A unified multi-touch & multi-pointer software architecture for supporting collocated work on the desktop

Kelvin Cheng, Benjamin Itzstein, Paul Sztajer and Markus Rittenbruch
NICTA, Australian Technology Park, Level 5, 13 Garden Street, Eveleigh, NSW 2015, Australia
{*firstname.lastname*}@nicta.com.au

ABSTRACT
Most current multi-touch capable interactive user interfaces for tabletop are built from custom toolkits that are decoupled from, and on top of, the "Desktop" provided by the underlying Operating System. However, this approach requires that each individual touch system build their own suite of touch capable custom applications (such as photo browsers), usually resulting in limited functionality. In this paper, we propose a software architecture for supporting and integrating multi-touch capability on existing desktop systems, where multi-touch and multiple single pointer input can be used simultaneously to manipulate existing application windows. We present an example implementation of this architecture on the Linux Operating System, demonstrating the possibility of using touch displays in a collaborative work environment on the tabletop. The paper concludes with lessons learnt and technical challenges from our experience.

Author Keywords
Software architecture, multi-touch, tabletop, multiple pointers.

ACM Classification Keywords
H5.3. Information interfaces and presentation (e.g., HCI): Group and Organization Interfaces: Collaborative computing, Computer-supported cooperative work.

INTRODUCTION

One aspect of the XYZ project focuses on collocated collaborative work. We are particularly interested in exploring novel interface technologies to improve productivity of face-to-face collaboration around the tabletop, specifically multi-touch capable tables.

Recently, multi-touch hardware technologies have gained increasing attention both in the research community [7] and the commercial world [13][17]. With these new technologies, it is now possible to allow the use of natural hand gestures to interact with virtual objects and applications collaboratively.

One of the main challenges in using these new multi-touch technologies is the fact that the major desktop Operating Systems (Linux, Mac OS and Windows) do not have the ability to natively support multiple pointers. Although recently Windows 7 [5] does provide limited multi-touch capability for specific applications and MacOS X does support multi-finger gestures on the trackpads for specific commands, a general underlying OS-wide multi-touch capability is not found.

For multi-touch application developers, the most common approach is then to develop custom applications that are written in an abstracted environment on top of the traditional desktop environment, to provide additional interpretation of multi-pointer inputs, as well as multiple hand inputs and gestures. However, this decouples the multi-touch input from the functionality provided by the underlying Operating System. Specific UI widgets must then be implemented to allow users to access the OS resources. This is both time consuming and redundant, and often results in simple applications that have limited functionality (e.g. photo browsing) compared to applications developed for traditional desktop environment (e.g. office productivity applications). Another drawback is that the user interfaces will not be consistent between these custom platforms provided by different developers.

In this paper, we propose an architecture that brings together the functionality and user interface provided by existing "legacy" applications – applications that only understands a single pointer – and the advantages that new types of multi-user-capable input methods provide. Specifically, our goal is to allow multiple pointers, touches and natural hand gestures to be easily integrated and used with the traditional Desktop environment, without the need for writing custom applications (yet still retains the ability to use custom applications on top of the Desktop). Our approach will also enable users to simultaneously scale, translate and rotate application windows to collaboratively work around a tabletop surface. We will demonstrate the use of our architecture with an example implementation on the Linux Operating System.

RELATED WORK

To position our work, we present a brief overview of literatures and related work involved in various aspects of building multi-touch capable applications.

Touch Libraries

Touch libraries detect touch events in the form of fingers, hands, and objects of various shapes and sizes. The hardware used determines the approaches, capabilities and limitations of various techniques. The most commonly found systems are vision-based. Community Core Vision (CCV) [2] is a popular open source multi-touch library (formerly referred to as tbeta and Touchlib). It uses various computer vision and image processing techniques on video input streams and produces tracking data such as fingertip coordinates and touch events such as finger down. It is designed to work with a range of vision-based hardware platforms. In constrast, DiamondSpin [18] is another well known touch library but it is specifically used for the DiamondTouch interactive display.

Touch Event Protocols

In many cases, the data about touch events that are sent from its underlying hardware level is specific to the individual hardware, and as such is only available to custom applications that utilize the manufacturer's SDK.

Recently, hardware devices and touch libraries have been adopting TUIO [11], a touch event protocol developed to allow for hardware-independence in tabletop systems. It allows developers to easily detect touches on a custom-built tabletop and exports TUIO events.

A different approach has been taken by the developers of SparshUI [19]. SparshUI is a development framework for multi-touch enabled applications, which includes a gesture server. SparshUI developers have created drivers for several types of hardware devices that communicate directly with the gesture server.

Multi-touch Interface Toolkits

The most common interface toolkits are either flash-based or OpenGL-based. The former group includes toolkits such as MERL's DTFlash [6], while the latter group contains MultiTouch Oy's Cornerstone [17]. There are also various other interface libraries such as flickOS [9] and Microsoft Surface SDK [13] using Windows Presentation Foundation (WPF).

In order to develop multi-touch capable applications, developers would pick one of these and use various available widgets to create a custom applications in a multi-touch supported environment. However, multi-touch support for traditional desktop applications is rarely seen.

Rotation and Multiple Pointers

Our main requirement for a collaborative tabletop architecture is to support the ability to allow several users to rotate application windows concurrently.

Metisse[1] is an X-based window system designed to facilitate low-fidelity window management prototypes and provides the ability to rotate application windows. Similarly, Compiz [3] is an open-source compositing window manager, which uses OpenGL to quickly draw windows on a system, as well as complex effects and also provides the ability for rotation. However, both of these are aimed at traditional desktop systems and lack the ability to support multiple pointers.

On the other hand, the Pebble system [16] allow multiple PDAs to control multiple cursors on the same application on a PC. However, the application had to be modified in order to allow for simultaneuos users. A more promising approach is the Multi-Pointer X Server (MPX) [8]. It is a modification to the X Server and provides the ability the use multiple cursor and keyboard foci.

As can be seen, a unified approach to providing tabletop window management for concurrent users is lacking.

Multi-touch Architectures

Echtler and Klinker [4] proposed a layered architecture in an attempt to abstract the varied multi-touch hardware used and allow them to be easily integrated into existing multi-touch software. However, they only focus on dealing with custom developed multi-touch aware software and do not address the issue of integrating with traditional desktop/mouse-based (legacy) applications.

In Figure 1 (left), we illustrate an architecture that generalized the structure used in the varied multi-touch capable systems such as in [2][17]. As can be observed, to allow multi-user touch input, they rely on custom developed applications in an environment separate from the underlying operating system. The use of legacy applications is often restricted to only one pointer (by means of a single touch point), and therefore restricted to one user.

THE UNIFIED ARCHITECTURE

Our goal is to allow easy integration of multi-touch hardware technologies on current Operating Systems, unifying multiple pointers, multiple touches and natural touch gestures for interacting with traditional (legacy) applications in the desktop environment.

An important distinction that must be noted is that we are not addressing the challenge of making legacy applications multi-touch capable, but rather we intend to support these single pointer applications so they can be integrated into a multi-touch environment, thus allowing multiple users to use multiple legacy applications simultaneously.

Figure 1 (right) illustrates an overview of our unified software architecture.

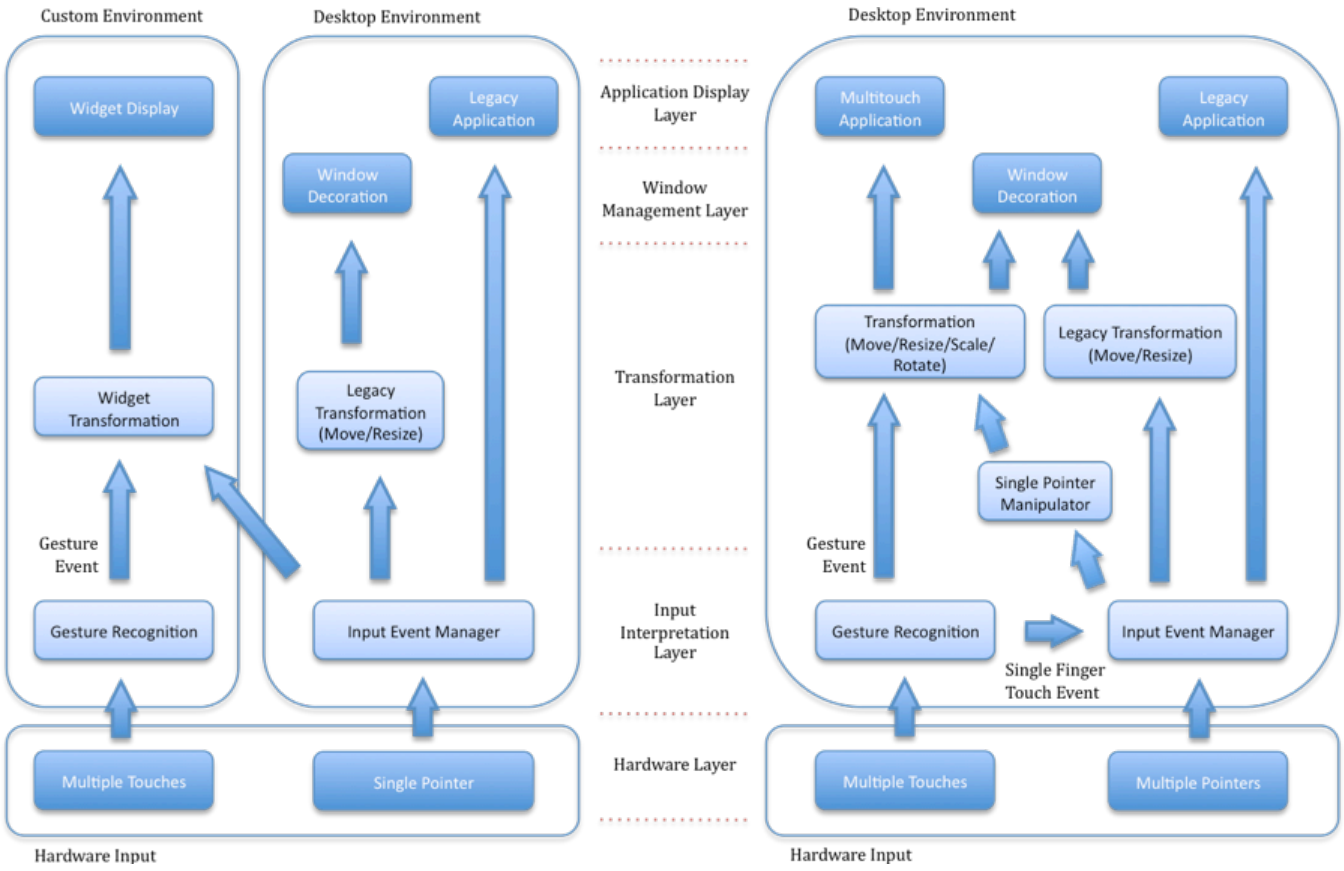

Figure 1. Left: Current architecture separates custom multi-touch aware environment and the underlying desktop. Right: Unified architecture for simultaneous multi-touch and multi-pointer input onn the desktop environment.

Gesture Recognition

This architecture is designed to support two main types of input methods: multiple pointers (e.g. computer mice) and multiple touch input (e.g. finger tip touches on multi-touch capable hardware).

Multi-touch input can be used in two different ways: (1) as mouse replacement for the selection of existing user interface elements such as buttons (often using just the one finger); and (2) as natural hand gesture, such as rotating or scaling virtual objects, or for issuing commands using more than one finger.

In order to distinguish these two input techniques, a gesture recognizer is used. From the multiple finger touch inputs, touch points are interpreted so that when only a single finger is used, it is recognized as a single finger touch event. For example, if two touch points are detected on the table, but are very far from each other, they can be interpreted as two separate single finger touch events. These are then fed into the input event manager where it is combined with other single pointer inputs (e.g. computer mice).

With the remaining multiple touch points, the temporal and spatial touch positions are interpreted as predefined gestures and are recognized as individual gesture events such as rotation or scaling. These are then passed onto the transformation module, where objects undergo the specified transformations based on the gesture events received. The transformed object is then displayed on-screen. These objects may be in the form of virtual objects in a custom multi-touch aware application, or legacy application windows themselves. In the latter case, it is accomplished by passing the transformation information to the underlying window decoration that surrounds the application.

Similar to the single finger touch events, the single pointer inputs are passed onto the input event manager, which is responsible for keeping track of position data for each separate pointer.

These pointers can be used in three different ways:

(1) as normal cursors for interacting with individual legacy applications (one for each application) and their legacy mouse based window transformation (e.g. dragging and resizing on borders)

(2) as single pointer/touch events for the multi-touch aware application

(3) (optionally) to provide additional functionality for tabletop usage, the architecture allows manipulating application windows, such as rotation and translation, with only a single pointer or touch. This is done within the single pointer manipulator. The process is made intuitive using on-screen widgets for each window. An implementation of this will be discussed in the next section.

The main advantage of this architecture is that it does not assume the legacy application itself knows anything about multiple pointers, nor touch inputs. All of these are handled by the layers below, which can then pass on the multi-touch events to applications that can apply them. In addition, users can use different input methods simultaneously. For example, one person can be using one finger as a pointer while their co-worker can be rotating a separate application using touch gestures (Figure 2). A third co-worker can be using a normal computer mouse.

Custom applications can still be run on top of all of this. With this architecture, it is also possible for using legacy applications simultaneously with custom multi-touch aware applications side-by-side.

Figure 2. A user using a single pointer via touch, and another using a five-finger rotation gesture.

IMPLEMENTATION

In this section, we describe how we achieved our implementation using this architecture on the Linux Ubuntu Operating System, illustrated in Figure 3.

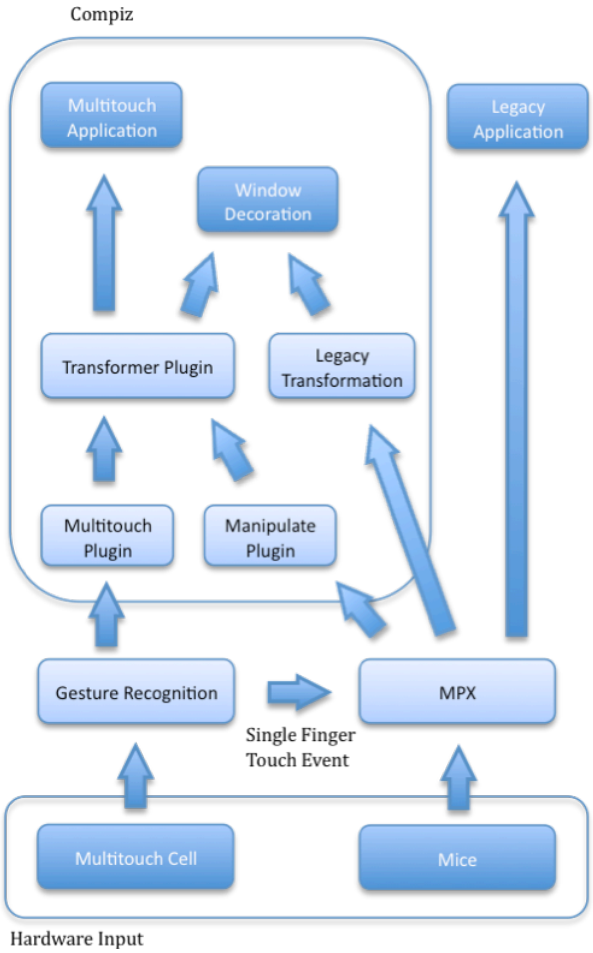

Figure 3. Our implementation of the unified software architecture.

Multiple Touches

The hardware used for our multi-touch architecture prototype was a MultiTouch Cell [17] which is based on the diffused infrared illumination. The accompanying software toolkit provides output data about the hands and fingers that are on the table. It also provides data about the orientation of fingers and hands.

Our implementation recognizes two simple gesture sets:

- A touch with a single finger from a hand is sent to the X server as a cursor, which only exists while the touch continues (Figure 4).
- A touch with multiple fingers from a hand is a gesture which will translate, rotate and scale the window underneath it. This gesture continues as long as at least one finger from that hand remains on the table (Figure 5).

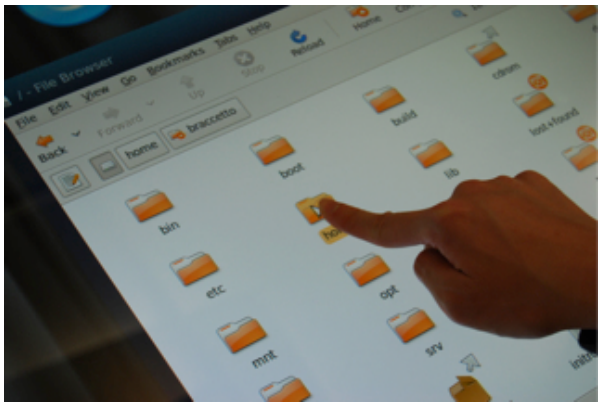

Figure 4. Single pointer using one finger.

Figure 5. Rotation using a five-finger gesture.

We implemented the transformation gesture applying the following parameters to the fingers associated with one hand:

- The translation was driven by the centroid of the fingers on the table
- The rotation was driven by the average angle of all the fingers on the table
- The scale was driven by the average distance between two fingers on the table

All these parameters were averaged over 10 frames to smooth the motion and were observed to be quick without introducing noticeable lag. In addition, throughout a gesture, the parameters would be changed by comparing the fingers common to two frames. This ensures that these parameters remain constant if a user adds or removes fingers from the gesture without moving any of the fingers on the table.

Manipulation and Transformation

For achieving transformations of legacy applications Compiz was used [3]. Compiz is an open-source compositing window manager, which uses OpenGL to quickly redraw application windows and provides complex effects. It has a plugin structure which allows for developers to add functionality and effects to windows on a Linux system.

We developed a plugin called *transformer*, which gives each window a scale and orientation attribute (both about the centre of the window). The *transformer* plugin draws the windows at their given scale and orientation, and provides interfaces to other plugins and applications to alter these attributes.

To implement the multi-touch gesture, we also built a plugin called *multitouch*, which receives events through Dbus (a multicast event bus for interprocess communication in Linux) about a gesture. *Multitouch* then alters the position, rotation and scale attributes so that the rotation and scaling is about the centre of the window, and updates the attributes through the *transformer* plugin.

The transformer plugin then applies a transformation matrix to the window before Compiz draws it to the screen (Figure 5).

Multiple Pointers

The Multi-Pointer X Server (MPX) [8] is a modification to the X Server that provides the user with multiple cursors and keyboard foci. This is in contrast to the mainstream windowing systems, X11, Quartz and Microsoft Desktop Window Manager (DWM), which only support a single cursor and keyboard focus [14].

Our primary interest with this system is the multiple cursor support, which means individual mouse cursors can move independently of each other. These cursors can be bound to hardware mice or controlled programmatically, by a touch event interpreter or remote control server for instance.

The MPX cursors interact with compatible multi-pointer aware applications and also with legacy applications to a limited extent. As mentioned earlier, we do not focus on bringing multi-pointer support to legacy applications, but rather allowing multiple pointers to simultaneously interact with different legacy applications.

In the context of the multi-touch and multi-pointer architecture, MPX is an important element for permitting collocated interaction. As can be seen in our implementation architecture (Figure 3), MPX pointers can interact with multi-touch and legacy applications, as normal mice do.

MPX cursors are created for all local hardware mice and single touches on the multi-touch table. When a new touch with a single finger is identified, an MPX cursor is created at that point and a left click is held down. The position of that cursor then tracks the position of the corresponding touch. Once the touch is removed, the cursor releases the click and is destroyed.

The Manipulate Plugin

The ability to manipulate a window through scaling, rotating and translating is quite intuitive when performing

multi-touch gestures. However, within a unified multi-touch & multi-pointer architecture there are times when a single touch transformation is beneficial.

To address this requirement, we implement the simultaneous rotation and translation algorithm (RnT) proposed in [12] as a left-click and drag mouse action. When RnT is active, the action is activated whenever a left-click begins within a window, in addition to the following special handles:

- A circular handle is located in the centre of each window that permits pure translation, as it has been shown to be intuitive [12].
- Triangular scaling handles are located in the four corners of each window, similar to [18]. Scaling is implemented by fixing the centre of the window and scaling the window to track the mouse or touch location.

This functionality has been implemented as another Compiz plugin: *Manipulate*. Being a plugin for the window manager allows it to interact with the window decorations, draw the manipulation handles on the window (Figure 6), intercept input mouse events in the window and, most importantly, transform the window.

Figure 6. The Manipulate plugin – triangles for scaling and circular handle for translating using a single touch.

A window can toggle between ordinary mouse input and the manipulate mode using a button on the title bar (Figure 7). The custom window decoration featuring this button was produced using a customized version of the Emerald Theme Manager.

In our architecture, the Manipulate plugin acts as a bridge between MPX cursor events and the window transformer (*Manipulate plugin* in Figure 3). When the *Manipulate* plugin is disabled, cursor events are passed through to the legacy application as normal.

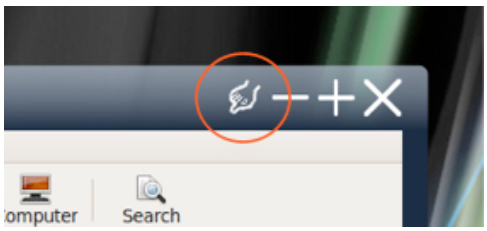

Figure 7. The Manipulate plugin Toggle.

This single touch manipulation will be useful for mouse users collaborating with multi-touch users on a shared surface. Despite only having a single point input, users can easily transform and position windows in a multi-touch-enabled environment.

The single touch transformation is also available for multi-touch users, as they are able to create mouse pointers with single finger touches, as explained in Section 4.3. This action is shown in progress in Figure 8. This unifies the abilities of a multi-touch user and a mouse users.

Figure 8. Rotation and translation in the Manipulate plugin using one finger.

Limitations

In implementing the unified architecture, we came across a number of technical challenges that hindered our progress. These tend to focus on the immaturity of the MPX software and unreliable MPX support as a result.

At the time of writing, the official integration of MPX and input redirection into X.Org Server (the reference implementation of the X Window System) has been delayed multiple times since its announcement.

The current experimental MPX patches are incompatible with some graphics drivers and suffer from noticeable

performance issues. In particular, ATI and Nvidia graphics cards are at the moment unsupported. Therefore we are forced to rely on Intel graphics, since they have open source drivers with full MPX support.

Unfortunately, Intel integrated graphics cards have performance limitations. The unofficial status of MPX and its considerable implications for changing the way applications behave also means it has no official support from Compiz. In its current state, Compiz will only give developers the location of a single mouse. A barebones MPX-aware version of Compiz is available but only provides the most basic functionality and has stability and performance issues [10]. Adding additional functionality requires complete rewrites of existing Compiz plugins. An implication of this is that both the manipulate plugin and window decorations are not fully multi-pointer aware. Although we are limited to only one mouse cursor in our implementation, multiple people are still able to use the multi-touch gestures to rotate and translate multiple desktop applications simultaneously.

DISCUSSIONS

So far, we have only discussed a number of technical challenges involved in implementing the unified architecture. There remain a number of outstanding issues yet unaddressed, in regard to using the traditional desktop in a tabletop collaborative setting:

- Focus: In the single user context, it is appropriate to have only one application in focus. However, in the groupware context, it may be necessary to have multiple applications in focus at once, operated by different users.
- Hand differentiation: The multi-touch gesture recognition relies on being able to distinguish between multiple single finger touches (for mouse cursors) and multiple touches from a single hand (to move, scale and rotate objects). However, the system has no way of knowing that two hands could be from the same user. Having such knowledge could allow more complex activities.
- Keyboard/Mouse/Touch set: Similar to the aforementioned issue, when keyboard or mouse and touch are used together, it may be difficult to identify the partnership between these items.

Finally, we present two use case scenarios to demonstrate the extensibility of our framework.

- Use Case 1: *Support for novel input techniques such as soft touchpad* - Apart from mouse and finger touches as input methods, one can make use of this architecture and the capability of the touch table to create a novel touchpad-like input on the tabletop. This would supplement the current direct touch ability and allow users to reach positions that are out of arm's reach.
- Use Case 2: *Integration of the TUIO protocol as an abstraction to the different input hardware types* - TUIO can be integrated into our unified architecture between the hardware layer and the input interpretation layer. This would allow a wider variety of input devices to be used. At the moment, customizations are necessary in order for these to fit in with our architecture, in particular, additional support for hands identification and gesture types.

CONCLUSION

We have proposed an architecture that unifies different input methods (multiple mice input, multiple separate touches and natural hand gestures) with the standard desktop Operating System for use on tabletop settings. With this architecture the need for writing custom multi-touch and multi-pointer aware applications is greatly reduced.

We have presented one working implementation of this architecture in Ubuntu, using Compiz and MPX technologies, and discussed our current limitations. The limitations primarily stem from immature software, which has been acknowledged by the community and is being addressed by the developers.

Nevertheless, we have shown that our architecture can provide multi-touch and multi-pointer support for existing applications on an existing Operating System.

Given the flexibility of our architecture, existing legacy applications can be used in a multi-touch aware environment. In addition, this architecture allows new natively multi-touch aware applications to be used simultaneously with legacy software.

We hope that our work will allow the community to continue further research and development to support collocated work on the tabletop.

Some interesting areas to investigate include the selection and implementation of other Operating System level gestures, which can interact with legacy applications, for example permitting scrolling through content, or implementing a physics engine to allow application windows to be thrown. Additionally, a framework allowing application specific gestures would add further multi-touch support for popular legacy applications, perhaps by mapping the local gestures to keyboard shortcuts or executable scripts.

REFERENCES

1. O. Chapuis and N. Roussel, "Metisse is not a 3D desktop!" In Proceedings of UIST'05, the 18th ACM Symposium on User Interface Software and Technology, pages 13-22, October 2005. ACM Press.
2. Community Core Vision, http://nuicode.com/projects/tbeta
3. Compiz, http://www.compiz.org/

4. F. Echtler, and G. Klinker, “A multitouch software architecture” in *Proceedings of the 5th Nordic Conference on Human-Computer interaction: Building Bridges (NordiCHI '08)*, vol. 358, Lund, Sweden, ACM Press., October 2008, pp. 463-466.
5. Engineering Windows 7 “http://blogs.msdn.com/e7/archive/2009/03/25/touching-windows-7.aspx”
6. A. Esenther, and K. Ryall, “Fluid DTMouse: Better Mouse Support for Touch-Based Interactions,” in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ACM Press, Venezia, Italy, May 2006, pp. 112-115
7. J. Han, "Low-Cost Multi-Touch Sensing through Fustrated Total Internal Reflection," in *Proceedings of UIST '05*, Seattle, Washington, ACM Press., 2005, pp. 115-118.
8. P. Hutterer, MPX – The Multi-Pointer X server, http://wearables.unisa.edu.au/mpx/
9. Ingenieurs Ltd., flickOS, http://www.ingenieursonline.co.uk/
10. “Input Redirection, MPX and NOMAD”, February 2009. [Online]. Available: http://smspillaz.wordpress.com/2009/02/21/input-redirection-mpx-and-nomad/
11. M. Kaltenbrunner, T. Bovermann, R. Bencina and E. Costanza, “TUIO: A protocol for table-top tangible user interfaces,” in *Proceedings of 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005)*, Vannes, 2005.
12. R. Kruger, S. Carpendale, S. Scott and A. Tang, “Fluid Integration of Rotation and Translation,” in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*, Oregon, 2005, pp. 601-610.
13. Microsoft, "Microsoft Surface", http://www.microsoft.com/surface/index.html
14. Microsoft Developer Network, Desktop Window Manager, http://msdn.microsoft.com/en-us/library/aa969540(VS.85).aspx
15. A. Morrison, G. Jacucci and P. Peltonen, “CityWall: Limitations of a Multi-Touch Environment,” in *Public and Private Displays workshop (PPD 08), International Working Conference on Advanced Visual Interfaces (AVI 2008)*, Naples, 2008, pp. 31-35.
16. B. Myers, H. Stiel, and R. Gargiulo, "Collaboration Using Multiple PDAs Connected to a PC." In Proceedings CSCW'98: ACM Conference on Computer-Supported Cooperative Work, November 14-18, 1998, Seattle, WA. pp. 285-294.
17. Multitouch Oy, “Multitouch Cell”, http://multitouch.fi/
18. C. Shen, F. D. Vernier, and M. Ringel, “DiamondSpin: an extensible toolkit for around-the-table interaction,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI’04), Vienna, Austria, April, 2004, pp. 167-174.
19. Sparsh-UI, A Multitouch API for any multitouch hardware / software system, http://code.google.com/p/sparsh-ui/