uPlatform: A Customizable Multi-user Windowing System for Interactive Tabletop

Chenjun Wu, Yue Suo, Chun Yu, Yuanchun Shi, and Yongqiang Qin

Department of Computer Science and Techonology, Tsinghua University,
Beijing, P.R. China
anwingwu@gmail.com, suoyue@tsinghua.edu.cn, yc2pcg@gmail.com,
shiyc@tsinghua.edu.cn, qyq08@mails.tsinghua.edu.cn

Abstract. Interactive tabletop has shown great potential in facilitating face-to-face collaboration in recent years. Yet, in spite of much promising research, one important area that remains largely unexplored is the windowing system on tabletop, which can enable users to work with multiple independent or collaborative applications simultaneously. As a consequence, investigation of many scenarios such as conferencing and planning has been rather limited. To address this limitation, we present uPlatform, a multi-user windowing system specifically created for interactive tabletop. It is built based on three components: 1) an input manager for processing concurrent multi-modal inputs; 2) a window manager for controlling multi-user policies; 3) a hierarchical structure for organizing multi-task windows. All three components allow to be customized through a simple, flexible API. Based on uPlatform, three systems, uMeeting, uHome and uDining are implemented, which demonstrate its efficiency in building multi-user windowing systems on interactive tabletop.

Keywords: tabletop, multi-user, windowing system, window management.

1 Introduction

In recent years, tabletop system has shown great potential in facilitating face-to-face collaboration. Yet, in spite of much promising research, one important area that remains largely unexplored is the windowing system on tabletop. Familiar in desktops, windowing system enables user to work with several programs simultaneously. It is also necessary for tabletops because many scenarios such as conferencing, planning, and exhibition require parallel interaction on multiple independent or collaborative applications. This requirement is being further magnified with the emergence of large-scale interactive tabletops [1, 2] where users may have more chance to work independently.

However, there arise several challenges when replanting the windowing system from desktop to tabletop due to three fundamental factors of tabletop: multi-user usage, direct input and horizontal display. First, the co-located multi-user support of tabletops brings many new problems, such as authorization and conflict handling. These issues are inherently ignored by current windowing systems such as Microsoft

J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, pp. 507–516, 2011.

Windows and X Window, because they are designed for single user usage. Second, the direct input makes the WIMP interfaces of these most popular windowing systems unsuitable for tabletops since multi-touch interfaces lack the precision usually afforded by indirect pointing devices. Third, the horizontal display requires more research to be performed to understand space management practices on tabletop.

Little research has been performed to investigate these challenges in a systematic way. While a number of improvements for current windowing systems have been proposed by HCI researchers in order to explore space management [3], or support multi-point input [4, 5], very few carefully consider the issues raised in co-located multi-user usage, such as authorization and conflict management. On the other hand, nor has work been performed to help researchers investigate these challenges. Most of current toolkits for tabletops [2, 6, 7] only focus on full-screen, single application development. Researchers must spend excessive effort on developing the underlying plumbing before they can focus on their interested areas, such as input authorization.

To address this limitation, we present uPlatform, a multi-user windowing system specifically created for interactive tabletop. Our driving goal was that this system would be customizable enough to facilitate the design of innovative user interfaces and window management policies in the context of multi-modal input, multi-task UI and multi-user usage, and at the same time be simple enough for average researchers to quickly learn and use. Therefore, uPlatform is a tool for creating new types of tabletop environment rather than a new tabletop proposal. It is built based on three components: 1) an input manager for processing concurrent multi-modal inputs; 2) a window manager for controlling multi-user policies; 3) a hierarchical structure for organizing multi-task windows.

This paper is organized as follows. After introducing some related work, we describe uPlatform by providing an overview of its architecture as well as a description of its API and configuration tools. We then illustrate its customizability through three sample systems. Finally, we conclude with a discussion on our experiences and findings.

2 Related Work

Many works focus on extending the functionality of current windowing systems in order to support groupware applications. The MIDDesktop [4] provides a desktop-like environment to execute several Java applets simultaneously. The Swing Applets simply execute in sub-windows within the MIDDesktop display area, and can receive inputs from multiple mice. MPX [5] is a modified X Window System that can accommodate multiple input devices, and provides new APIs for developers to create collaborative applications based on these inputs. SDGToolkit [8] brings multi-mice support to Microsoft Windows operating system. It also considers the orientation issues for tabletop displays and can automatically rotate the cursor according to a participant's seating angle. However, the shortcoming of these systems is that their paradigm is too closely tied to desktop environment and mice input, which makes their WIMP interfaces unsuitable for tabletops since multi-touch interfaces lack the precision usually afforded by these indirect pointing devices.

There are also several tools targeting towards helping HCI researchers explore the novel window management policies. Ametista [9] is a mini-toolkit that supports the creation of new OpenGL-based desktop environments using both a low-fidelity

approach using placeholders, as well as a high-fidelity approach based on X app redirection. Similarly, the Mettise [3] toolkit uses an image compositing approach that makes it possible to apply a number of visual effects and geometrical transformations on windows. As one of its examples, Mettise shows how it enables tabletop interface that features automatic window orientation. Unfortunately, both Ametista and Mettise do not take input customization into consider and only support single user interaction with traditional window applications using mouse.

Various Toolkits [2, 6, 7] have been built specifically for the interactive tabletop. They all provide abstractions to support the core tabletop interaction functionality, allowing researchers to concentrate on their applications. For example, the DiamondSpin toolkit [6] implements "around the table" interaction techniques to address orientation challenges in tabletop use. The T3 toolkit [2] allows multiple tabletops to be connected together to support mixed-presence collaboration. PyMT [7] provides a growing collection of reusable widgets and interaction techniques in order to accelerate the development of applications on tabletops.

Unfortunately, all these toolkits assume that the tabletop system can only display a single application at one time, and the API they provide is based on the notion of a single rectangle display with no layout information. As the result, it's impossible to use these toolkits to set up scenarios such as meeting and brainstorming that need to execute multiple collaborative applications simultaneously, unless we modify the source code or create an ad-hoc layer that host and manage other applications. However, both the approaches are a hard task, one that few HCI researchers are willing to do.

3 The uPlatform System

We design uPlatform for several reasons: Most importantly, we were unable to find a tabletop environment that supports executing multiple independent or collaborative applications simultaneously on interactive tabletop. It is a crucial requirement for our current research, since we are exploring many novel scenarios such as meeting and planning on a large-scale tabletop of size 2m*4m. These scenarios all involve mixed-focus collaboration [10], where individuals frequently transition between individual and shared tasks within a group. Therefore, we wanted a multi-user windowing system specifically created for interactive tabletop. We also definitely wanted the system to be customizable enough so that HCI researchers can explore innovative user interfaces and window management policies in the context of multi-modal input, multi-task UI and multi-user usage. Furthermore, as a tool for creating new types of tabletop environment, uPlatform should be efficient enough, allowing for rapid prototyping and experimentation.

In this section, we provide an overview of the uPlatform architecture, followed by a discussion of the implementation, a description of the customizability of each component, and the development and configuration tools provided by uPlatform.

3.1 The Architecture

uPlatform's architecture was designed with two main themes in mind: simplicity and extensibility. Figure 1 depicts a schematic of the architecture and execution of the main system loop. uPlatform is built based on three components: 1) an input manager

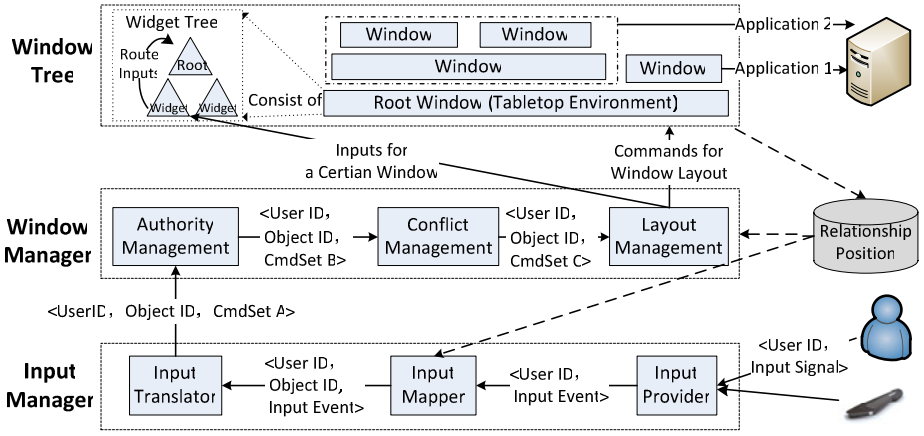

Fig. 1. The architecture of uPlatform

for processing concurrent multi-modal inputs; 2) a window manager for controlling multi-user policies; 3) a hierarchical structure for organizing multi-task windows.

Input Manager. The input processing is much more complicated on tabletops than that on desktops. Typically, multiple users provide input streams from different channels, including hands, styluses or physical objects. In addition, the same input signal can be translated in different ways. For example, different gesture libraries can be defined for different tasks. Therefore, it is important to separate higher level application logics from multimodal inputs to leave developers or researchers focusing on the design of interface components.

In uPlatform, we divide the input processing procedure into three steps: 1) input providing step to receive and normalize input events from different input devices; 2) input mapping step to determine which window an input action intends to operate on; 3) input translating step to encode raw inputs signals into primitive commands before they are sent to window manager for further processing. Each step can consist of one or several objects to do the concrete work. For example, the input providing step may contain TUIO provider and multi-mice provider at the same time.

Window Manager. The main task of Window Manager is to coordinate the inputs from different users, and change the position and relationship of windows based on the inputs and context. According to this, we assign the tasks of windows management to three sub components: authority manager, conflict manager and layout manager, which deal with command authorization, conflict resolution, and windows layout respectively. Input commands will pass through these three managers in order and each manager has the ability to stop the inputs from further traversing.

In order to obtain flexible and extensible management, the managers all work in a policy-based way. Formally, policy is an abstract representation of the properties and behaviors that define how the management is done. Each policy provides sufficient interfaces that can be used and extended by researchers. In uPlatform, we define three types of policies to deal with authority, conflict and layout issues respectively. Each

window can hold these three policy instances to define how its child or offspring windows are managed.

Window Tree. Like most windowing systems on desktop, uPlatform uses a tree structure to organize all the windows into a hierarchy. Each application based on uPlatform consists of one or several windows. The root of this tree is a full-screen application, which is called *Tabletop* and should be designated before the underlying windowing system starts to run. Other applications can then be loaded and hosted within the root window. Each window is encapsulated into an independent thread, which holds its own message loop and rendering engine. Therefore, windows can respond concurrently to multiple users' simultaneous input.

Also like many GUI toolkits, each window is made up of a widget tree, which can be seen as a WPF control, and created visually in Microsoft Visual Studio or other design tools. A number of operations are supported by uPlatform to manipulate windows. For example, rotate, reposition a window, maximize, restore a window or change the parent of a window.

3.2 Discussion of Implementation

We implement uPlatform based on WPF and C# programming language. WPF is Microsoft's next generation graphical subsystem on Microsoft Windows platform that enables modern UI features such as transparency and transforms. These features, together with powerful IDE provided by Microsoft, give great convenience for users building rich, rotatable applications. Besides, WPF provides a good framework for reusing, customizing and extending existed components, which is crucial for an extensible development tool.

We implement a WPF control called *UObject* as the root of each window. It holds the basic look and feel of the window, such as border and menu. To achieve multi-layer window organization, we define a Canvas control at the top of UObject which can hold child windows. In addition, each UObject is located within a *HostVisual* that holds its own event dispatcher and works independently with other windows. We monitor each dispatcher carefully, and catch any exception thrown out of its processing loop in order to guarantee the whole system running correctly.

Since WPF's input processing mechanism does not work in a multi-thread environment, we implement the input manger from the ground up. All input events inherit from UTable.Inputs.InputEventArgs class with fields such as device ID and user ID. After the inputs are mapped and sent to a window with the help of input mapper and layout manager, uPlatform is responsible for routing the inputs from a certain leaf widget to the root widget so that each widget along the path has the opportunity to handle the inputs and stop further routing. This is similar to WPF's input routing mechanism for handling inputs within a window.

3.3 uPlatform API

uPlatform provides a simple, flexible API for customizing the input manager, window manager and user interfaces discussed above. This API is packed into a development

toolkit called *uTableSDK* [11], which is freely available to academic researchers. In this section, we describe a few of the most important classes.

Customize Inputs. Inputs can be customized by implementing classes that inherit from *IInputProvider*, *IInputMapper* or *IInputTranslator* interface, and putting them into uPlatform's input manager. These three interfaces define how a certain type of input is provided, mapped and translated respectively. Take IInputProvider as example, the only method it contains is the InputTriggered event whose type is listed below:

```
delegate void InputEventHandler(InputEventArgs args);
```

The customized input provider can raise this event with any input argument that inherits from InputEventArgs at any time, which will be captured by input manager, and sent to the input mappers. The InputEventArgs in fact has no concrete fields, and it depends on researchers to add data such as multi-touch position or key type. Researchers can add or remove providers simply by calling the following two methods in UTableHelper class:

```
RegisterInputProvider(Type inputType, IInputProvider p)
UnregisterInputProvider(Type inputTy, IInputProvider p)
```

Customize Window Management Policies. *LayoutPolicy*, *ConflictResolvePolicy* and *AuthenticationPolicy* are base classes for researchers to modify the management policies of window manager. They each contain several template methods that will be called by uPlatform when commands are passed through the corresponding managers. For example, LayoutPolicy provides:

```
void OnInputReceived(IObject obj, InputEventArgs args)
void OnObjectInserted(IObject obj)
```

These two methods will be called to handle layout inputs on a certain child window or define how to put an incoming child window respectively. Therefore, the researchers have the freedom to define how the windows should be rearranged according to user input.

Unlike interfaces in input manager, the policy classes can be set in each window. It means different windows can hold different management policies, which gives even more flexibility. Management policies can be changed dynamically by simply calling the following methods in the UObject class:

```
LayoutPolicyType = typeof(DemoLayoutPolicy);
LayoutPolicyParameter = null;
```

Customize. User Interfaces. uPlatform allows researchers to rapidly create tabletop content in a way similar to implementing a WPF window: edit the widget tree in a WYSIWYG manner in Visual Studio. In addition, we provide a rich set of multi-touch controls and Visual Studio templates so that the development can be further accelerated by simple drag-drop under Visual Studio.

3.4 Development and Configuration Tools

uPlatform integrates well into Visual Studio, and provides an efficient and comfortable development environment to researchers. For example, researchers can simply open a project template in Visual Studio in order to create a new uPlatform-powered system.

uTableSDK provides an xml file for each system to configure environment settings (Fig 2). Because most of our development occurs on desktops, the system by default is configured to use multiple mice as its input provider, and use a blank window with physical layout policy as the root tabletop. These settings can be changed easily when we want to test the application in different environments. For example, change the input provider to TUIO when this application is deployed on an interactive tabletop.

```
<UTable>
    <!--Define the root of window tree and its management policies-->
    <Tabletop>
     <Class Type="UTable.Objects.UHome" Assembly="UTable.Objects"/>
     <LayoutPolicy Type="UTable.Policies.PhysicalPolicy" Assembly="UTable.Objects">
       <Property Name="LinearDamping" Value="0.6"/>
     </LayoutPolicy>
     <ConflictPolicy Type="UTable.Policies.RankPolicy" Assembly="UTable.Objects"/>
     <AuthorityPolicy Type="UTable.Policies.PublicPolicy" Assembly="UTable.Objects"/>
    </Tabletop>
    <!--Define the inputs-->
    <Input><Providers><Provider>
     <Class Type="UTable.Input.MultiMouseProvider" Assembly="UTable.Input"/>
  </Provider><Providers></Input>
</UTable>
```

Fig. 2. Customize windowing system through a config file

4 Examples

In this section, we will discuss three windowing systems constructed based on uPlatform. They demonstrate the variety of tabletop user interfaces and management policies that uPlatform can facilitate. All systems are developed within one day, which demonstrate the framework's efficiency in building multi-user multi-task windowing systems on horizontal displays.

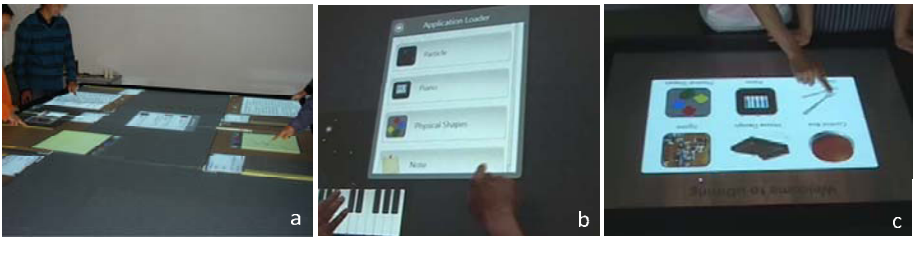

Fig. 3. The windowing systems based on uPlatform. (a) uMeeting, (b) uHome, (c) uDining

4.1 uMeeting

The first windowing system is called uMeeting (Fig.3a). It is developed for meeting scenario, where multiple users work individually or collaboratively on a large-scale interactive meeting table. This system shows the advantage of the flexible UI management using the layout policies, authority and conflict policies of uPlatform.

We create a UObject to represent the workspace that is located directly on the tabletop. The workspaces divide the tabletop into multiple sub-regions, and work as a working unit for each user. User-defined applications, such as document, photo and note can be put on the workspace, and transferred between tabletop and different workspaces. uMeeting implements a physical policy that is applied to the tabletop and workspaces with different parameters so that windows can be dragged and thrown with friction and inertia. Personal authority policy is applied to workspace to ensure that only the owner has the authority to manipulate the content of his personal space. The tabletop, however, is assigned public authority policy so that every user has the identical authority of the objects directly put on tabletop.

4.2 uHome

The scenarios for interactive tabletop at home, such as tea table, are quite different from those in a meeting room. Firstly, the display scale is smaller, and participants can see and interact with any part of the table; secondly, the activities on the tabletop usually involve collaboration of multiple users based on one or several applications; finally, the activities may change frequently, especially when there are some guests. Noticing these, we create a system, called uHome, to help manage applications on an interactive table at home (Fig.3b). It uses a simple two-layer structure to organize the user interface: the bottom is tabletop and the upper layer contains applications running on the system. A special application, called AppLauncher can be launched from the pie menu of the tabletop, which can be used to start other applications in this system. uHome replants uMeeting's public authority policy and the first-come, first-use conflict resolution policy to the tabletop of uHome so that everyone have the equal opportunity to manipulate the applications.

4.3 uDining

uDining is a windowing system custom made for an interactive dining table. When designing uDining, we focus on some specific characteristics a dining table has. For example, the participants sit at the four sides of the table, and involve in a single task, such as ordering food. Based on these observations, we apply a simplest layout policy to uDining: a single application is full-screen displayed on the table. All inputs are sent to the active application except for the special buttons at the center of the table, which can trigger a global menu. One main command in this menu is switching the active application to AppBrowser, a native application of uDining, as shown in Fig.3c. This application gives users a way to browser and selects all applications installed on the system.

Since users in uDining only interact with applications, not the management system itself. We do not assign any authority and conflict control policy to uDining. In fact, we believe in these scenarios, it's the application's task to care about these issues.

5 Discussion

The three systems described in the last section illustrate the basic capabilities and extensibility of uPlatform. Through our experience developing windowing systems for different interactive tabletops based on uPlatform, we noticed many issues related to policies design and replantation among different scenarios and devices. The following are a few of them that belong to part of our current investigation.

Although the screen scale and scenarios vary greatly for different tabletops, their windowing systems still share a lot of features on inputs, management policies and user interfaces. uPlatform takes advantage of this and greatly reduce the cost when implementing different windowing systems. In fact, the native input utilities we provide in uPlatform can satisfy most of the input requirements. Furthermore, since we can reuse the management policies and widgets in previous systems, the difficulties and time decrease significantly when our work go deeper.

The extensibility also plays an important role in implementing different windowing systems. We have an impressive experience when we want to add a pen as an input device to the system. The typical way is to add a multi-touch provider to the system so that application can respond similarly to other touch input without changing anything. However, we can also create a provider with a new input channel so that applications can respond differently.

The policies also give a good way for extending previous work. In practice, there are two ways to do this. One is reusing a previous policy with different arguments. For example, the tabletop and workspace can both have physical property while the fiction on them can be different. The other way is inheriting from a policy and rewrite or extend its logic. This way can be more powerful with a little more effort.

6 Conclusion

In this paper, we have described uPlatform, a customizable multi-user windowing system specifically created for interactive tabletop. We have presented the general architecture of the system and described its API and configuration tools. We have presented several examples of uses that demonstrate its efficiency in building multi-user windowing systems on interactive tabletop.

The SDK of uPlatform is available from http://utablesdk.codeplex.com.

Acknowledgment. This paper was supported by Development Plan of China under Grant No. 2009AA01Z336 and National Natural Science Foundation of China under Grant No. 61003005.

References

1. Jens, T., Marc, H., Benjamin, W., Lasse, S., Sebastian, F., Markus, K., Rainer, M.: Advancing Large Interactive Surfaces for Use in the Real World. In: Advances in Human-Computer Interaction, vol. 2010, pp. 1–11 (2010)

2. 1hilip, T., Peter, R.: T3: Rapid Proto-typing of High-Resolution and Mixed-Presence Tabletop Applications. In: Proc. TABLETOP 2007, pp. 11–18 (2007)
3. Chapuis, O., Roussel, N.: Metisse is not a 3D desktop! In: Proc. UIST 2005, pp. 13–22. ACM Press, New York (2005)
4. Shoemaker, G.B.D., Inkpen, K.M.: MIDDesktop: An Application Framework for Single Display Groupware Investigations. School of Computing Science, Simon Fraser University, Report No. TR 20001-01 (2001)
5. Hutterer, P., Thomas, B.H.: Enabling Co-located Ad-hoc Collaboration on Shared Displays. In: Proc. AUIC 2008, vol. 76, pp. 43–50. Australian Computer Society Inc. (2008)
6. Shen, C.: DiamondSpin: An Extensible Toolkit for Around-the-Table Interaction. In: Proc. CHI 2004, pp. 167–174. ACM Press, New York (2004)
7. Thomas, E.H., Juan, P.H.: PyMT: A Post-WIMP Multi-Touch User Interface Toolkit. In: Proc. TABLETOP 2009, pp. 17–24. ACM Press, New York (2009)
8. Tse, E., Greenberg, S.: Rapidly Prototyping Single Display Groupware through the SDGToolkit. In: Proc. AUIC 2004, vol. 28, pp. 101–110. Australian Computer Society Inc. (2004)
9. Nicolas, R.: Ametista: a mini-toolkit for exploring new window management techniques. In: Proc. CLIHC 2003, pp. 117–124. ACM Press, New York (2003)
10. Tang, A., Tory, M., Po, B., Neumann, P., Carpendale, S.T.: Collaborative coupling over tabletop displays. In: Proc. CHI 2006, pp. 1181–1190. ACM Press, New York (2006)
11. uTableSDK, `http://utablesdk.codeplex.com`