RC23719 (W0509-075) September 21, 2005
Computer Science

IBM Research Report

Inverted Browser: A Novel Approach towards Display Symbiosis

Mandayam Raghunath, Nishkam Ravi, Marcel C. Rosu, Chandra Narayanaswami
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Inverted Browser: A Novel Approach towards Display Symbiosis

Mandayam Raghunath, Nishkam Ravi*, Marcel C Rosu, Chandra Narayanaswami
IBM TJ Watson Research Center
*{mtr, rosu, chandras}@us.ibm.com, *nravi@paul.rutgers.edu*

Abstract

In this paper we introduce the Inverted Browser, *a novel approach to enable mobile users to view content from their personal devices on public displays. The* Inverted Browser *is a network service to start and control a browser that is then used to view the content. In contrast to a traditional Web browser, which runs on the client device and* pulls *content from a server, content is* pushed *to the* Inverted Browser *from a personal data source upon user input. This approach allows a wide variety of personal content to be viewed by facilitating symbiotic relationships between mobile devices and intelligent displays in the environment. In addition, this approach exploits the significant investments in Web browsers and related technologies.*

Our initial Inverted Browser *prototype is based on a Web Services wrapper around a traditional Web browser. We compare the* Inverted Browser *with alternative approaches based on VNC thin client technology and Bluetooth profiles.*

Our experiments show that the Inverted Browser *approach is superior in terms of user convenience, ease of use, energy consumption, and privacy. We expect hardware improvements to further reduce interaction latencies, an aspect where the other solutions presently have a slight edge.*

1. Introduction

Mobile devices, such as smart phones and portable music players, are becoming the preferred storage device for personal data, mainly due to their constant availability, wireless connectivity, data capturing abilities, and increasing amounts of storage they incorporate. However, their small screens coupled with lack of software to view several types of content limit the user's ability to interact with his personal data. At the same time, it is becoming clear that large displays hold enormous promise as a pervasive technology due their increasing availability in public spaces. Furthermore, many of the newer TV displays feature digital input capabilities and they can be directly attached to PCs or incorporate comparable processing and networking capabilities. For instance, some TVs, such as the Sharp Aquos B3, have the requisite hardware and software capabilities to directly show digital images stored on portable storage media, such as SD, xD and CF cards. In addition to displays, there is a trend towards adding more functionality to projectors. We are beginning to see projectors that support wired and wireless connectivity primarily for network management. Some of the newer projectors also accept portable storage media, including USB flash keys, and have built-in viewers to show PowerPoint files. Our objective is to take this trend one step further and elevate display devices to intelligent first class network entities that offer display services.

Several questions arise upon closer examination of the functions necessary to establish relationships between mobile and public display devices. What minimal software does the user have to carry on the mobile device to be able to interact with the displays? How does this software work in heterogeneous environments? What software stack is necessary on the display and how can the software investment for the displays be minimized? As such displays proliferate in public spaces, how can one reduce the expense associated with the management of such displays?

In an earlier paper [9], we outlined a vision of intelligent display devices offering display services to mobile devices. Essentially, intelligent displays are devices that are capable of displaying content in a variety of different formats and that are accessible to a large variety of mobile devices. Enabling a symbiotic relationship between mobile devices and such intelligent environmental displays that allows users to view and to interact with content from mobile devices is the subject of this paper.

The *Inverted Browser* is our scheme for implementing such intelligent display devices and for enabling symbiotic relationships between display and

personal devices. The *Inverted Browser* is a network service used by mobile users to push content from the personal device to the display and to control a browser-based viewer on the environmental display. Our approach exploits the significant investments made in Web browsers and related technologies. Similar to the widely-deployed web servers, delivering content to traditional Web browsers, we envision future public spaces being populated with intelligent displays running browser-like viewers, which are controlled using mobile devices over wireless connections.

Our initial prototype of the *Inverted Browser* display service uses existing technologies, such as Web browsers, Web Services, and Open Services Gateway initiative (OSGi) middleware[5]. Web Services are an emerging standard for distributed computing tasks, typically used for information exchange between businesses. More recently Web Services are being used to accomplish tasks in pervasive computing scenarios [8]. OSGi middleware simplifies remote management and on demand software provisioning.

The *Inverted Browser* service is implemented as a Web Service that is offered by intelligent displays. Mobile devices installed with the corresponding client software are able to use the services offered by such displays. Both the display and client side Web Services software are based on an OSGi-compatible service platform. Besides personal content, the mobile device stores data used by the service to configure the browser, such as user preferences or credentials for accessing remote servers such as web sites.

As an alternative to the *Inverted Browser* display service, we also considered leveraging existing off-the-shelf solutions to accomplish the same end goal of viewing content from mobile devices. The approaches we considered were based on using thin client technologies, such as VNC, and Bluetooth profiles.

VNC is a popular thin client solution used to export display and the UI controls of a server machine to a remote and typically less capable machine (the thin client). In one alternative approach, we run the VNC client on the mobile device to control the server machine, which is connected to the large display.

The other alternative is a combination of Bluetooth file transfer profile and the Bluetooth Human Interface Device (HID) profile. The file transfer profile is used to transfer the content that the user wishes to view and a HID-like profile is used to interact with the displayed content using the controls on the mobile device. We compare the Inverted Browser approach with the VNC and the Bluetooth approaches and report our qualitative and quantitative comparisons.

The key contribution of this paper is a software architecture for intelligent network displays that leverages existing communication protocols, middleware, and application software. The architecture accommodates independent evolution of each of the existing components, and is also tailored to the differences in resource constraints between environmental devices and mobile devices. In addition, we also address the complexity and costs involved in the deployment of a new class of intelligent network display devices.

2. Inverted Browser

The *Inverted Browser* is a software-only approach for creating intelligent displays that can establish symbiotic relationships with a wide variety of mobile devices. The purpose of establishing the symbiotic relationship is to use intelligent display capabilities to overcome mobile device limitations and, as result, to enhance mobile user ability to interact with data stored on their personal devices.

Our approach is built around the Web browser because of its widespread availability as client application and versatility in terms of content formats that it can handle. As a result, our approach leverages the huge investment that has gone into the development of the existing Web browsers and related technologies. This level of investment is responsible for the browser's ability to render a wide range of content types, as well as the support for plug-in APIs that enable the addition of viewers for several other non-native content formats. In addition, there has also been a significant level of investment by third parties who have developed large number of browser plug-ins.

Typically, browsers run on the client side and content resides on web servers. By invoking a browser, content can be fetched from remote servers over HTTP and displayed on the client side. To enable the interaction between public displays and mobile devices, we adopt this simple and successful approach with one key difference - we invert it.

The *Inverted Browser* is designed as a browser-based application running on the intelligent display device and remotely accessible as a network service. Mobile users of the service push commands and content from their personal devices to the display. However, browsers do not operate in a mode where some other device pushes content to the browser for display or where they accept commands from a remote device, such as from the mobile user's smart phone. Therefore, we had to augment the browser to accept incoming connections, and to display content and

processes the commands received on such a connection. A push-based interaction model, in which connections are initiated, and commands and data are sent by the mobile device to the display is more appropriate than a pull-based interaction model, since mobile devices may be protected by (provider-managed) firewalls or behind NAT devices, such as wireless access points.

To allow for various link- and transport-level networking technologies, network-level details of the display service are not defined. Typical display and mobile devices feature TCP/IP connectivity, in which case the lower layers of the display network service are identical to those of existing service, such as FTP or HTTP. Overall, we expect the service to be accessible over a variety of connections, such as 802.11, Bluetooth or IrDA.

Furthermore, to enable enhanced remote management features on both display and mobile devices, we expect their software stacks to be built on middleware which allows for dynamic provisioning, such as OSGi. The *Inverted Browser* approach leverages the dynamic provisioning capabilities of mobile devices to download the client components most appropriate for the capabilities of the surrounding displays. The role played by this feature in the initial prototype is explained later in the section.

To get a better understanding of the challenges of this approach and to determine level of optimization required for the display and client software stacks, we built an initial prototype of the *Inverted Browser* service. In this prototype, we do not directly modify the Web browser code to make it listen for incoming data. Instead, we wrap the browser code with a thin layer of software. In addition to reducing the implementation effort, this allows the browser itself to be upgraded as newer versions become available. Our wrapper listens for incoming display requests, accepts them and uses the installed browser to display the content. The wrapper queries the browser for its supported content types and uses this information to negotiate content formats with the client devices requesting display services. In the remainder of this section, we describe the *Inverted Browser* functionality, its first prototype and communication protocol.

2.1 *Inverted Browser* Capabilities

The *Inverted Browser* service passively waits for a first access from a mobile device. For TCP/IP-enabled devices, this translates into the service listening to a predefined TCP port for incoming connections. A device that accesses an idle display gets a description of the display capabilities upon the first access, i.e., immediately after the TCP connection is established. This device becomes the session owner. Subsequent accesses by other mobile devices are accepted only if authorized by the owner, which uses its already established symbiotic relationship to grant or deny access. Only the owner can download content on the display device; the other devices, if any, can only interact with the downloaded content. Typical scenarios involving multiple mobile users include the group review of documents and playing an applet implementation of a game, such as chess.

Inverted Browser commands are sent by the mobile device(s) to the intelligent display device. The most important commands are described next.

First, there are the commands to display various types of content, followed by the actual content. Only one such command is active at a given time, i.e., content sent in one command replaces the one sent with the previous one. These commands can be issued only by the session owner device.

Second, there are the commands for interacting with the content, i.e., I/O events, such as mouse movements or clicks and keyboard key events. Support for these commands is required because many display devices have very limited interaction capabilities, if at all. These commands can be issued by any mobile device interacting with the display service. However, it is the responsibility of the mobile users to synchronize with each other when, for instance, browse a document or play chess. Reducing the latency of these commands is one important challenge since interactive response requires fairly short latencies.

Third, there are the commands used for accessing content from remote servers, including but not limited to web servers. Typically, these commands configure the display service before it accesses the remote site; for instance, user credentials, such as cookies, from the mobile device are pushed to the intelligent display before the remote access is made.

Fourth, there are the commands for deleting the downloaded content from the intelligent display device, which typically returns the display to the idle state. These commands can only be issued by the session owner device. The display may also automatically clean up after the session times out. Some of these commands store back on the mobile device the modified content, such as an edited document and its viewing state (current page), or the state of an applet-based chess game, for later access. The later commands require content-specific extensions of the service implementation.

2.2 *Inverted Browser* Prototype

In our first prototype, a Web browser, such as Internet Explorer, is run on the public display and wrapped with a web service. The web service receives requests for displaying content and for interacting with it. The display service is described as a set of endpoints using the Web Services Description Language. Operations are described abstractly and then bound to a network protocol and message format to define an endpoint.

In the initial stage, the mobile device discovers the web service and dynamically obtains the client software stack for interacting with the web service using capabilities of the OSGi-based middleware layer. The client software invokes the web service with a *self-URL* (URL of the web server running on the mobile device). The web service invokes the browser, which fetches content from this URL over *HTTP* and displays it. The web service also receives UI control messages from the mobile device, such as mouse and keyboard events and inserts them in the system event queue. This allows the user to interact with the displayed content. Figure 2 shows the five main components of the *Inverted Browser* prototype.

***Inverted Browser* Web Service.** This service runs on the public display and implements two main methods: *DisplayContent* and *UIControl*. *DisplayContent* receives a URL as parameter and passes it on to a browser, which fetches content and displays it. *UIControl* receives control messages, such as mouse events, and passes them onto UI Monkey.

UI Monkey. This component receives the I/O events, such as mouse clicks, mouse movements, keyboard events, and inserts them in the system event queue. This allows the mobile device to interact with the displayed content.

***Inverted Browser* Client**. This component represents the client stack corresponding to the *Inverted Browser* Web Service; this component can be downloaded on the fly unless already cashed on the mobile device. It provides the user interface on the mobile device for interacting with the public display. There are two main components of this interface. The first component allows the user to browse the file system on the mobile device and pick the content he wishes to display. The URL corresponding to this file is generated automatically and communicated to the public display by invoking the *Inverted Browser* Web Service. The second component allows the user to generate mouse and keyboard events, such as left-click, right-click, page-up, page-down etc. It also programs the hardware button on the mobile device so that it can be used for generating mouse movements.

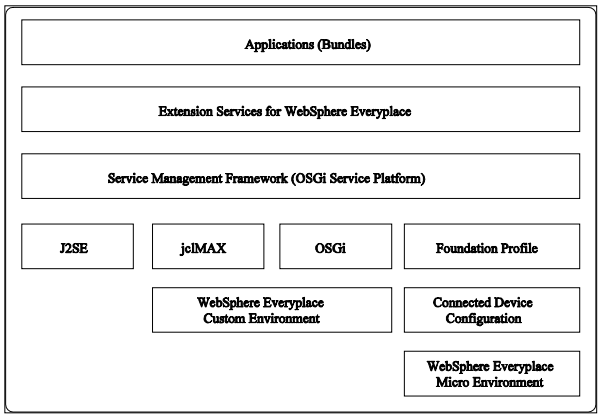

Figure 1: WCTME Architecture

Content Service. This component serves user's personal content over HTTP. It can also be downloaded and dispatched for execution on the fly.

Middleware. All the aforementioned components, with the exception of the *UIMonkey*, run on top of a middleware layer, which provides the capability to provision and run web services, http services, and download code on the fly. This middleware is the only software necessary on the user's mobile device to bootstrap the *Inverted Browser* software stack.

The *Inverted Browser Web Service*, *Inverted Browser Client* and *Content Service* are implemented in Java using the API's exported by the middleware. *UIMonkey* is implemented in Perl. The *Inverted Browser* architecture was tested on HP iPAQ h6325 phone which runs Windows Mobile 2003.

Workplace Client Technology Micro Edition (WCTME) [6] is used as the middleware. This middleware has been designed and implemented by IBM for mobile devices. It is currently available for Windows Mobile, Windows XP/NT/2000, Palm OS, Linux and Sharp Zaurus.

Figure 1 shows the WCTME architecture. WebSphere Everyplace Micro Environment (WEME) is IBM's implementation of J2ME that includes both Connected Device Configuration (CDC) as well as Connected Limited Device Configuration (CLDC). Service Management Framework (SMF) is IBM's implementation of the OSGi Service Platform Specification. SMF allows applications and services to be downloaded on the fly, as *bundles*. *Bundles* have manifests with special headers that enable sharing of classes and services at the package level. *Bundles* can be started and stopped dynamically. In many cases, the updates can be performed over the air without user intervention.

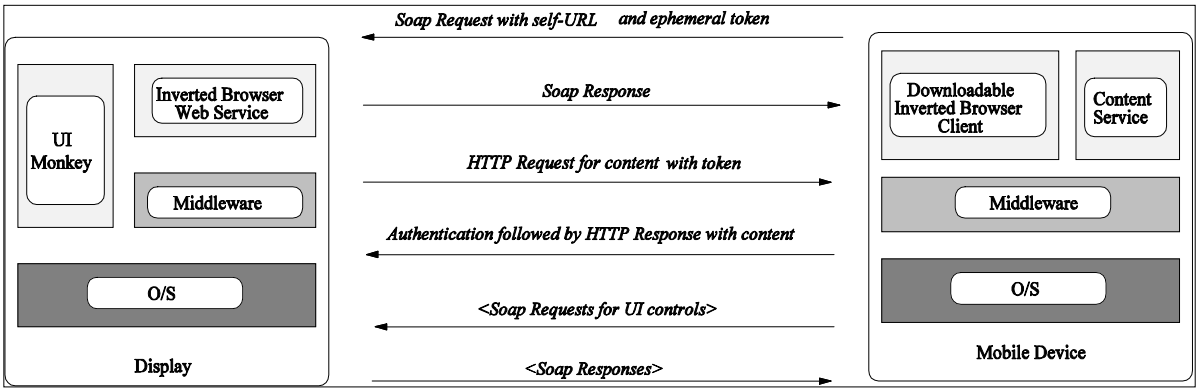

Figure 2: Inverted Browser Protocol

WCTME comes with many preinstalled bundles/services. Among them are MQe and DB2e which are embedded versions of IBMs widely known products. Needless to mention, WCTME provides complete support for web services and HTTP servers.

2.3 *Inverted Browser* Protocol

Figure 2 gives an overview of the Inverted Browser protocol. The *Inverted Browser* Client generates a SOAP request for the *Inverted Browser* Web Service. The SOAP request consists of a URL from where the content is to be fetched. This URL can correspond to a remote web server, or the server running on the mobile device. In the second case, the URL also includes an *ephemeral token* which serves as session key.

Upon receiving the SOAP request, the *Inverted Browser* Web Service sends back a SOAP response acknowledging receipt. It then invokes a browser and passes on the URL. The browser sends an HTTP request to the URL. In the case where the URL corresponds to the web server running on the mobile device, the Content Service is invoked.

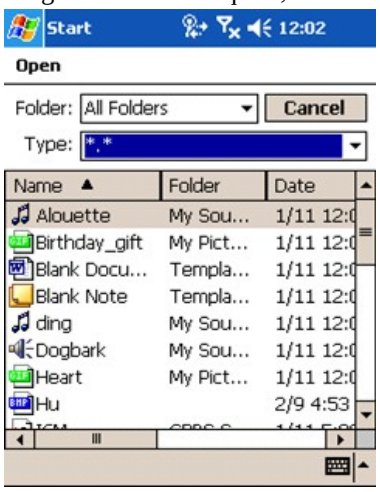

Figure 3: Inverted Browser iPAQ view

Content Service first authenticates the incoming request by inspecting the ephemeral token contained in the request. The ephemeral token serves to ensure that the incoming request is from the public display invoked by the user. If authentication is successful, the Content Service responds with the file after setting the correct MIME type and the browser displays it.

User generates control events (such as mouse clicks) using the UI provided by the *Inverted Browser* Client. The events are encapsulated in a SOAP request and communicated to the *Inverted Browser* Web Service, which passes them onto UIMonkey. The UIMonkey inserts the events in the system event queue. The Inverted Browser Web Service sends back a SOAP response at the end of the operation. Figures 3 and 4 show screen shots of the Inverted Browser

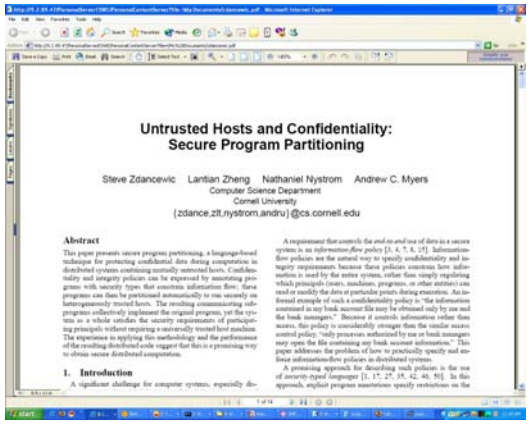

Figure 4: Inverted Browser host view

3. Alternative Solutions

This section describes two alternative approaches to implementing the functionality in Section 2 by leveraging existing remote access communication

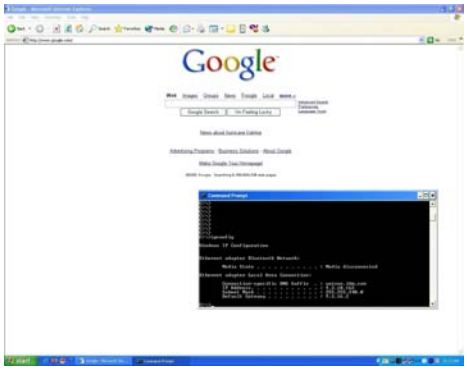

Figure 5: VNC Server

Figure 6: VNC Client on PDA

protocols to accomplish some of the equivalent functionality. For instance, if the mobile device runs a HTTP server, the display could simply pull content from the mobile device into a standard web browser. For instance, this approach is used by the Personal Server[26]. On displays that support virtual or physical keyboards, the user may directly type in the URL. On displays that do not support keyboard input, one would need to find an alternative way of provide the URL.

To ensure that the personal files on the mobile device cannot be accessed by any web browser, it will typically be necessary to protect the HTTP access using a password. In addition to providing the URL, the user would need to enter a password on the display device. Also since the user needs to provide this password manually, the same password is likely to be used across multiple files as well as multiple displays. One may wish to use SSL to secure the channel to protect the password and the private content.

The *Inverted Browser* approach avoids these complexities by enabling the user to select the content that he wishes to view on the mobile device and it either pushes this content to the display, or it pushes an ephemeral token to the display that authorizes it to pull only the content selected by the user.

In spite of these advantages of the *Inverted Browser* approach, it is still a new protocol that needs to be adopted by several displays and mobile devices before end users can benefit from its services. Therefore it is important to examine whether one can obtain a significant fraction of the functionality using existing standards, and to compare these approaches with the *Inverted Browser* approach.

3.1 Thin-Client Approach

A typical thin-client platform consists of a client application that runs on a client device and a server application that runs on a remote machine. GUI interaction operations such as keystrokes and mouse clicks that are performed on the client device are sent to the server. Screen or Window updates are sent to the client device. All of the application logic executes on the server with the user interface exported to the client.

One could use existing thin-client protocols to control the display from the mobile device and combine it with the standard password protected HTTP server approach described above. In this alternative, the thin-client server application runs on the display and allows mobile devices to take control of the display.

One of the nice features of the VNC protocol is that a client is available for PDAs and secondly, unlike the Microsoft Remote Desktop, the screen on the large display does not lock up as soon as the remote access client connects to the display PC.

To use VNC, we install Mocha VNC client [4] on the iPAQ and a VNC server on the display. Usually VNC servers are password protected to ensure that only authorized users connect to the display. In our case, since we wish to allow any mobile device to connect to the display, we use a simple password for the VNC server that is displayed prominently on the display monitor. Once the user connects to the display from the VNC client on the PDA, the user can send keystrokes and mouse operations to the display from the PDA. The content fetch can be triggered by entering a URL on the PDA using the PDA's soft keyboard. Subsequent controls, such as clicks on the scroll bar, or page up/down controls, can be sent from the PDA to the display.

In addition to sending the control operations from the mobile device, the VNC protocol also sends screen updates from the VNC server to the VNC client. For our application, the screen updates are not very useful since the user is generally in visual range of the VNC server's display. These updates tend to create a high volume of network traffic between the mobile device

and the display which is wasteful for the particular mode in which we are using VNC. VNC was not designed for situations where the client and server displays are within visual range of each other. One could consider an enhancement to the VNC protocol where the screen updates are turned off and only refreshed upon explicit user request.

Figures 5 and 6 show the screenshots of a VNC server running on a Windows XP desktop and of a VNC client running on Windows Mobile 2003 PDA, respectively

3.2 Bluetooth HID

The Bluetooth HID (Human Interface Device) [1] profile is another standard that is beginning to see adoption. Like VNC, Bluetooth HID enables mobile devices to send control operations to other devices. A mobile phone that supports the Bluetooth HID profile can act as a remote control for a computer capable of supporting HID devices. When connected to the computer over Bluetooth, the mobile phone would act like a combined mouse and keyboard. Unlike VNC, there are generally no screen updates, but the two devices generally need to be connected over a direct Bluetooth connection. Furthermore, only very small fraction of mobile devices supports Bluetooth HID or equivalent protocols.

Figure 7: Bluetooth Remote Control running on Sony Ericsson P900

Bluetooth HID itself does not provide any content transfer capabilities but it can be used in conjunction with the Bluetooth file transfer profile. Alternatively if the mobile device also supports TCP/IP connectivity, perhaps over a separate wireless interface, one could use the password protected HTTP server model to transfer files.

For our evaluation we used a proprietary J2ME implementation of Bluetooth HID-like protocol, called Bluetooth Remote Control [2] and tested it with the Sony Ericsson P900 phone. The Bluetooth Remote Control client was installed on the P900 and the corresponding server was installed on the display PC. Figure 7 shows a picture of Bluetooth Remote Control starting up on Sony Ericsson P900. Bluetooth Remote Control is not supported on the iPAQ.

4. Evaluation

4.1 Evaluation Metrics

Several important metrics have to be considered while building a system for symbiotic displays.

- **Usability:** The entire process of transferring content from the mobile device to the display, interacting with the display, and cleaning up the state at the end of the interaction should be simple and intuitive. While interacting with the content, the user should not have to split his attention between the mobile device and the display device. In other words, the user should be able control the mouse or send simple key stroke commands to the display from the mobile device without having to look at the mobile device.
- **Response time:** When the mobile device is used to send control events the display, the display should respond quickly and update the screen.
- **Energy:** Most mobile devices carried by users are battery-powered. Interacting with the environmental displays should not excessively drain the battery.
- **Privacy:** Can displayed files be deleted from the displays automatically? Can the user selectively share only some content with the public display? Can the user ensure that the content got transferred *only* to the intended public display and cannot be accessed from somewhere else?
- **Deployment Overhead:** The cost of deploying public displays, maintaining and updating the software components must be kept low. When new content formats become popular, it must be possible to dynamically upgrade the display service to support them. Displays should also be usable by a wide range of mobile devices.
- **Development Effort:** Software complexity has many side effects, such as high maintenance requirements, lack of robustness, security vulnerabilities, and lack of extensibility. The development effort should therefore be minimized. The adoption of well designed middleware enables code reuse and simplifies application development.

We evaluate the approaches: *Inverted Browser*, *VNC Thin-Client* and *Bluetooth profiles* by comparing

them against each other quantitatively as well as qualitatively on the design parameters described above.

4.2 Experimental Methodology

As stated earlier, we used an HP iPAQ h6325 PDA, and Sony Ericsson P900 phone for our experiments. The HP iPAQ h6325 runs the Windows Mobile 2003 OS, while Sony Ericsson P900 runs Symbian OS. The iPAQ supports both 802.11b and Bluetooth wireless connectivity, while the Ericsson phone only supports Bluetooth. Both devices support a cellular telephony interface but this was not used in our experiments. We used a standard desktop PC attached to a large monitor for the display. The desktop PC was connected to the lab network using Ethernet.

To test the *Inverted Browser* we installed the mobile device software components on the HP iPAQ h6325 on top of the WCTME middleware. We also installed the relevant display side software components on the display PC. We tested the Inverted Browser over both the 802.11b and Bluetooth interfaces. Our lab has several 802.11b access points, which are connected to the network. The iPAQ connected over 802.11b, obtained an IP address over DHCP and could then interact with the display over the network. For the Bluetooth experiments, we installed our own Bluetooth access point on the network which provided wireless connectivity to the iPAQ, and once the link level connectivity was established the iPAQ obtained an IP address from the DHCP server as in the 802.11b case. A GUI on the iPAQ enables the user to specify the name or IP address of the display device and also choose the content that the user wishes to view on the display. The GUI also enables the user to remotely control the keyboard and mouse from the PDA using the UI components described earlier.

As stated earlier, we used *Mocha VNC [4]* on the iPAQ h6325, and a VNC server on the display PC to test the thin client approach. The user starts the VNC client on the PDA and connects to the VNC server running on the display by specifying the name or IP address of the display.

The display monitor has a label with name and IP address of the display as well as the password for its VNC server. The user then manually starts a browser on the display device, and types in a URL that refers to the relevant file on the PDA. The starting of the browser and the typing of the URL may be done using the keyboard and mouse of the display, or using the I/O capabilities of the PDA. The PDA also runs a HTTP server to send the requested file to the display. The entire file system of the PDA is available to the HTTP server. However these contents are password protected, so that the HTTP server refuses to serve up files to unauthorized requestors. This adds an authentication step to the process where the user must provide a user name and password on the display using either its own I/O devices or using the PDA's I/O devices.

For testing Bluetooth HID, we used a J2ME implementation of Bluetooth HID capabilities, called *Bluetooth Remote Control [2]* which runs over the Bluetooth Serial Port Profile. Since this only worked on the Sony Ericsson P900, we were unable to test it on the iPAQ. The Bluetooth Remote Control works over a direct point-to-point connection. To test this we added a USB Bluetooth interface to the display device. First the file is transferred to the display using Bluetooth FTP, and then the Bluetooth remote control is used to send keyboard. We used *Windump* (implementation of *tcpdump* for Windows OS) to monitor and analyze network traffic between the display PC and the iPAQ, for both *Inverted Browser* and *VNC Client*. Using Windump we are able to measure the total amount of network traffic as well as the round-trip latency of the control operations. For the experiments where the mobile device was directly connected to the display PC over Bluetooth, we used USB monitoring software to measure the data transfer between the PC and the USB Bluetooth interface.

The response time corresponding to mouse events is a good indicator of user experience. To get an estimate of the response time, we measured the time interval between the incoming message representing a mouse click and the corresponding response from the display. This approach gives a lower bound on the response time, and enables a uniform comparison between the three different solutions. For the *Inverted Browser*, we measure the response time interval over both 802.11b and Bluetooth.

The total energy spent by the mobile device on an application primarily consists of: (1) Energy spent in transmitting and receiving data over the network interface, and (2) Energy spent on the other device sub systems such as the processor, display, backlight, etc. Our measurements are primarily focused on the energy used in the network, since this is the additional energy overhead of using the environmental display To get an estimate of the network energy consumption of the three solutions, we measure the total number of bytes exchanged between the mobile device and the public display for reading an 8-page PDF document. Since we used two different mobile devices, it was not possible to carry out a fair head-to-head comparison of the total energy for the three solutions.

Table 1: Comparative Study

Solution	Usability	Response Time	Network Energy	Privacy	Deployment Ease	Development Ease
Inverted Browser	****	450ms (.11b) 650ms (BT)	630KB*k 630KB*c	***	****	***
Thin-Client	**	150ms	10100KB*k	**	**	*
Bluetooth	***	200ms	670KB*c	*	*	**

4.3 Results

Table 1 shows the qualitative and quantitative comparison between the three approaches.

4.3.1 Usability. Although VNC provides I/O capability for interaction, it does not provide any content transfer capability. So one needs to add a password protected HTTP server for content transfer. This results in additional steps. In the case of Bluetooth, if one uses the Bluetooth File Transfer profile to transfer the content the user must manually delete the content from the display. In contrast, the *Inverted Browser* approach to content transfer needs the user to simply choose the file on the mobile device and the software takes care of the rest.

VNC suffers from the split attention problem. The user has to look at the PDA screen and use a stylus to position the mouse cursor. *Inverted Browser* and Bluetooth HID use hardware buttons for generating mouse and keyboard events. A hardware pointing mechanism such as a TrackPoint™ on the mobile device that relays mouse events to the large display can alleviate this problem for VNC.

4.3.2 Response time. Experimental results show that the response time for a mouse click is highest for *Inverted Browser*, 450 msec in the case of 802.11b and 650 msec in the case of Bluetooth. This can be attributed to the thick software-stack corresponding to the web service middleware. By virtue of being *closer* to the network layer, VNC Client and Bluetooth HID perform better. This implies better user experience with VNC Client and Bluetooth HID as compared to *Inverted Browser*. However, considering that performance of web services will improve with faster hardware and with software optimizations, we believe that the response time for the Inverted Browser can be within acceptable bounds.

One interesting point to note is that in all three cases the response time is not fast enough for interactive operation. When directly attached input devices are available on the display, or a dedicated remote control is available, they are preferable.

4.3.3 Energy. In our experiment of reading a 8 page PDF document, *Inverted Browser* and Bluetooth Remote Control generated almost the same amount of network traffic, approximately 650 KB. In contrast, VNC generates approximately 10100 KB, which is more than 15 times the traffic generated by the others. The increased traffic is attributed to the screen updates that VNC sends back to the PDA. Though the Bluetooth messages are smaller than the SOAP messages used by the *Inverted Browser*, there appears to be a significant amount of background Bluetooth traffic. We are investigating this, but have not been able to identify the reason for this traffic since we do not have the source code to the Bluetooth Remote Control software.

Bluetooth radios operate at lower power levels compared to 802.11b. Typical averages are 200 mW for Bluetooth and around 1W for 802.11b. These numbers vary somewhat with hardware vendors and will reduce with time. Let c be the constant that represents the average energy per byte consumed by Bluetooth radio while communicating (transmitting or receiving), and *k* be the constant representing the average energy per byte consumed by an optimized 802.11b wireless card. The ratio of *k* to *c* will vary with time and from device to device. However it is generally safe to assume that *k* is at least two times *c*. *Inverted Browser* running over Bluetooth consumes almost as much energy as Bluetooth Remote Control. The network energy consumed by the Inverted Browser running over 802.11 is at least twice as much as that consumed by the Bluetooth Remote Control. VNC client consumes at least 30 times more network energy than Bluetooth Remote Control, and at least 15 times more network energy than *Inverted Browser*. Also, a high amount of energy is spent by the mobile device on the screen updates and is not accounted for in these measurements.

4.3.4. Privacy. There are three major privacy issues: (1) How do we ensure that the data is deleted from the display at the end of the interaction? (2) How do we ensure that the only the identified content is

transferred? and, (3) How do we ensure that transferred content is not leaked to unintended destinations?

When the content is manually transferred as in the case of Bluetooth FTP, deleting data is a manual process. When the data is displayed by the browser in any of the three cases, the data is cached in the browser cache and will eventually get overwritten. In the case of the *Inverted Browser*, the display side software knows when the interaction is complete and can automatically clear the cache. In the case of VNC, users will have to clear out the browser cache manually if they do not want to rely on the browser's internal cache management to clear out the data.

We are in the process of building *amnesia* into *Inverted Browser*. *Amnesia* provides the capability of destroying private content after the user walks away, by sandboxing all the state associated with an interaction session and destroying it at the end of the session. The *Amnesic Inverted Browser* will not accept any other incoming connections while it is in session with the user, and will also be prevented from establishing any outgoing connections except with the user's mobile device to prevent viruses on the display from forwarding data to some other network device.

The *ephemeral tokens* used in the current *Inverted Browser* prototype ensure that only the user specified content is transferred to the identified display. In the case of a push based *Inverted Browser*, only the user specified content is pushed to the identified display by the mobile device. In the case of a standard HTTP access ensuring only the identified files are sent over is difficult because users cannot remember a separate password per file.

4.3.5. Deployment Overhead. *Inverted Browser* scores well on this criterion. The display side software is built on top of the WCTME middleware which allows the software to be remotely administered. Services can be started, stopped, uninstalled or updated from a remote server reducing the operational cost of displays. Even on the client side since we use WCTME, it is possible to download the required client side code dynamically, as opposed to manually installing each application beforehand. Since the interfaces are based on Web Services, it is possible to support mobile devices that use a different middleware stack. The *Inverted Browser* architecture is compatible with Windows Mobile, Windows XP/NT/2000, Linux as well as Palm OS.

Bluetooth based protocols require Bluetooth interfaces on both the public display as well as the mobile device. Devices that support other wireless interfaces such as 802.11b or emerging interfaces such as UWB cannot be used. Interoperability between Bluetooth stacks still continues to be a problem.

To use VNC, one would have to run an open VNC server on the display. This approach is likely to meet with some resistance from display administrators due to security concerns.

4.3.6. Development Effort. *Inverted Browser* is a good example of an application that leverages already existing code (i.e., the middleware) to accomplish a certain task with minimal additional development effort. *Inverted Browser* architecture reuses a collection of pre-existing libraries and services provided by the WCTME middleware. So the amount of code is relatively small. Thin-Clients such as VNC are big pieces of software with thousands of lines of code. Modifying a thin-client implementation to tailor it for our purpose, would involve substantial development effort. Similarly, adding the required features to the Bluetooth HID is non trivial. If we use VNC and Bluetooth as-is there is no additional development effort, but for the reasons mentioned earlier these protocols are inadequate.

5. Related Work

Prior work in this area can be categorized into three broad categories. The first focuses on sharing large displays with direct interaction capabilities, the second on using mobile devices to control displays, and the third on middleware for smart spaces.

Several systems have been created to support multiple-user interaction and collaboration on large displays. The Liveboard [12] system included a large rear-projection display that was connected to a workstation running the applications. A cordless pen with four distinct states that can be used several feet from the display served as the input device. The project focused on enabling group meetings and collaborations. The MMM [10] (Multi-Device, Multi-User, Multi-Editor) project used multiple mice to provide a user interface to allow multiple users to edit content on a single display. IBM BlueBoard [21] includes a large, interactive display surface with a touch-screen and a badge reader for personal identification. It has an integrated PC running a thin-client which can fetch display from a web-based content server. IBM Everywhere Display [18][25] uses a projector and a rotating mirror to project images anywhere in the room. Graphics and vision techniques are used to detect user gestures for interaction. In all of the above systems no content-transfer capability from mobile devices was provided. Moreover, mobile devices were not used to control the contents of the

display. Buxton et al. [11] explore the use of large displays for automotive design.

A few systems explore the interaction between mobile devices and large displays. Greenberg et al. [13] describe a system that allows users to move private notes created on their PDAs to a large display. One of their goals was to understand the distinction between private and public artifacts. The questions appear on a large display in the studio where the interview is conducted. Hello.Wall [24] is an ambient display transmitting organization-oriented information publicly and information addressed to individuals privately. A dedicated handheld-system called ViewPort can be used to show personal information and also show information related to visual codes on the wall. BBCi has built a system that allows people to watch interviews on displays in public spaces and to submit related questions via SMS text messages [23].

A few systems have been built to enable real-time interaction between handhelds and displays. The Xerox ParcTab [1] was essentially a thin client that accessed information, such as email and file from other machines using infrared communication. The ParcTab featured hard-wired buttons and a touch-sensitive screen and can be used as a remote control for other appliances. The Pebbles [16] project was among the first to explore handheld-based remote control functionality and study usability issues. Pebbles was an application built over Palm Pilots™ that had two components: Remote Commander, which provided remote control capabilities and PebblesDraw which allowed multiple users to draw at the same time. Palm Pilots™ were connected to the PC via serial port. Paek et al. [17] built a system to allow users to interact with displays using a mobile device. Their system adheres to a modular design in which diverse input devices send data to I/O modules, each of which is specifically designed to understand data from a single mode of communication. To date they have only implemented two I/O modules: email and instant messaging. Ballagas et al. [7] have implemented mechanisms to interact with public displays using phone cameras and visual tags. They use cameras on cell phone and vision techniques to accomplish this goal. Vodafone has also demonstrated a 13ftx13ft cube display system that allows users to request news and games using SMS text messaging [23].

Viewing private content on public displays is a problem that is being researched independently. Rukzio et al. [22] present a matrix relating the number of people that can see the display to the number of people that can interact with the display. Through this they show that there are different cases where personalization of services on public displays is useful. In an earlier work [9], we had described a technique for viewing private content in public, by blurring sensitive information on the public display and viewing it on the display of the mobile device.

Among the popular middleware based smart environments that use interactive displays are iRoom [14] and Gaia [20]. iRoom uses EventHeap [15] as the interfacing mechanism between connected devices, while Gaia uses the Gaia middleware.

The Inverted Browser shares goals with many of the applications described here, but is fundamentally different from them in the approach it takes and the capabilities it provides. The system that is closely related to Inverted Browser, in terms of the technology and approach used, is the Personal Server [26]. The Personal Server is a small handheld device that does not have any traditional I/O capabilities such as keyboard or display. The Personal Server includes a web server and acts like a wireless-enabled mobile hard-disk that can be accessed from host devices over Bluetooth. Most interaction between the Personal Server and the host device is initiated from input devices on the host device. The Inverted Browser and the Personal Server share the idea of running an embedded web server and using http for content transfer. However, there are a few key differences between the two. Since the Personal Server uses a host device for content selection, the listing of files, etc., must be sent to the host device, thus compromising the privacy of the user's data to some extent. In contrast, with the Inverted Browser, the native UI on the mobile device allows the user to select and send only the necessary content to the host device. The Personal Server architecture is built on top of Bluetooth, and is therefore protocol-dependent, as opposed to Inverted Browser which takes a network-protocol independent web services based approach. Finally, the Inverted Browser is a software solution that can be ported to any mobile device and aims at providing content-transfer and remote control capabilities for public displays.

6. Conclusions and Future Work

We introduced the vision for a viewing service, namely the *Inverted Browser*, which can be made available on displays and presented an initial prototype of such a service. Just as the HTTP protocol allowed people to retrieve content from web servers using browsers on heterogeneous devices and heavily impacted the way content was delivered, we envision that a protocol that allows people to send content from their personal devices, or other sources, to large

displays will dramatically change the way they view and interact with their personal content.

Our first prototype of the *Inverted Browser* utilized Web Services to create a wrapper to existing browsers where the wrapper triggered the browser to pull content. We are working on the push based model where all interactions go over a single TCP connection opened by the mobile device. In comparison to other approaches, such as the VNC Thin Client approach and Bluetooth HID, the *Inverted Browser* is agnostic to network type and consumes less energy. The latency of all three approaches is not acceptable and we are exploring several options for reducing the latency of the *Inverted Browser*.

7. References

[1] Bluetooth Human Interface Device Profile, www.bluetooth.org.

[2] Bluetooth Remote Control, www.bluetoothshareware.com.

[3] Citrix Meta-Frame, www.citrix.com.

[4] Mocha VNC Client, www.mochasoft.dk.

[5] OSGi Alliance www.osgi.org

[6] Workplace Client Tech. Micro Edition (WCTME), http://www.developer.ibm.com/isv/pvc/wctme.html.

[7] R. Ballagas, M. Rohs, "Mobile Phones as Pointing Devices," In *Proc. of the Workshop on Pervasive Mobile Interaction Devices (PerMid)*, 2005.

[8] S. Berger, S. McFaddin, C. Narayanaswami, M. Raghunath, "Web Services on Mobile Devices - Implementation and Experience," In *Proc. 5th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pp. 100-109, 2003.

[9] S. Berger, R. Kjeldsen, C. Pinhanez, M. Podlaseck, C. Narayanaswami, and M. Raghunath, "Using symbiotic displays to view sensitive information in public," In *Proc. of the Third Intl. Conference on Pervasive Computing and Communications (PerCom)*, 2005.

[10] E. A. Bier and S. Freeman, "MMM: A User Interface Architecture for Shared Editors on a Single Screen," In *Proc. of the 4th annual ACM symposium on User interface Software and Technology (UIST)*, 1991.

[11] W. Buxton, G. Fitzmaurice, R. Balakrishnan, and G. Kurtenbach, "Large Displays in Automotive Design," In *IEEE Computer Graphics and Applications*, Vol. 20, No. 4, pp. 68-75, 2000.

[12] S. Elrod et. al., "Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration," In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 1992.

[13] S. Greenberg, M. Boyle, and J. Laberg, "PDAs and Shared Public Displays: Making Personal Information Public, and Public Information Personal," In *Personal Technologies*. No. 3, Vol. 1, 1999.

[14] B. Johanson, A. Fox, "The interactive workspaces project: Experiences with ubiquitous computing rooms," *IEEE Pervasive Computing Magazine*, April-June 2002.

[15] B. Johanson and A. Fox, "The Event Heap: A Coordination Infrastructure for Interactive Workspaces," In *Proc. of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 2002.

[16] B. A. Myers, H. Stiel, and R. Gargiulo. "Collaboration using Multiple PDAs Connected to a PC," In *CSCW '98: Proc. of the 1998 ACM conference on Computer supported cooperative work*, 1998.

[17] T. Paek, M. Agrawala, S. Basu, S. Drucker, T. Kristjansson, R. Logan, K. Toyama, and A. Wilson, "Toward universal mobile interaction for shared displays," In *CSCW '04: Proc. of the 2004 ACM Conf. on Computer supported cooperative work*, 2004.

[18] C. S. Pinhanez, "The everywhere displays projector: A device to create ubiquitous graphical interfaces," In *UbiComp '01: Proc. of the 3rd international conference on Ubiquitous Computing*, 2001.

[19] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, Vol. 2, No. 1, pp. 33–38, 1998.

[20] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt., "GAIA: A Middleware Platform for Active Spaces," *SIGMOBILE Mob. Comput. Commun. Rev*., 2002.

[21] D. M. Russell, C. Drews, and A. Sue, "Social Aspects of Using Large Public Interactive Displays for Collaboration," In *Proc. of the 4th international conference on Ubiquitous Computing (UbiComp)*, 2002.

[22] A. S. E. Rukzio and H. Hussmann, "An Analysis of the Usage of Mobile Phones for Personalized Interactions with Ubiquitous Public Displays," In *Workshop on Ubiquitous Display Environments* in conjunction with UbiComp 2004.

[23] J. Scanlon, "If Walls Could Talk, Streets Might Join In," *New York Times*, September 2003.

[24] N. Streitz, et al., "Ambient displays and mobile devices for the creation of social architectural spaces" In *Public and Situated Displays: Social and Interactional Aspects of Shared Display Technologies*, pp. 387-409, Kluwer Academic Publisher, 2003.

[25] N. Sukaviriya, R. Kjeldsen, C. Pinhanez, L. Tang, A. Levas, G. Pingali, and M. Podlaseck, "A Portable System for Anywhere Interactions," In *Extended abstracts on Human factors in Computing Systems (CHI)*, 2004.

[26] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light, "The Personal Server: Changing the way we think about Ubiquitous Computing," In *Proc. of the 4th international conference on Ubiquitous Computing (UbiComp)*, 2002.

[27] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis, and M. Weiser, "An Overview of the PARCTAB Ubiquitous Computing Experiment," *IEEE Personal Communications*, Vol. 2, No. 6, pp. 28-33, Dec 1995.