

# ***Inverted Browser: A Novel Approach towards Display Symbiosis***

Mandayam Raghunath, Nishkam Ravi\*, Marcel-Cătălin Roșu, 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.*

*Our initial Inverted Browser prototype is based on a Web Services wrapper around a traditional Web browser. Our experiments show that the Inverted Browser approach is superior to other solutions in terms of user convenience, ease of use, energy consumption, and privacy, but interaction latencies need improvement.*

## **1. Introduction**

Mobile devices, such as smart phones and portable music players, are becoming preferred storage devices for personal data, mainly due to their constant availability, wireless connectivity, data capturing abilities, and increasing amounts of storage. However, their small screens limit the user's ability to interact with personal data.

At the same time, it is becoming clear that large displays hold promise as a pervasive technology due their increasing availability in public spaces. We are seeing projectors and other displays that support wired and wireless connectivity. Some of the newer projectors also accept portable storage media, such as 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.

In an earlier paper [6], we outlined a vision of intelligent display devices capable of displaying content in a variety of different formats and accessible to a 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.

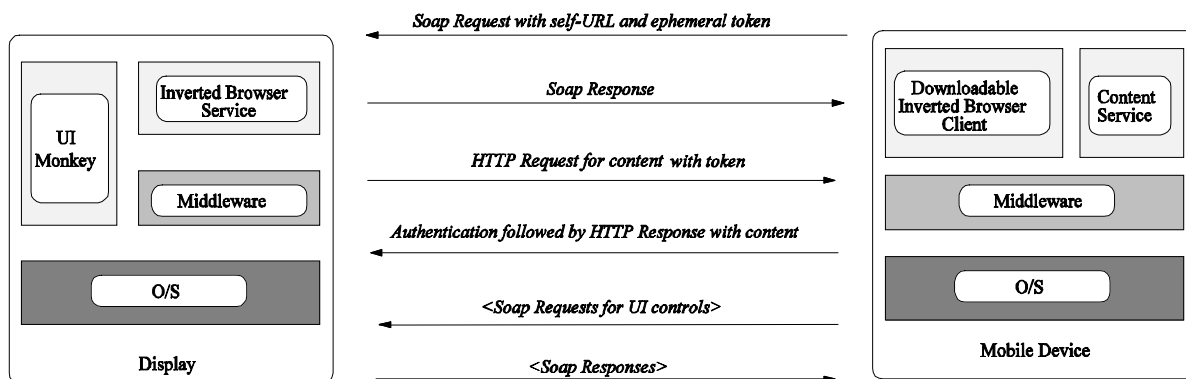
The *Inverted Browser* is a network service used by mobile users to push content from the personal device to the display and to control a browser-based viewer on the display. A push-based interaction model is more appropriate than a pull-based interaction model, since mobile devices may be protected by (provider-managed) firewalls or behind NAT devices. The proposed architecture accommodates independent evolution of each of the existing components.

Our initial prototype uses existing technologies, such as Web browsers, Web Services, and Open Services Gateway initiative (OSGi) middleware[4]. As an alternative to the *Inverted Browser* display service, we considered leveraging existing off-the-shelf solutions to accomplish the same end goal of viewing content from mobile devices. We report our qualitative and quantitative comparisons with Thin-Client technologies, such as VNC, and Bluetooth profiles.

The key contribution of this paper is a software architecture for intelligent network displays that leverages existing technologies. Similar to the widely-deployed web servers, delivering content to traditional Web browsers, we envision future public spaces being populated with intelligent displays offering their services, such as the *Inverted Browser* service, to users carrying mobile devices. In addition, we 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. Our approach is built around the Web browser



**Figure 1: Inverted Browser Protocol**

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 in the development of the existing Web browsers, support for diverse content types, and third party plug-ins. To enable the interaction between public displays and mobile devices, we adopt this simple and successful approach of web browsers with one key difference - we *invert* it.

Instead of the traditional browser model where the browser pulls content from a server, in the *Inverted Browser* approach, mobile users push commands and content from their personal devices to the display. Since standard browsers do not operate in a mode where some other device pushes content, we had to augment the browser to accept incoming connections, and processes the content and commands received on such an incoming connection. The service may be accessed over a variety of connections, such as 802.11, Bluetooth or IrDA. To get a better understanding of the challenges of this approach and to determine the level of optimization required for the display and client software stacks, we built an initial prototype of the *Inverted Browser* service.

## 2.1 Inverted Browser Capabilities

A mobile device that accesses an idle display gets a description of the display capabilities upon establishing a connection to the *Inverted Browser* service. This device becomes the session owner. Subsequent accesses by other mobile devices are accepted only if authorized by the owner. 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.

The *Inverted Browser* service responds to several different types of commands from the mobile device(s). 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. 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. Third, there are the commands used for accessing content from remote servers. Typically, these commands configure the display service with relevant information, such as user credentials or cookies, before it accesses the remote server. Fourth, there are the commands for deleting the downloaded content from the intelligent display device.

## 2.2 Inverted Browser Prototype

In our first prototype, a standard Web browser, such as Firefox, is run on the public display and wrapped with a web service. The web service receives and responds to the commands described above. 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.

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 addition to reducing the implementation effort, this allows the browser itself to be upgraded as newer versions become available.

Figure 1 shows the main components of the *Inverted Browser* prototype. The *Inverted Browser Client* running on the mobile device communicates

with the corresponding *Inverted Browser Service* running on the display. The client sends a SOAP request that includes 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 URL also includes an *ephemeral token* which serves as session key. This HTTP request is processed by the *Content Service* running on the mobile device. The 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. The client software also captures user gestures on the mobile device and sends these UI events to the display. The *UI monkey* accepts these UI events and inserts them into the system event queue.

The *Inverted Browser Service*, *Inverted Browser Client* and *Content Service* are implemented in Java using the APIs exported by the *Workplace Client Technology Micro Edition* (WCTME) [5] middleware, which include an OSGi compliant layer. The *UIMonkey* is implemented in Perl.

Since all of the mobile device components are running on top of OSGi, it is possible to dynamically provision the client software stack.

The *Inverted Browser* architecture was tested on HP iPAQ h6325 which runs Windows Mobile 2003.

### 3. Alternative Solutions

It is possible to implement parts of the functionality described in Section 2 by leveraging existing remote access communication protocols. For instance, the user could manually start a browser on the display and pull content from a web server running on the mobile device, as used in the Personal Server[10]. To use this approach on displays that do not support keyboard input, one would need an alternative way to provide the URL.

To ensure that the personal files on the mobile device cannot be accessed by a random 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 the 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 providers 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*.

#### 3.1 Thin-Client Approach

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. A thin-client server application runs on the display and allows mobile devices to take control of the display. The thin-client protocol sends regular updates of the screen from the display to the mobile device and also handles the input operations from the mobile device.

In our implementation we used the Mocha VNC client [3] 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. The content fetch is triggered by entering an URL on the PDA using the PDA's soft keyboard. Subsequent controls, such as clicks on the scroll bar, or page up/down controls, are sent from the PDA to the display.

For our application, the screen updates on the PDA are not useful since the user is in visual range of the VNC server's display. To avoid this unnecessary network traffic, one could consider an enhancement to the VNC protocol where the screen updates are turned off and only refreshed upon explicit user request.

#### 3.2 Bluetooth HID

The Bluetooth HID (Human Interface Device) [1] profile is another standard that is beginning to see adoption. Like the VNC thin-client, 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 the VNC thin-client, there are generally no screen updates, but the two devices generally need to

be connected over a direct Bluetooth connection. Furthermore, only a small fraction of mobile devices supports Bluetooth HID or equivalent protocols.

Bluetooth HID itself does not provide any content transfer capabilities but it can be used in conjunction with the Bluetooth file transfer profile or, if the mobile device also supports TCP/IP connectivity, with the password protected HTTP server model.

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.

#### 4. Evaluation

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 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.

**Response time.** When the mobile device is used to send control events the display, the display should respond quickly and update the screen.

**Network traffic.** Data over wireless interfaces is expensive from an energy perspective. Interaction with displays should not require excessive data transfer.

**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.

Table 1 shows the qualitative and quantitative comparison between the three approaches: Inverted Browser, VNC thin-client, and Bluetooth Remote Control.

**Usability.** Although the VNC thin-client 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 the user transfers the content using File Transfer, 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.

In addition, the VNC thin-client 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 Remote Control use hardware buttons for generating mouse and keyboard events. A hardware pointing mechanism such as a TrackPoint™ on the mobile device can alleviate this problem for the VNC thin-client.

**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 and the loose binding between the server and the UI monkey. By virtue of being *closer* to the network layer, VNC thin-client and Bluetooth Remote Control perform better. This implies better user experience with VNC thin-client and Bluetooth Remote Control as compared to *Inverted Browser*. However, the performance of web services will improve over time. In addition, a more optimized path for remote UI events can reduce the response time for the *Inverted Browser* to within acceptable bounds.

One interesting point to note is that in all three cases the response time is not fast enough for interactive operation. As a result, input devices attached to the

**Table 1: Comparative Study**

Solution	Usability	Response Time	Network Traffic	Privacy	Deployment Ease	Development Ease
Inverted Browser	****	450ms (.11b) 650ms (BT)	630KB	***	****	***
VNC thin-client	**	150ms	10100KB	**	**	*
Bluetooth RC	***	200ms	670KB	*	*	**

display, or a dedicated remote control are preferable.

**Network traffic.** In our experiment of reading a 8 page PDF document, the *Inverted Browser* and Bluetooth Remote Control generate almost the same amount of network traffic, approximately 650 KB. In contrast, the VNC thin-client 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 to keep the Bluetooth connection alive.

From an energy impact perspective, the VNC thin-client is significantly worse than *Inverted Browser*. In addition to the energy impact of the network traffic, additional energy is spent by the mobile device on the screen updates.

**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?

In any of the three solutions, when the data is displayed by the browser, 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 the VNC thin-client, users will have to clear out the browser cache manually if they do not want to rely on the browser's internal cache management. Similarly, Bluetooth Remote Control/HID users will have to clear out the browser cache manually. In addition, when the content is manually transferred as in the case of Bluetooth FTP, deleting the transferred data is a manual process.

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 *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 solutions which include standard HTTP access, ensuring that only the user specified files are sent over is difficult because one needs to use a separate password per file.

**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.

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 Ultra-Wideband) cannot be used. Bluetooth stack interoperability 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.

**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. The *Inverted Browser* architecture reuses a collection of pre-existing libraries and services provided by the WCTME middleware; as a result, the amount of new code is relatively small. Thin-clients such as VNC are large 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 Remote Control/HID is non trivial. If we use VNC thin-client and Bluetooth Remote Control/HID as-is there is no additional development effort, but for the reasons mentioned earlier these approaches are inadequate.

## 5. Related Work

We are unable to discuss all of the related research efforts in detail due to space constraints. Please see a longer version of this paper [8] for an extensive discussion on related work.

The Pebbles [7] project was among the first to explore handheld-based remote control functionality and study usability issues. Viewing private content on public displays is a problem that is being researched independently. Rukzio et al. [9] present a matrix relating the number of people that can see the display to the number of people that can interact with the display.

The system that is most closely related to *Inverted Browser*, in terms of the technology and approach used, is the Personal Server [10]. 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 interactions between the Personal Server and the host device are 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, as opposed to *Inverted Browser* which takes a network-protocol independent approach. Finally, the *Inverted Browser* is a service that can be used by any mobile device that can support the corresponding client software.

## 6. Conclusions and Future Work

We introduced the vision for a display service, namely the *Inverted Browser*, and presented an initial prototype of it. 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 allowing 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 one connection opened by the mobile device. In comparison to other approaches, such as the VNC thin-client and Bluetooth Remote Control/HID, the *Inverted Browser* requires less network traffic and is agnostic to network type, respectively. 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](http://www.bluetooth.org).
- [2] Bluetooth Remote Control, [www.bluetoothshareware.com](http://www.bluetoothshareware.com).
- [3] Mocha VNC Client, [www.mochasoft.dk](http://www.mochasoft.dk).
- [4] OSGi Alliance [www.osgi.org](http://www.osgi.org)
- [5] Workplace Client Tech. Micro Edition (WCTME), <http://www.developer.ibm.com/isv/pvc/wctme.html>.
- [6] 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.
- [7] 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.
- [8] M. Raghunath, N. Ravi, M.C. Rosu, C. Narayanaswami, "Inverted Browser: A Novel Approach towards Display Symbiosis", *IBM Research Report RC23719*, Sep 2005.
- [9] 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.
- [10] 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.