A Framework for the Development of Multi-Display Environment Applications Supporting Interactive Real-Time Portals

Chi Tai Dang, Elisabeth André
Augsburg University, Human Centered Multimedia, Department of Computer Science
Universitaetsstr. 6a, 86159 Augsburg, Germany
{dang, andre}@informatik.uni-augsburg.de

ABSTRACT

Advances in multi-touch enabled interactive tabletops led to many commercially available products and were increasingly deployed at places beyond research labs, for example at exhibitions, retail stores, or showrooms. At the same time, small multi-touch devices, such as tablets or smartphones, became prevalent in our daily life. When considering both trends, occasions and scenarios where tabletop systems and mobile devices form a coupled interaction space are expected to become increasingly widespread.

However, application development or research prototypes for those environments will foreseeable require considerable resources when considering nowadays heterogeneity of device platforms and the functionality to establish a connected interaction space. To address these concerns, this paper discusses challenges and answers questions that arose during design and implementation of the Environs framework, a multi-display environment software framework that eases development of interactive distributed applications. In particular, Environs enables applications utilizing video portals that put high requirements on responsiveness and latency.

Author Keywords

Multi-display environments; portal; latency; design; tabletop; mobile devices; tablets.

ACM Classification Keywords

D.2.6 Programming Environments: Interactive environments; H.5.2 Information Interfaces and Presentation: Miscellaneous

INTRODUCTION

The last decade has shaped the landscape of mobile touchscreen devices tremendously by generating a variety of device classes such as small smartwatches, smartphones, paper sheet sized tablets, or touch enabled notebooks. Even intermediate form factors, such as notepad sized "phablets" or hybrid devices with detachable keyboards (e.g. Microsoft Surface, Asus Transformer Pad) emerged and the variety of device classes is expected to develop further since the technology has matured out of its infancy and variations can be manufactured easily. Along with larger screen sizes, the equipment of those devices, such as CPU, GPU, connectivity features, or sensors, has been improved constantly making them nowadays well suited for demanding applications. Due to this trend and along with drop in cost and high availability, those devices meanwhile became prevalent in daily life. Hence, people got used to absolute input devices [11] in addition to the relative input of traditional mice, which also alleviates comprehension for interactive tabletop interaction.

At the same time, proliferation of large immobile touchscreen devices, such as interactive tabletops, has risen at smaller paces but yet constantly. More and more interactive tabletops became commercially available (e.g. Microsoft Surface tabletops[1], SmartTech SMART Table[2], ideum Touch Tables[3]) and increasingly deposited at public places beyond research labs, for example museums, showrooms, customer service places, or airports. Considering both trends, environments composed of multiple mobile displays working together with interactive tabletops are expected to become increasingly commonplace, also at home. Such environments, called MDEs (Multi-Display Environments), promise for novel applications, usage scenarios and interaction techniques spanned across multiple displays [22, 23]. A quite challenging example for an MDE application is to establish wireless video portals between devices which is the primary application domain of this work, see Figure 1. Portals replicate part of a large workspace to the smaller workspace on a mobile device at hand, for instance to enable a world-in-miniature view [7], to create a virtual loupe [26], to enable collaborative visual exploration [17], or, more general, as an extension in collaborative interactive spaces [1, 8, 13]. These recent research efforts indicate a growing interest of the HCI community in real-time video portals for interactive spaces or appropriate interaction techniques, which essentially require applicable and scalable MDE and portal realizations for todays mobile devices.

EICS 2014, June 17–20, 2014, Rome, Italy.

ACM 978-1-4503-2725-1/14/06 ...$15.00.
http://dx.doi.org/10.1145/2607023.2607038

[1] http://www.pixelsense.com (Apr. '14)
[2] http://smarttech.com/ (Apr. '14)
[3] http://ideum.com/products/multitouch/ (Apr. '14)

Figure 1. Interactive Real-Time Portals.

On the other hand, with a foreseeable heterogeneous device environment comprised of different kinds of mobile device and tabletop platforms, it is also a hassle to establish the functionality and infrastructure required to couple devices, identify devices and users, perform and synchronize distributed application logic, and establish interactive portals. This is not only an issue for research prototypes, but also for engineering of commercial interactive applications for MDEs. Instead of developing all the aforementioned functionality from scratch for each application, those best fit into a framework that can be used for application development. Such a framework help reduce development effort and time and optimize the process of engineering interactive application prototypes for research studies or products for consumer market.

To address these issues, we have implemented a software framework called *Environs* aimed at alleviating the development of MDE applications. The framework supports nowadays heterogeneous device environments and particularly addresses *low latency* and *high resolution video portals* for interactive applications. It consists of self-contained platform specific libraries that manage available application counterparts within the MDE, dynamically couple each other, establish video portals, and enable user interaction and applications spanned across multiple displays.

This paper gives an overview over the Environs framework and answers questions that we encountered in designing and implementing the framework, for example how to design the architecture/infrastructure, what components are required, how to couple application counterparts and establish communication, or how to distribute responsibilities over components. We also present two example applications that employ our framework to enable research for interactive video portal applications within MDEs. Even though parts of Environs provide service concepts typical of middlewares, we use the term framework for Environs because of the more general meaning of frameworks.

RELATED WORK

The research literature related to MDEs provides studies for interaction techniques [5, 13, 19, 22, 28], interaction metaphors and gestures [23, 24, 27], or example applications [7, 17, 26]. However, no work has yet considered the requirements and challenges on appropriate application frameworks. In particular, interactive video portal applications for MDEs have a strong demand on low latency for the presentation. Therefore, we discuss related work in terms of interactive portals in conjunction with latency issues in the light of touch interaction.

Interactive Portals

A large body of research for interactive portals draws on user interface and interaction metaphors, for example a toolglass [6], a peephole [9, 10, 29], or a magic lens [6, 10, 21]. They applied mobile augmented reality techniques to enable a portal, where the device's back facing camera was used to capture the portal source in order to augment the captured portal and ultimately to display the result on the device's display [9, 10, 21]. While this approach works well for the underlying metaphor, it also pose limitations of the application area. For example, navigation or zoom interaction is intrinsically tied to the mobile device's physical position and orientation.

Alternative portal approaches include having the whole portal content preloaded on the mobile device as proposed by Yee et al. [29] or restricting the portal content to geometrical drawings as demonstrated by Holmquist et al. [12]. Only few realized a portal for mobile devices that overcome the former described limitations [7, 13, 25, 26], however, they still suffer from restricted applicability for mobile devices. For example, Tsao [25] or Baudisch [3] facilitated portals based on VNC[4], which was originally designed to transmit screen captures on an event triggered request mechanism. Thus, VNC is not particularly designated for real-time streaming a portal in video quality.

Latency

Besides visual quality and applicability of portal implementations, latency is an equally important quality. In this work, we define latency as the duration for changes on the portal source to be visible on the portal destination. While prior works neglected latency issues, the impact of latency on user interaction and user experience increasingly became the focus of attention of recent research efforts [2, 4, 14, 15, 20], emphasizing the negative effects of high latency on task performance and error rates.

Since early work in 1968 by Miller [18], the "100ms rule of thumb" has been widely asserted for an upper recommendation for GUI feedback to *seem instantaneous*, whereas the evolution of technology educed increasing performance gains of mobile processors and new forms of devices, applications, and corresponding interaction techniques. Consequently, researchers focused again on system latency, for example, Jota et al. [14] studied the effect of latency in direct-touch pointing tasks and showed how task performance significantly decrease and error rates increase as latency increase. Thus, reinforcing an earlier Fitt's law study of MacKenzie [16] who identified *latency as a major bottleneck for usability*. Ng and colleagues [20] proposed to explicitly consider latency in user

[4] http://www.realvnc.com (Apr. '14)

interface design to cope with system latency. In addition, latency also has an effect on user experience where users perceive lower latency as more responsive [2]. Overall, portal implementations suffering from high latency in visualization and interaction not only has a negative effect on task performance, but also becomes annoying for users which in turn declines user experience [2].

CONTRIBUTION

The contribution of this paper for the research community is twofold. Firstly, we address questions regarding design, architecture, and implementation, that arise when engineering frameworks for MDEs supporting high resolution and low latency video portals. Secondly, we describe the software architecture and implementation details of our approach to enable interactive applications for MDEs. This work seeks to help advance research for interactive portal applications in MDEs that also account for user experience in which user expectations on applications rise with increase of mobile device performance.

EXAMPLE APPLICATIONS

Before describing concepts and technical details of Environs, this section aims at giving the reader an impression of the framework's functionality by depicting example applications and scenarios. Thus far, we have realized two example applications employing the Environs framework to prove the usefulness of the framework's capabilities and its advantages in terms of easy integration as well as reduced development cost. Those applications also served to conduct research for appropriate interaction techniques in MDEs through interactive portals.

MediaBrowser

The MediaBrowser is a distributed application consisting of an application for tabletop surfaces and mobile devices. The applications are designed for collaborative reviewing or examining of media data on large tabletop displays. They aimed at studying interaction techniques that best support collaborative tasks within such an interactive MDE scenario.

Users who run the MediaBrowser on their mobile device are first presented a list of available MediaBrowser devices and tabletops that were detected by the framework. The framework updates this list automatically allowing users to participate in an ad-hoc fashion. Upon being presented with the list of application counterparts, users may transmit different kinds of media data, such as images or text-documents, with each other through the MediaBrowser. Media data transmitted to the tabletop are immediately shown on the tabletop display where all media objects can be manipulated through multi-touch input.

In Figure 2, the MediaBrowser shows multiple images which can be moved or scaled with multi-touch gestures. Bystanders who want to take part in the collaborative task just place their device on the tabletop surface whereupon a video portal between the devices is automatically created. As depicted in Figure 1 and 2, the mobile devices appear as transparent windows that show the tabletop surface area occluded by the device. In order to detect mobile devices on the tabletop surface,

Figure 2. Example application: Media Browser.

every mobile device has a Microsoft Surface supported visual byte tag[5] attached at the backside, see Figure 3. The portal

Figure 3. Microsoft Surface byte tag attached at the back side of a mobile device.

stays connected and updated if a device is lift off from the tabletop surface allowing users to virtually pick up a piece of the tabletop surface by means of their personal device. Users can further input multi-touch gestures on their mobile device which are directly applied to the media on the tabletop surface. By this way, multiple users collaboratively interact with the tabletop surface in parallel while the presentation of the large tabletop surface does not suffer from space conflicts or occlusion issues due to too many arms and hands of collaborators. However, users are still able to interact on the tabletop surface if the collaborative task requires for. Development of the application hugely benefited from the Environs framework's functionality allowing developers to focus mainly on user interface and presentation related logic.

Public Display Toucher

The second example application *Public Display Toucher*, as shown in Figure 4, consists of an application for public displays and mobile tablets. This application demonstrates how users may operate large public displays by means of a tablet's input capabilities. Users connect to a public display through an according tablet application which enables them to transfer media data to the public display's desktop or operate through a video portal. Upon creation of a video portal, the portal's

[5] http://www.microsoft.com/download/en/details.aspx?id=11029 (Apr. '14)

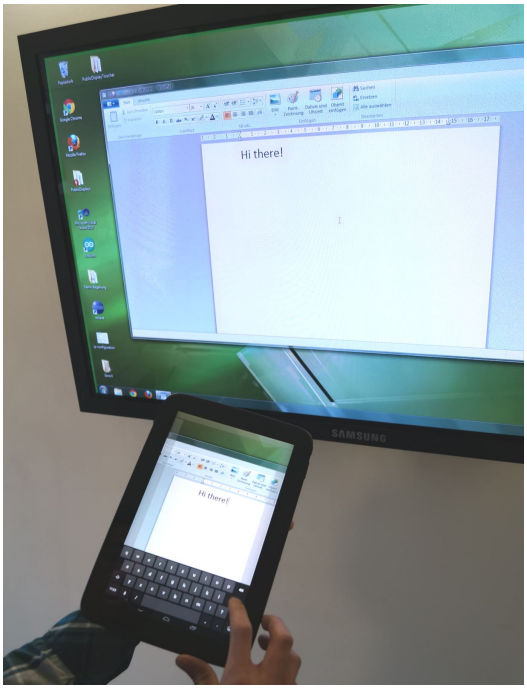

Figure 4. Example application: Public Display Toucher.

position and size can be adjusted through performing three finger multi-touch gestures. The public display can be controlled by means of single touches on the tablet which are translated to mouse clicks on the public display's user interface. Furthermore, key input on the tablet's virtual keyboard are put through to the public display and translated into regular key events. By this means, the tablet takes over the public display's mouse and keyboard allowing users to operate the public display's desktop, for instance to start applications, perform mouse clicks, or enter text. Based on this basic functionality, multi-touch enabled applications may be started on the public display which may be controlled further with the tablet. Just as with the MediaBrowser, development of the application mainly focused on user interface related logic. In addition, the application for the public display included logic for translating key messages from tablet devices into Microsoft Windows key events.

CHALLENGES FOR MDE FRAMEWORKS

When engineering a framework for the MDEs in focus of this paper, questions arise such as how to design a framework to support different platforms without implementing, managing and developing the whole framework for each platform separately, or how to structure and distribute responsibilities for a reasonable architecture, or how to manage the participating devices and applications in case of multiple different MDE applications running in the same physical MDE?

We address such questions regarding architecture, design, and implementation by first identifying essential requirements on the MDE framework in question and then presenting our approach. The following requirements are also considered challenges to tackle within the engineering process:

1. *Heterogeneity of platforms*. From a technical point of view, a big challenge for a framework is to support different heterogeneous device platforms. For each platform, the framework's functionalities have to be implemented based on platform specific development requirements. For example, each platform requires developers to use a specific programming language, such as Java for Google Android, C# for Microsoft Surface, or Objective-C for Apple iOS. Moreover, each platform provides access to the functionality through different APIs, packages and methods.

   Supporting different device platforms is not only reasonable for commercial development, but also for scientific research in case of distributing the framework or framework-based applications to fellow researchers who may not necessarily use the same device platforms.

2. *Efficiency and latency*. A framework must provide efficiency and low latency for network transfers and for framework logic. In particular, video portals require fast packet transfers and fast processing of the video stream in order to enable low latency. The lower the latency, the more responsive an application appears, which directly affects user experience [2].

3. *Flexible device management*. Finding and managing available devices must support MDE scenarios, where devices take part in an ad-hoc manner and may vanish suddenly. Devices have to identify themselves to each other and approve or deny connection requests as well as handle connections from multiple devices in parallel.

4. *Sensor support*. Nowadays devices are richly equipped with sensors, such as touch sensor, accelerometer, compass, or gyroscope. In order to offer novel interaction experiences, such sensors are often integrated into interaction with MDE applications. Therefore, a framework has to provide the support and infrastructure to retrieve, transport, process, or consume the sensor data.

DESIGN OF ENVIRONS

Environs is a software framework designed to aid development of distributed applications for MDEs with support for nowadays heterogeneous device platforms. Developers of those applications are given a set of self-contained platform specific libraries of which they include the library targeting their device's platform. In the current development state, Environs supports the platforms Microsoft Windows, Google Android, and Apple iOS. All libraries provide a consistent API across all platforms to custom applications for accessing the functionality of the framework.

Environs' functionality was designed to be implementation agnostic to the application logic layer which means that applications don't have to care about how to detect other devices or transfer files to them. Applications only need to invoke calls of the framework's API which can succeed or fail. The framework's functionalities so far include device and environment management, management of connections to other devices, transfer of files or binary data, communication between application instances by text or binary messages, real-time streaming of touch contacts to other devices and conducting touch

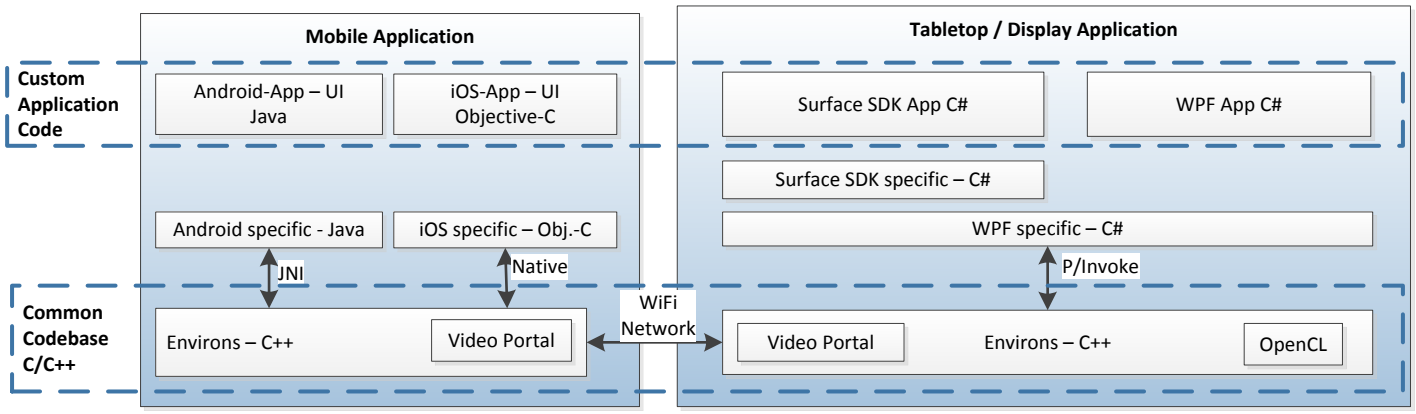

Figure 5. Environs libraries' code distribution.

events on destination devices, real-time streaming of sensor data to other devices, and real-time streaming of customizable high resolution video portals.

Code Distribution

Each library is comprised of a native component written in C/C++ and a platform specific component developed in the particular platform language, that is either C#, Java or Objective-C as outlined in Figure 5. Most part of the libraries, that is the aforementioned functionalities including image processing and video encoding, or gesture recognizers, share the same source code. This *common code base* is developed as portable C/C++-code and compiled as native component in order to *benefit from reduced development time*. This code distribution architecture addresses the *challenge (1)* and allows for the majority of the framework to be developed on the common code base for all platform specific libraries, for instance adding new or extending functionalities, modifying transport protocols, or debugging. In order to add support for a new platform, a platform specific API layer has to be written that connects the common code base with custom application logic.

Overall, the common code base drastically minimizes the development cost required for maintenance and development whilst supporting heterogeneous device platforms. In addition, native code provides rich opportunities to realize high performance implementations and low latency optimizations, for example in terms of memory management, platform optimizations, or direct access to hardware components which addresses the *challenge (2)*. The platform specific component of the libraries, that is the tiny slices above the common code base in Figure 5, primarily implements a consistent framework API across platforms supporting quick and easy integration into custom applications. Custom application logic (located at the top level in Figure 5) simply add the framework library and access MDE functionalities through the framework API. This code layer distribution best suits the needs of the engineering process for interactive applications supporting heterogeneous MDEs.

Concept of Environs' Environments

The Environs framework enables multiple logically separated application environments to coexist within the same physical MDE as sketched in Figure 6. Each application environment is identified by a project name and an application name which allows scenarios where, for example, multiple different Environs-enabled applications run on a tabletop display while the framework guarantees that each application sees and interacts only with other corresponding applications running within the same MDE. This logical separation helps the application logic of custom applications in communication with other application instances because different applications potentially communicate with disparate communication protocols. For example, a tablet application that exchange videos or images with a tabletop display does not need to connect to a public display application in the same MDE that runs a blackboard application.

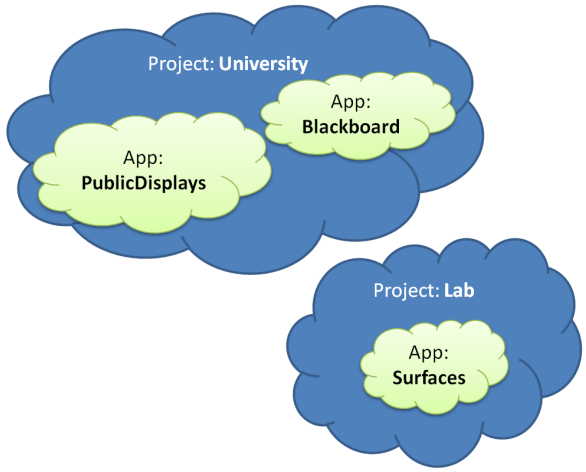

Figure 6. Multiple application environments distinguished by project/application names.

Each application can request to see and communicate only with applications either with matching project and application

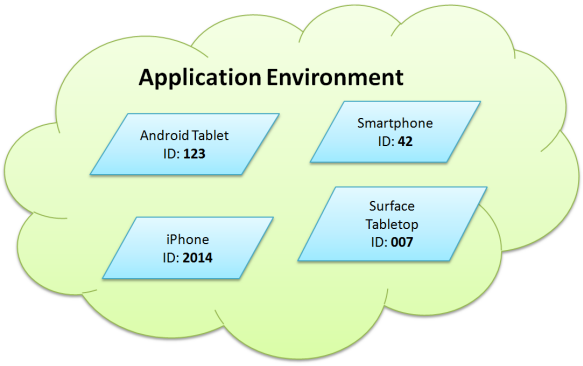

Figure 7. Device IDs in application environments.

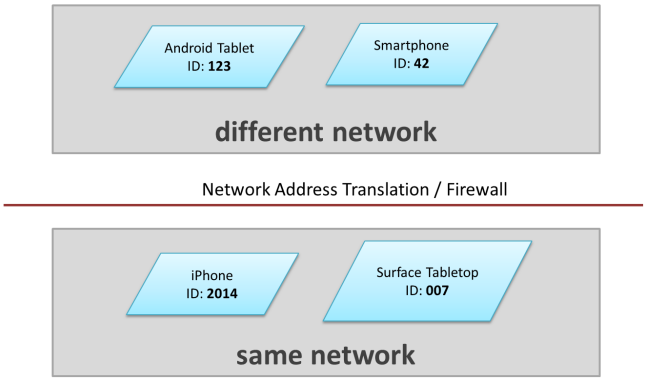

Figure 8. Supported network scenarios.

name, or with matching project name. The additional project name enables hybrid applications that provide the functionality of multiple different applications together and therefore needs to see application instances of all application names within the same project. In order to identify devices within an application environment, each device is assigned an ID (numeric 32 bit value) as sketched in Figure 7. This ID must only be unique within an application environment, that is the same device may use a different ID for a different application environment. The concrete assignment or partitioning of IDs can be chosen application dependent. For example, IDs lower than 1000 are assigned to tabletops and IDs greater than 1000 are assigned to mobile devices.

FRAMEWORK COMPONENTS

The Environs framework includes the following main components: (1) API/core component; (2) device and application management (mediator component); (3) network connectivity, communication and data transfer (network component); (4) touch event/sensor data handling which also includes gesture recognizer modules.

The API component (1) represents the linchpin for custom applications to access the framework functionality. It receives API calls from custom applications, translates them to commands, manages and delegates them to appropriate framework components, and informs the caller about results of its call, whereby results of longer lasting tasks are asynchronously notified to the caller. Each component mutually makes use of the services of the other components if appropriate. For example, the touch/sensor component makes use of the network component to send its data to destination applications. In addition to that, we implemented a subcomponent for streaming of video portals which also makes use of the services of the main components. For example, the network component is used to transfer stream packets to other devices. The following sections provide a description of each component.

Mediator Component

An essential requirement to realize distributed MDE applications is to detect corresponding application instances and its associated details for network connectivity within MDEs where application instances appear or vanish in an ad-hoc manner. The mediator component is responsible for those tasks and addresses the *challenge (3)* by providing a list of currently available application instances. This list covers application instances for the following two scenarios as sketched in Figure 8: (1) the devices are in the same network and are able to contact each other directly; (2) a device is located in a different network and is not directly accessible. One example for the latter case is when mobile devices are connected to a wireless private network provided by a NAT[6] router whilst the corresponding large static display is connected to a different network provided by a LAN. Such scenarios often occur in research labs where multiple wireless private networks for dedicated projects or applications coexist.

Devices in the same Network

The mediator component in each application instance employs broadcast messages to maintain a list of all available application instances within the same network. Broadcast messages are received by every device on the same network. Therefore, they suit well to exchange application instance IDs with each other. An application instance ID contains the device ID, the project / application name and connectivity details. Upon start of the mediator instance, a greet ID message is broadcasted to the network to tell other instances about the existence and availability of the application. Every mediator instance that receives a greet message broadcasts its own ID message to inform the availability of itself, thus update the alive status to other mediator instances. On exit of the mediator instance, a bye ID message is broadcasted to the network in order to tell other mediator instances about the absence of itself.

Devices in different Networks

In order to support environments where an MDE is comprised of multiple private networks which have no direct route to each other but have routing to external addresses through NAT, the mediator component builds on a mediator server instance which all devices have access to. This setup is optional and only required for connections across different networks. Mediator instances of applications can register at a mediator

[6]Network Address Translation

server instance which maintains a list of registered application instances with their according application ID. This list is retrieved by the application's mediator instance and augments the list of available application instances with those not directly available. Furthermore, the mediator server instance helps application instances connect each other across different networks by means of the mechanisms STUNT[7] for TCP and STUN[8] for UDP.

Network Component

The network component is responsible for establishing connections to other devices and transferring messages and files or data buffers between devices. It is designed to support interactive systems through selectively distributing the data to be sent to appropriate transport channels. The most important requirement of responsive interactive applications is that the communication of custom application logic with other application instances (e.g. status updates, commands or requests) must not be delayed as best as possible. Furthermore, real-time data such as touch events or video stream packets have to be transferred as fast as possible without affecting the application logic's communication. Therefore, the network component operate with different transport channels as described in the following section. In preparation for establishing connections, the network component interacts with the mediator component to retrieve connection details for the mediator scenario (1) or to employ the mediator server instance's service to initiate STUNT/STUN channels in case of the mediator scenario (2).

Connections between Devices

Upon successful connection with a device, the network component has established the following channels: (1) TCP main channel; (2) TCP bulk data channel; (3) UDP interactive channel. The first channel (1) serves as communication channel for custom application logic as well as framework communication with other framework instances, for example to start/stop a video portal or handshake options for the video portal. The transfer of large files or data buffers potentially takes more time and would induce lag and wait times on communication of the application logic if conducted over the main channel. Therefore, such transfers are handled over the bulk data channel (2) to ensure responsiveness of the interactive application. The UDP data channel (3) is used for touch events and sensor data due to the timely constraints of such kind of data regarding interactivity. For example, users would not notice missing intermediate touch events during a touch gesture or missing intermediate compass values when rotating a tablet. However, they severely notice the lag in visualization of the effect of such events. For example, if touch events are delayed due to retransmissions of past touch events, then their happening as well as the according visualization occurs timely disrupted on the destination.

In addition to the three channels, a fourth portal channel is established on demand for a video portal. Applications can choose which transport protocol (TCP/UDP) the portal channel shall use. This additional channel is required because of the real-time character and amount of data of video portals where the receiver is typically flooded with stream packets. Therefore, those data would disturb and negatively affect the other channels.

Handshake

Upon successful connection of the main channel (1), the devices exchange their capabilities, such as device type (tablet, smartphone, tabletop, display, etc.), screen dimensions in pixels, display density in dpi, support for video formats, availability of sensors, or socket buffer sizes. Those capabilities are autonomously detected by the framework and used to optimize the transport channels or to automatically derive parameters. For instance, if the custom application logic has not specified the size of the video portal on a tabletop, then the video portal's size is calculated to match the area that the device covers on the tabletop surface by means of pixel and dpi values.

Touch/Sensor Component

The touch/sensor component is responsible for putting through received events to the platform specific layer as fast as possible where the events are further handled depending on the particular platform. For instance, on tabletop systems, touch events are injected into the touch system and thereby appear as regular touch events. The second responsibility of the this component is to keep the event states consistent, that is to compensate for missing events due to packet drops or to drop old events that were outrun by newer events. For this purpose, each exchanged event carries an incremental sequence number and the current touch/sensor state is transmitted once every second. Both help the component detect missing or outdated packets. This mechanism is the same as employed in the TUIO protocol[9].

Furthermore, this component supports gesture recognizer plug-ins which are feed with the current event states on each change of the event states. If the plug-in has recognized a gesture, then a plug-in defined gesture string is put through to the platform specific layer which may consume the gesture event or pass it on to the custom application logic. Based on this infrastructure, we have implemented a recognizer plug-in that enables three finger touch gestures for scaling the video portal's size (pinch gesture) or moving the video portal's position (pan gesture) on the tabletop surface.

Smart Portal

In order to enable video portals, we have implemented a subcomponent called *Smart Portal* which was designed to provide a high quality, high resolution video stream optimized for low latency. *Smart Portal* replicates part of a source window, such as the application visualization of an interactive tabletop, to the application window of a mobile device. The framework automatically renders the video stream to the window background of an application window specified by the user application. Thereby, developers can build portal applications as regular applications taking advantage of operating system widgets without the inclusion of additional external

[7] RFC5382 http://tools.ietf.org/search/rfc5382

[8] RFC5392 http://tools.ietf.org/search/rfc5389

[9] http://www.tuio.org/

stand-alone applications. The following design elements of the subcomponent are decisive contributions to achieve low latency and high resolution portals:

1. *Video compression* is used to minimize latency induced by network transport. *Smart Portal* employs the high efficiency video codec H.264 enabled through the opensource implementation libx264[10] based on the encoding profile "superfast/zerolatency".

   This library was natively compiled for all platforms and used for encoding and decoding. However, software decoding is only used as a fall-back case. The framework on mobile devices makes use of hardware decoding if available, which unburdens the CPU from video decoding while application and framework logic fully benefits from the CPU.

2. *GPU acceleration*: Virtually all nowadays graphic cards support scientific computation by means of the standardized OpenCL[11] API. For this reason, computational intensive preprocessings of the video stream's source images are performed on the GPU for which we developed optimized OpenCL kernels.

Real-time GPU Pipeline

Creation of the source portal stream includes several image processing steps such as comparison of subsequent frames, rotation by a given angle, bilinear scaling, and image format conversion from RGB to YUV. Comparison of subsequent frames is highly recommended and help reduce the system load by skipping all remaining filters in case of equality. Bilinear scaling is required since *Smart Portal* supports arbitrary portal source sizes which need to be scaled to the desired video stream resolution. Rotation by a given angle is a requirement for tabletop surfaces since mobile devices may be placed arbitrarily oriented on the tabletop. Finally, format conversion is a requirement for the H.264 encoding process.

All of the image processing tasks are moved from CPU to GPU because of two reasons. First, most of the time modern GPUs have only little workload caused by rendering the application user interface. Second, modern GPUs have multiple computing units where each unit can run a multitude of work items ($>= 256$) in parallel and extremely fast, thus process much more pixels of an image in parallel and much faster than the CPU.

Thereby, disburden the CPU results in preserved computing resources for the benefit of application and framework logic which further reduce system latency.

Scalability

In particular, this solution scales much better with increasing number of portals computed in parallel because of the available GPU computing units, where the GPU exploit the available units in parallel. In contrast, workload of the CPU increases with each additional portal resulting in potentially added latency or drop of frames.

[10] http://www.videolan.org/developers/x264.html (Apr. '14)
[11] http://www.khronos.org/opencl (Apr. '14)

Figure 9. Latency measure of a portal that covers the tablet's physical size on the surface.

Figure 10. Latency measure of a high resolution portal source.

Latency

We roughly measured the latencies of our portal approach by means of a simple technique wherein we superimposed the portal's source with a number that increased with every frame at 30 fps. This number is then photographed in the way that the portal's source and the portal's destination are on the same picture, see Figure 9. Based on the difference of the numbers and the frame rate of 30 fps, we determined the latencies shown in Table 1 for a Microsoft PixelSense 2.0 device and a Samsung Galaxy Tab 2.

Table 1. Latencies at 30 frames per second ($\pm 16ms$).

Video stream size	Min. (ms)	Median (ms)	Max. (ms)
294 x 454	66.6	99.9	133.2
844 x 1080	99.9	133.2	166.5

Table 1 lists latencies of a TCP portal for two video stream resolutions, i.e. 294x454 (pixels of the surface covered by the tablet's physical size, see Figure 9) and 844x1080 (full height of the surface tabletop, see Figure 10). For each row in the table, we took at least 30 pictures in sequence and determined the median, the lowest, and the highest latency. The average latencies are between 100ms and 133ms which we consider

low for such a complex MDE system. The difference between the video stream sizes are 33.3ms on average, which gives a strong indication that the main part of the latency was induced by network transport. Smaller video stream resolutions yield fewer video data to be transmitted which in turn can be displayed earlier on the portal destination. Therefore, the results revealed possibilities for further latency improvements through network optimizations.

FUTURE WORK

Environs is currently used to build MDE applications employing video portals in order to conduct research on appropriate interaction techniques and user interfaces. The further development of the framework includes distributing Environs[12] to fellow researchers and get feedback on the framework's concepts and functionality. We hope that our framework will help advance and conduct research for portal interaction within MDEs.

The code distribution of Environs allows for easy extension for further platforms, such as Linux, Apple MacOS, or Microsoft Windows Phone. Hence, we plan to add platform specific API layers for those platforms on demand.

CONCLUSION

Developing interactive video portal MDE applications requires engineering of essential functionality, such as device and environment management, reliable and responsive network communication, or enabling video portals. Considering nowadays heterogeneous device platforms, implementing those functionality for each platform can be elaborate and error prone. This paper addresses these issues by means of a multi-platform software framework that helps developers and designers focus on application and presentation logic. Engineering a framework for the targeted MDE application domain is quite challenging and rise questions in terms of architecture and design.

We have presented the Environs framework as an approach to tackle the engineering issues and described details which answer questions that arise within the engineering process. The framework's code distribution reduces the resources required to develop and maintain the framework while allowing the framework to exploit platform specific optimizations or adaptations. Our approach shows how to cope with multiple coexisting applications within the same physical MDE through the concept of application environments, how to realize high resolution video portals, or how to subdivide framework functionalities to components in order to enable interactive and responsive applications.

While this paper emphasized on MDEs with tabletop surfaces, the presented example applications demonstrate the general use of Environs also for other kinds of devices, such as large public displays. The examples also proved the usefulness of Environs and the concepts behind.

ACKNOWLEDGMENTS

The work described in this paper is co-funded by OC-Trust (FOR 1085) of the DFG.

[12]http://hcm-lab.de/environs

REFERENCES

1. Ajaj, R., Jacquemin, C., and Vernier, F. Rvdt: a design space for multiple input devices, multiple views and multiple display surfaces combination. In *Proceedings of the 2009 international conference on Multimodal interfaces*, ICMI-MLMI '09, ACM (New York, NY, USA, 2009), 269–276.

2. Anderson, G., Doherty, R., and Ganapathy, S. User perception of touch screen latency. In *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*, A. Marcus, Ed., vol. 6769 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg (2011), 195–202.

3. Baudisch, P., Good, N., and Stewart, P. Focus plus context screens: combining display technology with visualization techniques. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, ACM (New York, NY, USA, 2001), 31–40.

4. Bérard, F., and Blanch, R. Two touch system latency estimators: High accuracy and low overhead. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, ACM (New York, NY, USA, 2013), 241–250.

5. Bezerianos, A., and Balakrishnan, R. View and space management on large displays. *IEEE Comput. Graph. Appl. 25*, 4 (July 2005), 34–43.

6. Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, ACM (New York, NY, USA, 1993), 73–80.

7. Cheng, K., Li, J., and Müller-Tomfelde, C. Supporting interaction and collaboration on large displays using tablet devices. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, ACM (New York, NY, USA, 2012), 774–775.

8. Dippon, A., Wiedermann, N., and Klinker, G. Seamless integration of mobile devices into interactive surface environments. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, ITS '12, ACM (New York, NY, USA, 2012), 331–334.

9. Grubert, J., Morrison, A., Munz, H., and Reitmayr, G. Playing it real: magic lens and static peephole interfaces for games in a public space. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*, MobileHCI '12, ACM (New York, NY, USA, 2012), 231–240.

10. Henze, N., and Boll, S. Evaluation of an off-screen visualization for magic lens and dynamic peephole

interfaces. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, ACM (New York, NY, USA, 2010), 191–194.

11. Hinckley, K., and Wigdor, D. *Input Technologies and Techniques*.
12. Holmquist, L. E., Sanneblad, J., and Gaye, L. Total recall: in-place viewing of captured whiteboard annotations. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '03, ACM (New York, NY, USA, 2003), 980–981.
13. Jetter, H.-C., Dachselt, R., Reiterer, H., Quigley, A., Benyon, D., and Haller, M. Blended interaction: envisioning future collaborative interactive spaces. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, ACM (New York, NY, USA, 2013), 3271–3274.
14. Jota, R., Ng, A., Dietz, P., and Wigdor, D. How fast is fast enough?: a study of the effects of latency in direct-touch pointing tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, ACM (New York, NY, USA, 2013), 2291–2300.
15. Kaaresoja, T., and Brewster, S. Feedback is... late: measuring multimodal delays in mobile device touchscreen interaction. In *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*, ICMI-MLMI '10, ACM (New York, NY, USA, 2010), 2:1–2:8.
16. MacKenzie, I. S., and Ware, C. Lag as a determinant of human performance in interactive systems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, ACM (New York, NY, USA, 1993), 488–493.
17. McGrath, W., Bowman, B., McCallum, D., Hincapié-Ramos, J. D., Elmqvist, N., and Irani, P. Branch-explore-merge: facilitating real-time revision control in collaborative visual exploration. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, ITS '12, ACM (New York, NY, USA, 2012), 235–244.
18. Miller, R. B. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, AFIPS '68 (Fall, part I), ACM (New York, NY, USA, 1968), 267–277.
19. Nacenta, M. *Cross-display object movement in multi-display environments*. PhD thesis, University of Saskatchewan, http://library2.usask.ca/theses/available/etd-01062010-123426/, 2009.
20. Ng, A., Lepinski, J., Wigdor, D., Sanders, S., and Dietz, P. Designing for low-latency direct-touch input. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, ACM (New York, NY, USA, 2012), 453–464.
21. Rohs, M., Essl, G., Schöning, J., Naumann, A., Schleicher, R., and Krüger, A. Impact of item density on magic lens interactions. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '09, ACM (New York, NY, USA, 2009), 38:1–38:4.
22. Seyed, T., Burns, C., Costa Sousa, M., and Maurer, F. From small screens to big displays: understanding interaction in multi-display environments. In *Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion*, IUI '13 Companion, ACM (New York, NY, USA, 2013), 33–36.
23. Seyed, T., Burns, C., Costa Sousa, M., Maurer, F., and Tang, A. Eliciting usable gestures for multi-display environments. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, ITS '12, ACM (New York, NY, USA, 2012), 41–50.
24. Spindler, M., Stellmach, S., and Dachselt, R. Paperlens: advanced magic lens interaction above the tabletop. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, ACM (New York, NY, USA, 2009), 69–76.
25. Tsao, C.-L., Kakumanu, S., and Sivakumar, R. Smartvnc: an effective remote computing solution for smartphones. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, MobiCom '11, ACM (New York, NY, USA, 2011), 13–24.
26. Voida, S., Tobiasz, M., Stromer, J., Isenberg, P., and Carpendale, S. Getting practical with interactive tabletop displays: designing for dense data, "fat fingers," diverse interactions, and face-to-face collaboration. ITS '09, ACM (New York, NY, USA, 2009), 109–116.
27. Wallace, J., Ha, V., Ziola, R., and Inkpen, K. Swordfish: User tailored workspaces in multi-display environments. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, ACM (New York, NY, USA, 2006), 1487–1492.
28. Wilson, A. D., and Benko, H. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, ACM (New York, NY, USA, 2010), 273–282.
29. Yee, K.-P. Peephole displays: pen interaction on spatially aware handheld computers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, ACM (New York, NY, USA, 2003), 1–8.