Towards A Usable API for Constructing Interactive Multi-Surface Systems

Chris Burns, Teddy Seyed, Theodore D. Hellmann, Jennifer Ferreira, Frank Maurer
University of Calgary, Department of Computer Science
2500 University Drive NW
Calgary, Alberta, Canada, T2N1N4
{chris.burns, teddy.seyed, tdhellma, jen.ferreira, frank.maurer }@ucalgary.ca

ABSTRACT
Research into multi-surface systems goes back for more than thirty years, yet these systems have not been taken up in real-world settings. We believe the reason for the lack of adoption is that constructing multi-surface systems is costly and requires specialist knowledge of tasks related to device discovery, cross-platform interoperability, networking, and spatial tracking. These tasks represent a significant distraction from implementing features that actually matter to end users. While some APIs exist for supporting the set-up of multi-surface systems, they are directed at specialist developers. We propose to develop a highly learnable API for constructing multi-surface systems, which is targeted at non-specialists.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features – *Frameworks*

General Terms
Design, Human Factors.

Keywords
Multi-surface system, API Usability, API Design

1. INTRODUCTION
Multi-surface systems integrate multiple, heterogeneous computing devices into a single application solution. They rely on Natural User Interface (NUI) approaches like multi-touch interaction, gesture recognition and object tracking in 3d space for creating advanced user experiences (as opposed to multi-display environments that primarily focus on WIMP-based interfaces on multiple displays). There has been over thirty years of research into developing interactive multi-surface systems [1], yet examples of systems deployed in the real world are rare. We hypothesize that this is due to the cost and complexity of developing such applications for use in an interactive space. Currently, a developer creating such an application would have to be knowledgeable about device discovery, cross-platform interoperability, networking, spatial tracking, and other tasks. While these ancillary tasks are necessary for the application as a whole to work naturally, these tasks represent a significant distraction from implementing features that matter to end users.

To reduce the amount of this specialist knowledge that developers require to build applications for use in interactive spaces, developers need advanced and usable tool support. We believe this tool support should take the form of a highly usable and reusable application programming interface (API). This API should allow developers to focus on development of their applications rather than on ancillary tasks like interpreting video input, interpreting depth sensor data or advanced 3d graphics processing. The API should also include functionality for handling difficult, ambiguous cases for interaction in multi-surface interactive spaces – such as automatically determining the devices that different users intend to interact with. These situations will become increasingly common as interactive spaces increasingly incorporate multiple devices in one room, e.g., smartphones and tablets. It is important that this API be highly usable – especially in terms of learnability – so that it can be easily adopted and used by typical development teams.

In this paper, we discuss existing applications for interactive spaces, challenges to the development of such systems, and the characteristics of an API that would be better able to support their development. A brief description of our work towards the creation of such an API – along with a description of our plans to evaluate it – is also included at the end of this paper.

2. INTERACTIVE SPACES
Several definitions of multi-surface interactive spaces have been proposed. In 2011, Gjerlufsen et al. used the following to describe interactive spaces: "Multi-surface environments are ubiquitous computing environments where interaction spans multiple input and output devices and can be performed by several users simultaneously" [1]. However, in 2006, Shen et al. made use of a much more specific definition: "By using the term multi-surface, instead of multi-display, we emphasize the nature of many of today's interactive walls, tables, Tablet PCs, desktop displays, laptops and PDAs that often can be interacted upon in addition to be merely the visual display" [2]. We follow the latter definition because it allows us to focus on systems in which devices participate in both interaction and display of data. Few real-world systems actually fit under this description, i.e., systems that are deployed and in active use outside of research labs.

The research literature has many examples of prototypes of multi-surface systems created and used within research laboratories. The earliest example is i-Land, a system developed by Streitz et al. in 1999 [3]. This early interactive system included a tabletop and wall display. Many of the interactions with this system focused on transfer of data between devices without tracking those devices. Since then, interactive spaces have been extended to include tracking of people and devices in interactive spaces using motion

Conference'10, Month 1–2, 2010, City, State, Country.

Figure 1: "Pouring" data from an iPad to a tabletop.

capture systems like those produced by VICON[1]. The WILD room, for example, employs such a tracking system to support interactions based on the position of individuals and items in the interactive space [4]. Code Space is a system developed at Microsoft Research that uses depth-sensing cameras to support interactions between a digital tabletop and a large-format wall display [5]. This system employs touch gestures as well as gestures performed in the air to support collaborative meetings. Touch gestures are performed on the device, using the multi-touch capabilities of the device. Gestures performed in the air are physical gestures, such as waving or pointing. Both types of interactions are accomplished through integration of smartphones into the interactive space.

Real-world products are harder to find. One real-world example is the PBCave[2]. This system is commercially available, can be used for information visualization and is composed of a digital tabletop and wall display. It does not, however, support integration with now-ubiquitous smartphones and tablets to provide spatial tracking.

The lack of examples of interactive spaces in use outside the laboratory is troubling given the long history of research in this field. In order to address this issue, we should first ask: what makes development of applications for use in interactive spaces difficult? We explore these development challenges in the next section.

3. DEVELOPMENT CHALLENGES

We believe that there are two causes of the lack of real-world interactive spaces: (1) the cost of hardware commonly used in these systems, and (2) the complexity of developing applications on top of a multi-surface environment. We believe that both of these difficulties can, at least partially, be overcome with an API which supports the use of widely available hardware (cost reduction) and which developers can use to effectively incorporate their applications into an interactive space (complexity reduction). The API needs to provide a certain set of features. In the following, we'll discuss a minimum core of these.

3.1 Spatial Tracking

Spatial tracking allows a system to track the position of people, devices, and other items in the interactive space. Coupled with a model of the interactive space, this information can be used to support a variety of proxemic interactions. Currently, this is typically implemented using expensive, high-end VICON motion-tracking cameras.

In the past it was necessary to use such high-end technology to accomplish accurate tracking. With the introduction of devices like the Microsoft Kinect, it is now possible to get motion-tracking using inexpensive, widely-available technology. Using the Kinect together with orientation-aware devices – such as Apple's iPhone and iPad – it is possible to track both the position and orientation of these devices. However, compiling low-level information about the position and information of a device into meaningful high-level information – for example, out of several possibilities, which device is a user trying to interact with – is a complicated task. These tasks require a significant amount of mathematical knowledge and also involve cross-platform communication between the mobile devices and the system tracking the position of mobile devices in the interactive space.

3.2 Heterogeneous Device Integration

Digital tabletops are large, touch-enabled surfaces that are good for collaborative work. Examples of these devices include the Evoluce One[3], the SMART Table[4], and the Microsoft Surface[5]. Of these, only the SMART Table is capable of running a non-Windows operating system. On the other hand, most smartphones and tablets run either iOS or Android operating systems while the Windows Phone 7 has only around a 2% market share worldwide[6]. In order to build a multi-surface system that supports common existing devices, developers are required to overcome the significant challenges that exist in trying to communicate information between these platforms. For example, transferring an image between two devices may require setting up network connections and handling the different file formats of the content. What is needed is a software architecture that supports device discovery and message passing between heterogeneous devices.

4. API DESIGN

In this section, we propose the two main features of an API for the development of applications that run in multi-surface interactive spaces. We believe supporting spatial awareness and communication between devices would simplify the implementation of these systems.

4.1 Spatial Awareness

The API must be aware of people and devices in the room. Spatial awareness is the ability to determine position and orientation over time – including mobile devices such as tablets and smartphones and fixed devices such as digital tabletops and wall-sized displays. The position of fixed devices within the room as well the layout of the room should be specified using a graphical user interface.

The location of mobile devices can be tracked using a Kinect while orientation data can be captured from the gyroscopes already built into many mobile devices. This data can be transmitted back to the system and integrated with the fixed position data to support spatial interactions. For example, when a user attempts to use a flick gesture to transfer data from a mobile device to a fixed device, such as in Figure 2, the system would use

[1] http://www.vicon.com/

[2] http://www.pbworld.com/capabilities_projects/visualization/cave.aspx

[3] http://www.evoluce.com/en/hardware/multi-touch_table.php

[4] http://smarttech.com/table

[5] http://www.microsoft.com/surface

[6] http://www.gartner.com/it/page.jsp?id=1848514

Figure 2: A "Flick" gesture transfering data from an iPad to a wall sized display

spatial information to select the destination device that the user intended.

4.2 Communication

The API must also allow all devices in the system to communicate without requiring developers to consider the platform-specific details of each device. This should be accomplished using HTTP as the transportation layer, and implementing a REST-ful interface for each device. To simplify communication tasks further, client libraries should be implemented for each common platform, such as Android, iOS and Windows. Developers could simply subscribe to communication events using the language common to their platform, for example, Java for Android or Objective-C for iOS.

Finally, the discovery of devices in the system should be handled using a standard protocol such as Bonjour. This would allow new devices to be added into the system dynamically without writing code to specifically handle each device.

5. SUPPORTED INTERACTIONS

Several types of interactions should be supported by the API to help designers and developers in the creation of multi-surface applications. Each interaction should be treated as a *first-class* event in the API. Developers will be able to assign certain functionality to be triggered on each device when it receives a specific event. For example, when a flick gesture is sent from one device to another, the system will alert the target device with a notification of this event. This design will allow designers and developers to add these interactions into their application with very little time and effort. This section further describes the different types of gestures that should be supported by the API.

5.1 Proxemic Interactions

Proxemic interactions have been explored in detail in the field of human-computer interaction. Centrally, Marquardt et al. presented Proximity Toolkit, an API for supporting proxemic interactions [6]. Developers can define functionality to be triggered based on the position of users in the system, the number of users being tracked, and other spatial information. This toolkit, however, relies on the use of VICON cameras. However, it is now possible to duplicate most of the functionality provided by this system by using the Kinect for motion capture. Given the difference in price between these systems, the API should focus on this new and promising device. By combining two Kinects we can deal with occlusion issues, specifically, *walk past* occlusions.

5.2 Physical Gestures

Physical gestures performed in the air are triggered by users moving their arms, hands, or fingers. For example, in some applications, users are able to control applications by pointing at icons as a means of choosing a selection or waving their hands to go back up one level of a directory hierarchy. The implementation of these gestures in an application is currently a very low-level, complicated process. In order to promote the use of these interactions in multi-surface applications, the API should provide high-level methods to allow these gestures to be used by designers and developers who do not have the specialist knowledge that would normally be required to implement these gestures.

5.3 Device Gestures

A device gesture is a gesture performed by a user physically interacting with or moving a mobile device. These gesture types can be subdivided into *control* and *information-passing* gestures. Control gestures are distinct from gestures which pass information.

5.3.1 Between-Device Control Gestures

Gestures can be used to allow one device to control another. Touch interactions with a smartphone could be used to trigger events on another device in the interactive space. An example of using gestures with one device to control another device is the Keynote[7] application for the iPhone. When the user swipes between slides on the iPhone, this changes the slides on the presentation screen. In the literature there is an example of using a multitouch tabletop to control the interface of a mobile phone, such as work by Olsen et al who presents a paper on "spilling" control from one a mobile phone to a multitouch tabletop. [8].

5.3.2 Information-Passing Gestures

Certain gestures can be used to support information transfer between applications running on different devices e.g. flicking summoning or pouring. Gestures performed with devices seem especially appropriate for this kind of task. Some work in the literature exists describing these gestures. Bhandari and Lim describe a system in which an entire smartphone is moved in a specific way to trigger interactions [7]. For example, they proposed the rotate gesture, which is triggered by rotating the device from the horizontal to the vertical position. A throw and pull gesture was shown by Döring et al. for communicating data between a tabletop and a mobile device, and this gesture was also used by Daschelt and Buchholz for communicating data to large public displays [8,9]. Finally, the chucking gesture – a one handed gesture using a mobile phone – was proposed by Hassan et al. as another simple data-transfer gesture [10].

5.4 Existing APIs

Gjerlufsen et al. [1] developed an API – which they refer to as a middleware layer – for developing multi-surface systems. This API provides much of the functionality required for developing these systems. It includes support for heterogeneous devices (devices on different platforms), a communication layer, and access to position information for devices in the interactive space. While these are desirable characteristics, the API was developed using a data-oriented programming model rather than the more common object oriented approach. A data-oriented approach is less common and thus, may make it more difficult for typical developers to use this API. The API is also built at a relatively low-level of abstraction. Both these issues make it difficult for non-expert developers to use. Further, it's not clear if the API could be used with the widely available Kinect rather than the VICON motion-capture system it was designed to use.

[7] http://itunes.apple.com/ca/app/keynote/id361285480?mt=8

Another existing API is the Proximity Toolkit developed by Marquadt et al. [6].This API allows developers to work with proxemic interactions, but it does not provide a communication layer or support for other interaction types.

6. STATE OF IMPLEMENTATION

We are currently developing a prototype multi-surface system. This prototype uses an Evoluce Tabletop, a SMART board and 2 iPads. The system also uses the Microsoft Kinect to gather positional tracking and iPad's internal gyroscope to provide orientation tracking. We are experimenting with several physical gestures for this system, such as flick, summon and pour, see Figure 1 and Figure 2. In the flick gesture, the users swipe across the iPad while pointing in the direction of the target device, triggering an image transfer between the devices. The summoning gesture allows users to pull content from a target, by rapidly pulling back the iPad. Finally, by placing the iPad over the target tabletop and rotating it on its side, the pouring gesture allows users to transfer content to the tabletop. We intend to extend this prototype into a complete API over the next several months.

7. FUTURE WORK

Our focus is on easing the implementation work involved in setting up multi-surface interactive spaces by allowing developers with non-specialist knowledge to do so. Therefore, we consider an API usable if it exhibits high *learnability*. That is, developers who are not networking specialists, for example, should be able to use the API to set up network connections between devices easily and efficiently. Similarly, all the tasks that are common to implementing spatial awareness and communication between devices should be supported by the API. The learnability of the API can be evaluated with user studies – having users carry out small but specific tasks with the API in a similar way to the study by Stylos and Myers [13].

8. CONCLUSION

This paper has suggested that a lack of real world adoption of interactive multi-surface systems is due to the lack of a learnable API that is accessible to non-specialist developers. Versions of the API need to be available on leading platforms such as Android and iOS for mobile devices and Windows for tabletops and wall displays. The API would need to solve ancillary tasks common to setting up any multi-surface system but unrelated to the core development goals of developers. These ancillary tasks include spatial tracking and communication. We propose that a highly learnable API targeted at non-specialist developers will reduce costs, in terms of effort and money, and make interactive multi-surface systems more widely available.

9. BIBLIOGRAPHY

1 Bolt, R. A. "Put-That-There": Voice and Gesture at the Graphics Interface. In *Computer Graphics and Interactive Technology* (Seattle 1980), 262-270.

2 Gjerlufsen, T., Klokmose, C. N., Eagan, J., Pillias, C., and Beaudouin-Lafon, M. Shared Substance: Developing Flexible Multi-Surface Applications. In *Human Factors in Computing Systems* (Vancouver, British Columbia, Canada 2011), 3383-3392.

3 Shen, C., Esenther, A., Forlines, C., and Ryall, K. Three Modes of Multi-Surface Interaction and Visualization. In *Human Factors in Computing Systems* (Montreal, Quebec, Canada 2006).

4 Streitz, N.A., Geißelr, J., Homber, T. et al. i-Land: An Interactive Landscape for Creativity and Innovation. In *Human Factors in Computing Systems* (Pittsburgh 1999), 120-127.

5 Beaudouin-Lafon, M. Lessons Learned from the WILD Room, a Multisurface Interactive Environment. In *French-Speaking Conference on Human-Computer Interaction* (Sophia Antipolis, France 2011).

6 Bragdon, A., DeLine, R., Hinkley, K., and Morris, M.R. Code Space: Touch + Air Gesture Hybrid Interactions. In *Interactive Tabletops and Surfaces* (Kobe, Japan 2011), 212-221.

7 Marquadt, N., Diaz-Marino, R., Boring, S., and Greenberg, S. The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies. In *User Interface Software and Technology* (Santa Barbara, California, USA 2011), 315-326.

8 Olsen Jr., D. R., Clement, J., and Pace, A. Spilling: Expanding Hand-Held Interactions to Touch Table Displays. In *Horizontal Interactive Human-Computer Systems* (Newport, Rhode Island 2007), 163-170.

9 Bhandari, S and Lim, Y. Exploring Gestural Mode of Interaction with Mobile Phones. In *Human Factors in Computing Systems* (Florence, Italy 2008), 2979-2984.

10 Döring, T., Shirazi, A.S., and Schmidt, A. Exploring Gesture-Based Interaction Techniques in Multi-Display Environments with Mobile Phones in a Multi-Touch Table. In *Advanced Visual Interfaces* (Rome, Italy 2010), 47-54.

11 Dashelt, R. and Buchholz, R. Natural throw and Tilt Interaction between Mobile Phones and Distant Displays. In *Human Factors in Computing Systems* (Boston 2009), 3253-3258.

12 Hassan, N., Rahman, M.M., Irani, P., and Graham, P. A One-Handed Document Sharing Technique. In *Human-Computer Interaction* (Berlin 2009), 264-278.

13 Stylos, J. and Myers, B. A. The Implications of Method Placement on API Learnability. In *International Symposium on Foundations of Software Engineering* (Atlanta 2008), 105-112.