Towards Multi-Application Public Interactive Displays

Tomas Lindén, Tommi Heikkinen, Vassilis Kostakos, Denzil Ferreira, Timo Ojala
MediaTeam Oulu, Department of Computer Science and Engineering
University of Oulu, Finland
{firstname.lastname}@ee.oulu.fi

ABSTRACT
Public displays are becoming a common sight in the urban landscape and they are increasingly being equipped with interaction components such as touch screens. In addition, the Web and its set of enabling technologies are attractive for realizing applications for public displays. We argue that to develop multi-application public displays, then services generally need to be easy to develop, robust, and be easy to deploy and maintain. In this position paper we present a virtual machine-based Web application platform, for decentralized deployment of interactive services on heterogeneous public displays, which satisfies the aforementioned requirements. We also report on some usage experiences of utilizing the platform in a network of large public displays, which we have deployed in a mid-size city.

Categories and Subject Descriptors
H.5.4 [**Information systems**]: Hypertext/Hypermedia – *Architectures*; K.6.4 [**Computing Milieux**]: System Management – *Centralization/decentralization*

General Terms
Management, Performance, Design, Reliability, Experimentation

Keywords
urban computing, public displays, distributed computing, hardware virtualization, web technology

1. INTRODUCTION
Public displays are becoming a common sight in urban areas around the world and they are typically being used for one-way communication purposes such as digital signage and relaying information to the public. A recent development is that these kinds of displays are increasingly being equipped with touch screens and other interaction capabilities enabling two-way usage modes and thus more sophisticated applications including info-kiosks, ticket vending machines, and various games and entertainment services.

Our experiences with a network of 12 such urban interactive displays over the period of two years have shown that due to the public nature of these installations, there are several characteristics that set them apart from typical personal computing such as desktop and handheld device. Based on our experience, we have identified the following requirements for public display services: First, a high degree of robustness and availability is required of interactive services being deployed, since service failure may be highly visible, potentially resulting in bad publicity on the display operator's part. Second, it is important that content can be easily authored and that development resources are readily available. Third, displays and their services should be easy to setup and maintain.

In this position paper we argue that one way to deal with the specific requirements of multi-application public displays is to use a virtual machine-based Web application deployment platform. The combination of using web applications and virtual machines addresses the three requirements mentioned above. First, by deploying web applications on a display-specific virtual machine, we improve robustness against server and network problems. Second, based on our experiences, we can lower the cost of developing services for public displays by leveraging web applications' low learning curve and large available developer base. Third, by employing the virtual appliance approach, we can encapsulate all our functional components into a single file, which eases deployment [7]. Additionally, by using a Unix-like OS (Linux) for our virtual appliance we are also able to ease remote maintenance.

Next, we present related work, and then we discuss the themes that influenced the design of our platform. Subsequently, we describe its actual implementation and how it is currently being used in our large public display infrastructure. Then, we discuss our experiences with the platform, and finally we report some unexpected side-effects of using the virtual machine-based approach for the development, hosting and maintenance of public display services.

2. RELATED WORK
Our previous work [6] in this field has focused on providing a mid-size city with an open public display infrastructure with the goal of making the authoring of public display services more accessible to the general public, requiring only basic web design and development skills. This public display infrastructure uses a Web-based screen real-estate partitioning framework for large public displays [3] that is deployed on the display hardware in a decentralized fashion. The applications themselves, however, have been deployed in the more traditional client-server Web application hosting style.

Other researchers have also relied on Web technologies for realizing public display services. Stahl et al. utilized web technologies for realizing the display component of a smart shopping assistant [8]. Erbad et al. have utilized web technologies for implementing a middleware toolkit for interactive public displays [1]. Finney et al. used early web technologies in their flexible ubiquitous monitor project [2]. McCarthy et al. used the web in OutCast – a semi-public office space display [4]. Stortz et al. support content creation using web technologies in their e-

PerDis 2012, June 04 - 05 2012, Porto, Portugal

Campus deployment [9]. In addition, recent HTML5 off-line Web application support in browsers has made it possible to decentralize web-application deployments on client machines.

Virtual machines have also been employed in ubiquitous computing environments previously [5][7]. To our knowledge, none have used virtualization in conjunction with multi-application public displays. The approach advocated in this paper is to provide a common platform for enabling decentralized Web application deployments meant for heterogeneous networks of large public displays and utilizing virtual machines.

3. ARCHITECTURE AND IMPLEMENTATION

Our experience has shown that to support an ecosystem of multi-application public displays, services need to be

- easy to develop,
- robust, and
- easy to setup and maintain

Our proposal for addressing these requirements rests on two sets of technologies: hardware virtualization and Web technologies. Here we discuss the design, architecture and implementation of our distributed web application deployment platform for public displays.

3.1 System architecture

In this section we discuss the two main themes that influenced the design and architecture of the proposed distributed Web application deployment platform: decentralization of Web application deployments and virtualization of a public display's control unit's hardware resources.

3.1.1 Decentralization of Web application deployments

In today's Web, content and applications are typically hosted on numerous servers, seamlessly combined, and delivered, on-demand to web clients running on computers all over the world. This approach has the advantage that it eases the delivery of content to clients, harmonizes the clients' runtime environments and gives convenient managerial control and access for updating content and applications on the centralized servers.

Usage of the Web as the enabling technology for delivering dynamic and interactive content to public displays is attractive for its expressive power, maturity, and large existing leverage-able content and code base, in addition to the reasons mentioned above. However, it has one limitation in the context of public displays. Web applications are typically delivered to browsers running on computers operated by people who often access the Web in a flexible, ad-hoc manner, and if there is a problem with a server or the network is slow users may try to fix the situation by refreshing a page, go to some other Web site or change to a different network up-link.

A public display, on the other hand, may have been designed with a much narrower usage scenario in mind, tailored to its location and context. It therefore requires much more robustness with regard to the content and services it is delivering to its intended user base. If content is hosted on a remote server and it is delivered to the public display on-demand, without any off-line content caching mechanisms, it is very vulnerable to server and network related problems. When such a problem occurs, it results in service failure, which can be detrimental in a public setting and may easily cause bad publicity on the display or service provider's part, even if handled gracefully.

A solution for improving public display Web application fault tolerance with regard to network and server problems is therefore needed. This can be achieved by authoring the content in such a way that a compatible browser is able to dynamically store critical parts of the application on the client machine for later retrieval and access the local cache if the original server somehow becomes unreachable. This mechanism is a handy feature in early versions of web browsers. When dynamic database-driven Web applications finally took off, pages stored by caching mechanisms quickly became outdated, because Web pages were created on-the fly for every request separately. Today, similar functionality is again becoming available for rich Web applications with the advent of *HTML5 Offline Web application* compatible browsers. The drawback of utilizing HTML5-features for enabling offline-support is that it makes applications more complex, and results in higher application development and maintenance costs.

An alternative solution is to replicate the Web application hosting environment to the client machine, deploy the application locally and push content updates to the client's machine over the network, as they become available. This may be more suitable when the clients are known and relatively limited in number, such as in the case of public display deployments. Of course, all aspects of an application might not be possible to support off-line, especially if it resides on a 3rd party server elsewhere on the network. It might, however, be possible to decentralize an application's most critical parts, thus making at least some of the content available in the event of the network or a server becoming inaccessible.

3.1.2 Virtualization of a public display control unit's resources

Public displays, due to their public nature and complexity, typically require some custom services to be present near or on the display's control-unit itself. These services perform a variety of functions, such as monitoring, content caching and providing UI-rendering services. The services are typically often tailored for the particular service that the display is performing and are written for a particular run-time environment provided on the public display's control unit. Being custom pieces of software, they need dedicated development and maintenance resources, which may be expensive and result in efforts being limited to supporting only a single or at most a limited set of compatible operating environments. This in turn limits the amount of available public display hardware and operating system platforms being supported by a public display service vendor. While not a significant problem in itself, a more decoupled relationship between hardware, public display operating systems and custom public display software would give access to a larger number of display setup options, and thus potentially lowering deployment costs and hopefully providing a better end-user experience.

Additionally, if public display applications and content needs to be deployed locally, for-example for robustness reasons, as discussed in the previous section, they need to be written for any environment available on the public display's control unit. Since maintaining several different software run-time environments on different kinds of public display operating system and hardware setups may result in a combinatory explosion and subsequently an unmanageable maintenance burden, a trade-off between the

number of public display setups and software environments needs to be made.

A solution to addressing the combinatory explosion problem would be to separate the display's hardware and OS from the software deployment environments using a single virtualization layer, ideally making it possible to support an arbitrary number of combinations. The problem is then reduced to just concern supporting the virtualization layer on the different hardware/OS-setups and keeping the additional maintenance burden introduced by the virtualization layer manageable.

3.2 Virtual machine-based distributed web application platform

The approach we have taken to realizing our multi-application public display infrastructure builds on the idea of putting as many of a display's software components as possible inside a virtual machine that is paired with a particular display. The virtual machine is then installed on the display itself or as close as possible to it from a network topology point of view (see figure 1).

If the individual virtual machines installed on different physical machines are similar, regardless of the host machine's OS and hardware, then we have implemented the virtualization theme discussed in section 3.1.2. If the virtual machine also contains a Web application hosting environment, then a formerly centrally deployed Web application compatible with the environment can be installed on the virtual machine directly, thus implementing the decentralization theme presented in section 3.1.1.

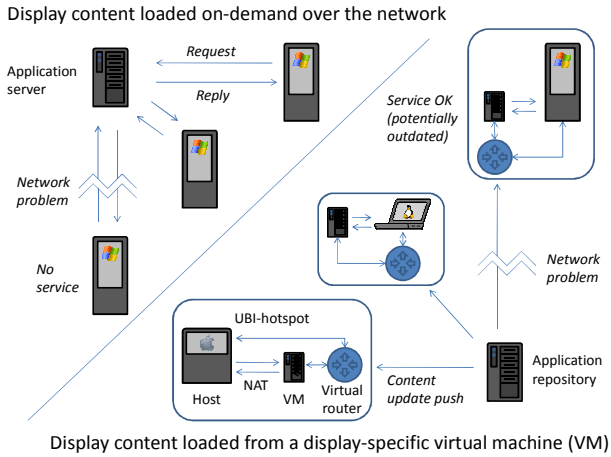

Figure 1. A comparison between the more traditional centralized web application approach (left) and the proposed virtual machine-based approach presented in this paper (right).

A display's host OS and the virtual machine's OS, also called guest OS, are separate from a computing point of view and applications running on either machine are isolated from each. The guest and host machines are also separate from each other from a network topology point of view in that they have separate network interface cards (NICs), one physical and the other virtual. This means that they have different IP-addresses and appear as separate computers when viewed from the outside. In addition, if the host and guest machines are also equipped with a second set of NICs that receive a private IP-address from the host-guest virtual network's DHCP-server, then, communication between the host and guest machine can be transparently maintained even if they become isolated from the rest of the network. Thus, if all resources needed by a display are located either on the display host OS or preferably on the virtual machine, then the display can be seen as a completely self-reliant, more fault-tolerant independent entity as long as the local content is not outdated.

3.3 Architecture of proposed platform

A layered architecture of the proposed decentralized web application platform for public displays can be seen in figure 2.

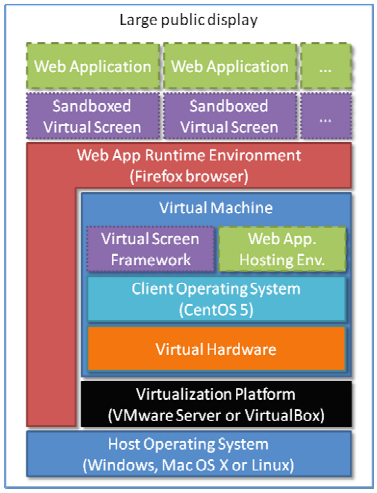

Figure 2. The main conceptual components of the proposed decentralized Web application hosting platform for public displays.

Our platform consists of two main functional components, a virtual machine, which was discussed in the previous chapter and a browser, which is installed and runs on the display's host machine's OS. The virtual machine contains a screen real-estate partitioning framework, which is responsible for dividing the public display's screen area into smaller sub-screens called virtual screens, and a set of standard web application hosting environments that cater to the needs of applications that are distributed for use on the public display.

The browser, which runs in full-screen mode on the display's host machine, implements the Web applications' runtime environment and virtual screen sandboxes (HTML iframes). The browser also gives Web applications access to the display's hardware and peripherals (such as web cameras and microphones). A more detailed description of the virtual screen framework can be found in [3].The virtual machine is specified to reserve up to one CPU core of the host machine, is allocated a small chunk of the display control unit's memory, and has two Ethernet cards, one "bridged" over the physical adapter and one connected to the machine's local virtual NAT-network.

The implementation of our virtual machine is currently packaged and distributed as a so called virtual appliance, which essentially consists of two files: one contains the virtual hardware specification and one the hard disk image. This virtual appliance can be copied and started on compatible virtualization platforms which include VMware Server 2 and Oracle VM VirtualBox.

Different virtual machines running on different displays are separated and identified by their hostnames. When a virtual machine on a particular display is started, it receives a fixed IP-address from our network's DHCP-server. Each display has also

been specified with a hostname that resolves to the display's IP-address. On public displays that run VMWare Server as their virtualization platform and have their virtual machines deployed locally, we have also specified their host and guest machines' respective NAT IP-addresses as so called hostname redirects, so that they are accessible from each other in case of network failure. E.g. when accessing a virtual machine from its host machine, the host machine's local DNS resolver resolves the virtual machine's NAT IP-address. From elsewhere in the network, the DNS resolvers resolve the IP-addresses as configured in the DNS-servers.

All applications that we have deployed on the displays' virtual machines are packaged using the Redhat Package Manager and updates to them are pushed in an automated fashion over public-key authenticated SSH-connections. Adding an application to a public display is done by inserting, via a web form, an URL into a database which resides on the virtual machine. The URL is subsequently loaded by the Virtual Screen Framework into a Virtual Screen (HTML iframe) when the application is launched. The URL may point either to a locally deployed application (residing on the virtual machine) or to one deployed anywhere on the public Web.

3.4 The UBI public display infrastructure

Our flagship large public display, of which we have several deployed both outdoors and indoors, is called *UBI-hotspot*, consists of a 57-inch touch screen display and an integrated control unit. They run Windows Embedded POSReady as their host operating system and VMware Server 2 as their virtualization platform (see figure 1). See [6] for more information.

We also have two facade-scale projector-type displays called *UBI-projectors* deployed outdoors in the market square of a mid-size city, which each project a 6x9 meter size projection on a prominent facade of a large building. They both have an all-weather projector-enclosure, a fixed-type large-venue projector and a Mac mini server running Mac OS X Server as the host operating system and VirtualBox as the virtualization platform. The UBI-projectors run a subset of the UBI-hotspots' content, and are currently only used for one-way broadcast.

In addition, we have two portable touch-screen-enabled public displays, called *minihotspots* that run Windows 7 as the host operating system. The portable UBI-displays run the same kind of content as our flagship UBI-displays. These displays have their virtual machines' running in a data-center and not locally in order to make the local display setups simpler and enable maintenance of the displays' decentralized services when the displays are packed away.

All displays run the Firefox browser in full-screen mode. It virtualizes the various displays' runtime environments of our virtual screen framework and our public display Web applications. On all virtual machines we run CentOS and a couple of Web application hosting environments (Apache Tomcat and LAMP [Linux, Apache httpd server, MySQL and PHP]). The virtual machines are used for locally deploying public display services. The virtual machine virtualizes the deployment environments and makes it possible to deploy identical copies of public display Web applications on all our displays in an automated fashion, regardless of the underlying host machines' HW and OS makeup.

Key applications, developed by our research team, such as a menu for launching applications on the displays, a digital signage-style advertisement media player and a map application with a service directory are hosted locally on the displays' virtual machines. Non-critical applications, such as applications provided by external parties, are still deployed in the traditional client-server Web hosting style due to its simplicity of managing updates on centralized servers without having to prepare RPM-packages for every update separately.

4. DISCUSSION

All public displays that we have in our infrastructure use the virtual-machine approach as presented in this paper and all services use web-technologies as presented above. By utilizing virtualization technologies and web technologies we have been able to homogenize our highly heterogeneous display infrastructure and we are able to deploy the same services locally on the displays regardless of the underlying hardware or operating system.

By utilizing web technologies, we have been able to tap into the large developer base available and we have a diverse range of display services deployed and in use. On the other hand the quality of services has sometimes been lacking and we have observed reluctance among web developers to perform the added burden of building an installation package (RPM) for decentralized deployment of services. This reluctance undermines the fault-tolerance benefits and the utility of our virtual-machine-based approach.

Even if our virtual machine-based approach of encapsulating a specific set of software components and isolating them from the host machine eases the setup and remote maintenance of a large heterogeneous network of public displays, it does not eliminate all configuration and maintenance efforts needed on the host machine's part. For example, the browser needs to be configured for full-screen tamper-proof use in public spaces and it needs to be kept up to date using remote maintenance tools available on the display's host operating system. In addition, any browser plug-ins, such as Adobe Flash Player, need to be maintained separately as well. In addition, with this approach there are two OSs for each display (the host's and the guest's) that need to be kept up to date and secured individually. Any components that have to be run on the host machine, for example due to simply not being available for our virtual machine's OS (Linux), need to be maintained separately as well. There is also a distinct overhead in installing the virtualization platform along with the virtual appliance and setting up the NAT-network between the host and guest OSs. Extra care must also be taken to make sure the NAT IP addresses are fixed and that the hostname overrides of the guest and host are setup correctly. However, these steps only have to be done once per display.

An additional benefit of using virtual machines on the displays is that the host and guest machines can be managed by different organizations, thus making it possible to cleanly and securely divide management responsibilities between the parties. Further, since a virtual machine encapsulates a service's components along with its OS, setup and maintenance of a display becomes easier and reduces the amount of configuration and installation required on the host machine.

Since the browser does not run on the virtual machine we also make it possible to place the virtual machine on a different machine from which the browser is running (i.e. the display itself). By placing the virtual machine on the display hardware itself, we receive a higher degree of decentralization and thus

higher load-balancing and fault tolerance benefits. On the other hand, by centralizing a group of displays' virtual machines we ease the resource load on the display control units and make the display setup simpler. The latter approach may also be useful if displays need to be setup quickly or in an ad-hoc manner. This way, we only need to make sure a display's browser runs in full screen mode and that it is pointed to its corresponding virtual machine.

Since many of our displays run a flavor of Windows (due to our displays' limited non-windows device driver support) as the host operating system, having most components on the virtual machine's Linux side eases remote behind-the-scenes maintenance, instead of having to do it manually using GUI-administration tools for each host machine separately. With a Unix-like OS (Linux) on each display's virtual machine, we can also perform the same maintenance operations in batch mode for all displays in parallel and in an automated fashion, regardless of the displays' actual host operating systems.

We have also found it useful to provide public display application developers with a copy of our virtual appliance so that they can setup an application development environment on their own computers that is virtually identical with the Hotspots' production environments. This makes it easier to develop applications directly for the platform on the developers' own machines and test that software packages (RPMs) are ready for deployment. Adding applications to our public displays currently requires one of our administrators' intervention.

Moving a virtual machine from one physical machine to another has turned out to be straightforward. As with physical machines, it is just a matter of shutting the virtual machine down, moving the virtual appliance's files to the other machine and starting it. No configuring except for adding the machine in the virtualization environment on the host machine is needed. This gives the flexibility of grouping virtual machines together or separating them as the need arises, in a relatively convenient way.

5. CONCLUSION

In this paper we have described a virtual-machine based web application platform for public displays that makes it possible to deploy multiple Web applications on a network of heterogeneous public displays in a decentralized manner, reminiscent of the way applications are traditionally installed on desktop computers. We described the design of our platform, how it is setup on our network of public displays and shared some experiences we have gained from its usage among our public display software developers.

Our approach of utilizing virtualization and Web technologies has enabled us to homogenize our mix of heterogeneous public displays so that services can be deployed locally on the display's virtual machines. Additionally, deploying web applications locally on the displays also improves robustness against network or server failures, which is a risk in the more traditional centralized web application deployment style. Further, by using web technologies, we have been able to tap into the large developer base available and have been able to foster the production of a diverse range of third party services for our display infrastructure. Finally, by utilizing virtual appliances, which encapsulate a display's web application platform along with any locally deployed services, we have been able to ease the setup and administration burden of maintaining our display infrastructure.

We also believe that the approach we have outlined here is also, to some degree, applicable to the ubiquitous computing domain in general. This approach enables a more flexible way of utilizing deployed hardware and makes application development over a wider variety of devices more straightforward. We believe virtualization is a trend that will only gain more momentum in the future, in many different computing areas, including urban and ubiquitous computing.

6. REFERENCES

[1] Erbad, A., Blackstock, B., Friday A., Lea R., and Al-Muhtadi, J. 2008. MAGIC Broker: A Middleware Toolkit for Interactive Public Displays. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM '08)*. IEEE Computer Society, Washington, DC, USA, 509-514.

[2] Finney, J., Wade, S., Davies, N., and Friday, A. 1996. Flump: The flexible ubiquitous monitor project. *In Cabernet Radicals Workshop.*

[3] Linden, T., Heikkinen, T., Ojala, T., Kukka, H., and Jurmu, M. 2010. Web-based framework for spatiotemporal screen real estate management of interactive public displays. In *Proceedings of the 19th international conference on World wide web (WWW '10)*. ACM, New York, NY, USA, 1277-1280.

[4] McCarthy, J., Costa, T., and Liongosari, E. 2001. UniCast, OutCast & GroupCast: Three Steps Toward Ubiquitous. In *3rd International Conference on Ubiquitous Computing (UbiComp'01)*. Springer Verlag, Atlanta, Georgia, 332—345.

[5] Oikawa, S., Sugaya, M., Iwasaki, M., Nakajima, T. 2004. Using virtualized operating systems as a ubiquitous computing infrastructure. In *Proceedings of second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (May 2004)*. 109- 113, 11-12.

[6] Ojala, T., Kukka, H., Lindén, T., Heikkinen, T., Jurmu, M., Hosio, S., and Kruger, F. 2010. UBI-Hotspot 1.0: Large-Scale Long-Term Deployment of Interactive Public Displays in a City Center. In *Fifth International Conference on Internet and Web Applications and Services (ICIW)*. pp. 285-294.

[7] Rudolph, L. 2009. A Virtualization Infrastructure that Supports Pervasive Computing, In *Pervasive Computing,* IEEE, vol.8, no.4, pp.8-13 (Oct.-Dec. 2009).

[8] Stahl, C., Baus, J., Brandherm, B., Schmitz, M., and Schwartz, T., 2005 , Navigational - and shopping assistance on the basis of user interactions in intelligent environments. *In The IEE International Workshop on Intelligent Environments*, (Ref. No. 2005/11059). pp. 182- 191.

[9] Storz, O., Friday, A., and Davies, N. 2006 Supporting content scheduling on situated public displays. In *Computers&Graphics*. volume 30, pages 681--691. Elsevier. (October 2006).