers displaying HTML pages fetched from web servers.

`Distributed Logic:` Here presentation layer is implemented on the client side and the data layer is implemented entirely on the server side. The application layer is distributed between both server and client. The components of the application layer communicate with each other through messages.

`Remote Data Management:` The client implements both presentation and application layer and contacts the server which manages application data. This is actually fat client architecture. Traditional fat client enterprise applications are based on this architecture. Examples are common office software such as word processor or e-mail client.

`Distributed Data Management:` In this architecture, the presentation and application layer are entirely assigned to the client. Moreover the data layer is distributed between client and server. This requires guaranteeing data consistency between server and client.

Figure 1 provides a cross section of the divisions of three layers between client and server of the five base architectures. From the thin client perspective, the first two architectures, i.e. the distributed presentation and remote presentation architecture are most popular and heavily used architecture. We will now discuss three approaches of thin client architecture, which fall under these two client/server category.

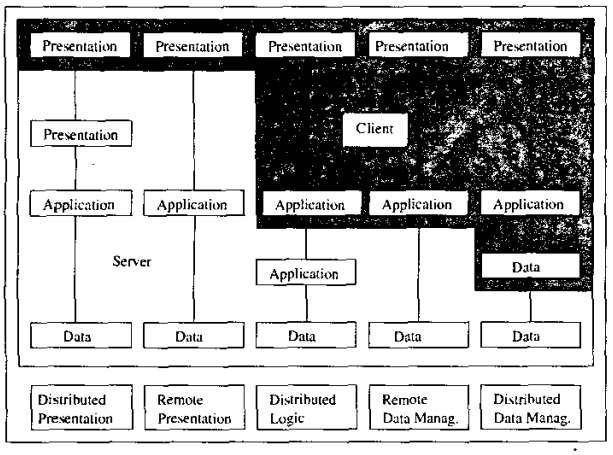

Figure 1: Two-tier based Client/Server architecture

2.2 Web Based Approaches

The web based approach falls under the category of remote presentation architecture. The client is responsible for implementing presentation layer. The application and data layer reside on server. This approach, in most cases, is based on Hyper Text Transfer Protocol (HTTP) and Hyper Text Mark Up Language (HTML). The basic idea is as follows: The presentation layer is implemented using HTML and the application logic is written with a special web API, i.e., CGI script, ASP, PHP or JSP. The GUI to be displayed on the client-side is written using HTML, which can be interpreted by a browser on the client side. The browser communicates with the server using HTTP to get the HTML content. Once the initial web content is displayed on the browser, which also handles all the user interactions locally, the user puts the values on the widgets and press a submit button. Then browser contacts with the server using HTTP supplying the user input with get/post method invocation. The server executes the application logic (i.e. executes a CGI script) and sends the response of the application (which is merely some more HTML code) back to the client. The Web browser interprets the HTML and renders the GUI on the Client buffer for display. Figure 2(a) shows this kind of architecture.

There are several drawbacks in the web-based approach using HTTP/HTML. The widgets available in HTML are very basic. Secondly, browser must adhere to a page-oriented paradigm. Each time, server processed the client's request and replied with updated GUI, the browser has to render the HTML-page code from the scratch. This causes a heavy load on the client side.

Several attempts have been taken to solve these problems. One approach is to send the entire binary executables of the applications over HTTP (i.e. Java applets). But this also concerns security [11]. Also sending java

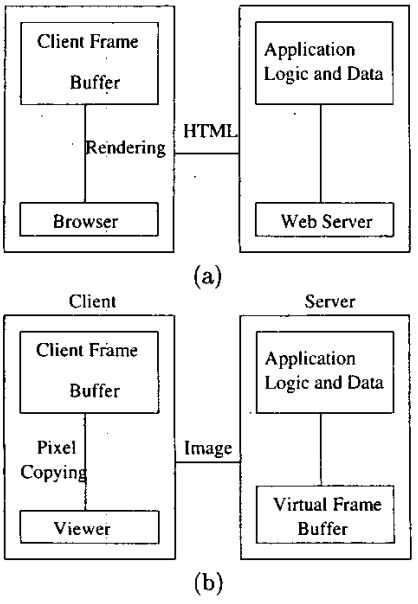

Figure 2: Architecture of (a) the Web Based Approach, (b) the Remote Frame Buffer Based Approach

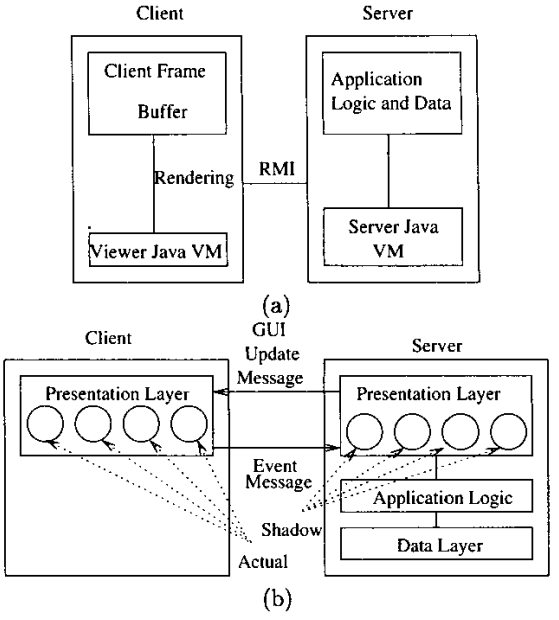

Figure 3: Architecture of (a) the Distributed User Interface Toolkit Based Approach, (b) the Object Reconstruction Based Approach

applets means sending a part of application logic to the clients, which will cause the client to become fat instead of, thin. Another approach is to create and incorporate separate browser plug-ins to support own languages to provide rich user experience [11] (i.e. Macromedia Flash and Shockwave). But this also needs extra heavy computing capability on the client side.

2.3 Remote Frame Buffer Based Approaches

This approach falls under the category of distributed presentation architecture. In this architecture, only a tiny part of the presentation layer is run on the client. The remaining presentation layer parts as well as the application and data layers completely run on the server. The client merely displays the graphical representation of the GUI of an application on the client. One simple way to implement this is to create a virtual frame buffer in the RAM of the server on which the application can render it's GUI and then transporting the resulting raster image to the client. Client displays the resulting image in the client's viewer space and forwards all user interactions such as mouse clicks, keystrokes back to the server. The server will send the user interaction to the corresponding application, which in turn will render new display content, based on user input into the virtual frame buffer. Server will then send the new raster image of the display content to the client for update. This procedure will continue until the process terminates. Figure 2(b) shows the architecture of this kind of thin client.

There are a numerous examples of thin client products adopting remote frame buffer based approach. Products such as Critix Metaframe [1], Tarantella [4], Graphon RapidX [2], VNC, X server are among those to mention.

Certainly this approach can provide rich user interface experience thereby overcomes the drawback of limited widget collections of web-based approach. This approach also has several disadvantages to mention. One serious draw back is its higher bandwidth requirements. As it is transferring the entering display content in the form of image from remote frame buffer of the server to the client, definitely this will cause a lots of network traffic to flow. A second drawback is its slow response time compared to the web based approach. Here every user interactions are transferred to the server over the network, which causes a time to be lapsed to get a response back from the server. Although the use of lossy compression algorithms (e.g. MPEG) are available, but none of the existing systems has employed such techniques. Moreover, if the graphical interface of the application contains text then the usability of any lossy compression technique will become questionable.

2.4 Distributed User Interface Toolkit Based Approach

Another interesting approach to thin client computing has been proposed in [11], which is called distributed graphical user interface approach. In this approach the

server can create, modify and delete any of the components available in the distributed toolkit on the client side as if it were working with a local application. Instead of sending pixel data rendered on the server across the network, this approach sends the semantics necessary to render that pixel data on the client. All the user interactions are handled locally on the client side thereby reducing the perceived latency. Roughly this toolkit is based on JFC toolkit and they have named it RJFC (Remote JFC) toolkit.

The underlying architecture of this approach is shown in Figure 3(a). The base of this architecture is the support of Remote Method Invocation (RMI) that is a very popular feature in Java. The application code is interpreted in a server Java virtual machine. When the server Java VM finds a code to instantiate, modify or delete a graphical user component (in this case a remote JFC component), it transparently informs the remote JFC viewer running on the client side Java VM using remote method invocation (RMI). The remote JFC viewer then reacts to this by performing the exact same action on the viewer. For example, if the server requests that a new remote "JButton" to be created, the server Java VM would transmit that command to the viewer Java VM on the client side using RMI. The viewer Java VM then creates a "JButton" using the standard JFC API, thus causing the actual button to be rendered on the client. The event handling occurs in the same manner but in the opposite direction. For example, if someone clicks on a "JButton", The Viewer Java VM transmits this to the server event handler via RMI and actual event handling code is executed on the server side.

This approach causes a significant less amount of network traffic to be transferred than that of remote frame buffer approach. Also the client side viewer size is small compared to web based approach browser. But size of the viewer is obviously larger than the remote-buffer-based approach viewer.

A major drawback of this approach is its dependency on Remote Method Invocation. The Remote Method Invocation is not supported in every language. In particular, this feature is very specific in Java language [3]. RMI support also requires higher capabilities on the client side than any other approach.

3 Thin Clients Via Object Reconstruction

In our approach to thin client computing we have considered to build up thin clients in small wireless devices such as cell phones, PDA etc. For that purpose we have solely employed Java 2 Micro Editions (J2ME), which is specially designed for building applications on these small wireless devices. In particular we have used Sun's reference implementation of MIDP and CLDC.

3.1 The Prototype

Our approach falls under the category of distributed presentation architecture mentioned in Section 2.1. The application layer and the data layer has been implemented in the server side. The presentation layer is implemented in both the server and client side.

In this approach an application written in J2ME runs as a server process serving many clients. On the client side an application written also in J2ME constituting a small part of the presentation logic connects to the server process to get the information about the graphical user interface. Now let us describe how the presentation layer is implemented by employing the server and the client at the same time.

Every widget that needs to be displayed for a particular application is created on both sides. The widget will be created on the server memory space first but it will not be displayed on the server side. Instead, the server will transmit a message specifying all the widget properties to the client to create the same widget into its own memory space. For example, if the application needs to create a text field with specific label, width and height then, server creates the text field into its own memory space first then transmits a message with all the properties of the text field (i.e., label, text to be displayed inside the text field, maximum size and constraint information) to the client process. Client then reconstructs the same text field with provided information into its own memory space and displays the text field into its own screen. That is why this approach is called 'object-reconstruction-based approach'.

In this way, every widget object has two instances of it. One is on the server's memory space, which is created by the server process but not displayed. The other is on the client's memory space, which is reconstructed by the client viewer and also displayed on the client screen. As the server side instance of the widget does not have any graphical representation we can think it as a shadow of the widget created on client side and may call it 'shadow' widget. All the user interface development is basically based on these 'shadow' objects.

Whole application logic is executed on the server side. The objects other than the widgets (i.e. text field, button) have only single instance and resides on the server's memory space. Because of the fact that the necessary computations is done by the server, the objects other than the widgets need not to be duplicated. Also all data will reside on the server side. Thereby, this approach minimizes the memory requirement[2] on the client side by causing only those objects to be created on the client side which are used for display purpose.

Events are generated asynchronously by the users on the client side. The client viewer registers event listener

[2]Note that due to shadow objects, the server's memory consumption will be little bit higher but server side is expected to be capable of bearing high load

for each of the widgets but does not execute any of the event handler code. Rather, once the events are generated, it sends simple messages containing the widget information that caused the event to the server. Server posseses all the event handling code for each of the widgets and executes them accordingly.

Display update mechanism is very simple. At application start-up, client will connect to the server process. Server will send a description of visible user interface as implemented with the shadow objects. Based on this information client creates the widgets and builds the overall user interface. Now usually the graphical user interface changes its state after the user interacted with the application through the GUI. When an event is generated, the client immediately informs server about the event and the corresponding widget object causing the event. The server will transmit the updated user interface at that time if needed. Figure 3(b) shows the schematic of the object-reconstruction-based thin client prototype.

3.2 The Protocol

The protocol for transmitting GUI update messages from server to client and the event messages from client to server is pretty straightforward. There may be exactly three cases associated to each widget object. A widget object may be 1) instantiated, 2) modified or 3) deleted. Whenever a widget object is created on the server's memory space, we are assigning an id number corresponding to the widget object. Then the server process immediately transfers a message containing type of the widget, the widgets id number and the property information of the widget. From this information, client will create the same widget object with the same property, assign it the same id number as in server, include it to the corresponding container and display the widget object. The deletion is handled in the same manner. However, modification of widget objects is little bit trickier. There may be several properties associated to each widget object and the server code may modify one or few properties of a particular widget object at a time. Also there are different methods to modify different properties of each widget object. But we cannot incorporate all the modification methods in the client side. For example, a text field has several property values such as text field label, text to be displayed inside the text field, maximum size and constraint information. Now there are four methods associated with a text field to change any of its property namely, *setLabel*, *setString*, *setMaxsize*, *setConstraint*. Now instead of supporting each modification method on the client side, we will let the server to modify the property of the text field using desired method invocation and then transmit all the property fields to the client to instantiate a new widget objects with modified properties. The client will create a new text field, remove the old one and add

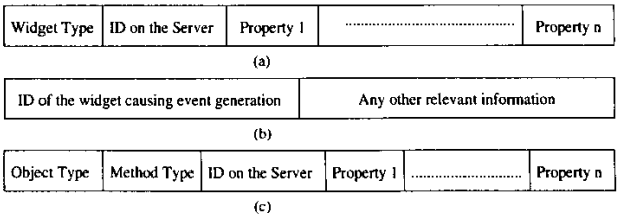

Figure 4: Format of (a) the message to create a new widget object, (b) the event message, (c) the message to support irrecoverable objects

the new one but the id number of the new one will be same as the old one thereby keeping consistency in object id between the server and the client. In this way the modification is also supported in almost same manner as creating a new one.The format of the message for creating a new widget object is shown in Figure 4(a). The event generation message flows from the client process to the server process. For each widget, the client viewer has a corresponding event listener method. Whenever there is an event generated to a widget by the user, the client viewer immediately transmits the event message to the server. The event message contains the id of the widget that caused the event and any other required information such as text typed in a text field by the user or the item selected by a user from a list. If it is a simple command button that is pressed by the user then the client viewer will simply send the id number of the corresponding command button. The format of the event message is shown in Figure 4(b).

There is another type of message that can be transferred from server to client. Not all objects can be reconstructed from its property values. J2ME does not necessarily provide all the property value of each object after it's creation, for example the graphical objects. If a user want to draw a line in graphics object then some complicacy arises using our approach. After drawing a line in a graphics object, there is no way to retrieve information about the line from the graphics object properties. Thereby at the time of drawing into a graphics object in the server side, the server immediately sends a message to the client to invoke a similar kind of method to draw in the corresponding graphics object on the client side. The parameters of the method are passed in the body of the messages. In this way we are handling those special objects, which cannot be reconstructed from the property values after creation. Figure 4(c) shows the message format for such unrecoverable objects.

3.3 The Viewer on the Client Side

The size of the viewer on the client side is very small (22.1 KB). In a Web-based client the client viewer needs to interpret the HTML, find out the HTML widgets

and then translate those widgets into a J2ME representative widget. This will definitely cause the client viewer size to increase because here client needs to interpret the code and then create the widgets. But in our approach the viewer is only creating the widgets, the application code is interpreted on the server side, only thing the client needs to interpret are some messages sent from server to the client viewer. Also the event handling code is executed in the server side. The client viewer only needs to record the widget ID causing the events and inform the server to take appropriate actions. Hence client viewer code will be considerably small. On the other hand, with respect to remote frame buffer approach, the client viewer size in our approach will be larger. This is because in remote frame buffer approach the viewer only needs to display an image. All required rendering of widgets are already done on the server side and converted into pixel data. But in our approach, the client viewer is also rendering each widget into its own view space. Instead of image data, a higher-level semantics to create widgets are transferred in the form of message from server to client. That's why the viewer size will be larger than the remote frame buffer approach. As a rough estimate, the client viewer size in our approach will be in between the two approaches.

On the other hand, in terms of network traffic, certainly our approach will require less amount of data transfer between server and client than that of remote frame buffer approach. In remote frame buffer approach, the entire display content in the form of image is transmitted from the server buffer to view in the client's screen. Each time when a display needs to be updated, the server needs to render again all new widgets into its frame buffer and transmit the pixel data over the network. But in our approach, we are only sending higher-level semantics to render widgets in the client side. So actual rendered data is not transmitted over the network, instead only the required properties to render a widget are transmitted over the network. Hence bandwidth requirement will be less than the remote frame buffer approach.

4 Conclusions

An Object reconstruction based approach to thin client computing has been proposed. Few points need to be mentioned here about our approach. First of all, web-based solution is not directly feasible for small wireless devices. One reason for that is the small display screen of these wireless devices. It is very difficult to view any web content entirely in this small display screen. We can employ some scrolling mechanism on the client viewer but that may cause user frustrations by forcing them to scroll many times to view the entire content. Also not every widgets written using HTML can be directly viewed without any middleware transformations. As for example, there is no way to display a button in HTML directly in the viewer. The viewer needs to transform it into some 'Command' type object and then can display in its own form. Also the remote-frame-buffer-based solution cannot be applied using J2ME. There is no way to define a remote buffer for the application running on the server for its GUI rendering purpose. Thereby it is not possible to draw off screen images in the server side and then transmit to the client for viewing purpose. The third approach called distributed user interface toolkit approach is also not applicable in these small wireless devices simply because J2ME does not support any remote invocation (RMI) mechanism. For that reason, our approach is a good approach to achieve thinner client solution in small wireless devices.

References

[1] Critix metaframe. http://www.critix.com/products.

[2] Graphon radix. http://www.graphon.com.

[3] Java documentation. http://www.java.sun.com/.

[4] Sco tarantella. http://www.tarantella.sco.com.

[5] Thin is in. http://www.geoplace.com/.

[6] Aksoy Cumhur and Helal Sumi. Optimizing thin clients for wireless active-media applications, 2003. University of Florida.

[7] Alexis Gutzman. Wireless graphics: Why a picture may be worth 1000 words. *Article from mcommercetimes.com.*

[8] A. Helal and S. Ramamaurthy. Client-server computing in mobile environments. *ACM computing Surveys*, 1999.

[9] A. Helal and S. Ramamaurthy. Optimizing thin clients for wireless computing via localization of keyboard activities. In *The International Performance, Computing and Communication Conference*, April 2001.

[10] Ebling Maria R. Mummert Lily B. and Satyanarayanan. Exploiting weak connectivity for mobile file access. *SIGOPS*, 1995.

[11] William M. Chiong Simon Lok, Steven K. Feiner and Yoav J. Hirsch. A graphical user interface toolkit approach to thin client computing. *ACM 1-58113-449-5/02/0005*, May 2002.

[12] Starner Thad. Technology trend favor thick clients for user-carried wireless devices, October 2001. Georgia Institute of Technology.