An Environment for Fast Development of Tabletop Applications

Ombretta Gaggi
Department of Mathematics
University of Padua, Italy
gaggi@math.unipd.it

Marco Regazzo
Anyt1me S.r.L.
via Siemens 19, Bolzano, Italy
marco.regazzo@anyt1me.com

ITS'13, October 6–9, 2013, St. Andrews, United Kingdom.
ACM 978-1-4503-2271-3/13/10.
http://dx.doi.org/10.1145/2512349.2514917

Abstract
In this paper we present *Xplane*, a software layer for fast development of applications running in separate, independent windows in multi-touch interactive tabletops. Our framework supports gestures recognition and communication between different windows or applications. Moreover, it is based on web technologies to abstract from hardware and software configuration.

Author Keywords
Tabletop applications; touch interfaces; webkit engine.

ACM Classification Keywords
H.5.2. [User Interfaces]: User interface management systems (UIMS)

Introduction
The number of applications developed for multi-touch tabletops is dramatically increased in the last years. Interactive tabletop surfaces are used to improve learning activities, inside museums, where the diversity of visitors create a natural laboratory for testing this kind of interface, to help the management of emergency, and in many other collaborative activities like, e. g., photoware, etc.

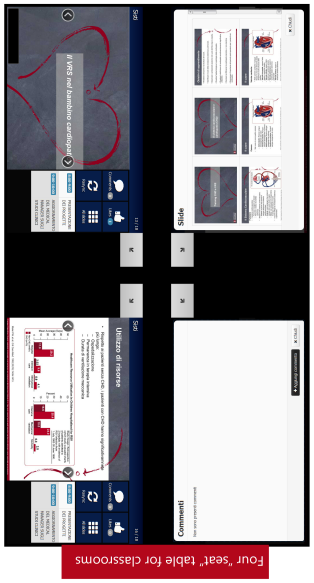

Figure 1: Screenshot from an application developed,using our framework, for doctors training. Four medicians cooperate to find out the correct diagnosis using four different windows, which are affected by the interaction of all the partecipants.

Tabletops provide an important set of new features in terms of user interface: the users are placed around the table, they can collaborate using the same space and objects or can compete for them. Despite this difference, many applications are still developed and designed as single or even full-screen applications, thus using a tabletop as a big tablet. But tabletops size allows the contemporary presence of different applications in stand alone windows which can communicate each other (see Figure 1).

In this paper we present *Xplane*, a software layer which is able to abstract from hardware and software constraints of the device in which it is installed (screen size, operating system, etc), allows the developer to create applications running in separate windows and handles communication between them. Moreover, our solution provides a Javascript API which allows a developer to build an entire tabletop application only using web technologies, in particular HTML5, CSS3 and Javascript, without the need to know a particular language bound to the sw/hw configuration.

Related Works

There are a lot of applications for interactive tabletops and surfaces available on the market. They have many common features: they are highly visual systems, mainly controlled by touches and some common gestures performed on the surface of the system, e. g., browsing a collection of items, selecting a particular item, accessing a documents, and so on. All these applications can interact with several users at the same time, and each user requires a different orientation of the interface, according to his/her position. Despite many common requirements, the developers of all these applications need to implement the majority of these features from scratch and the available frameworks provide only very low level features.

A solution for this problem is represented by frameworks, like *Microsoft Surface 2.0 SDK and Runtime* [4], *Windows Presentation Foundation + Native Touch recognition by Microsoft Windows 8* [5] and *Smart Table SDK* [6], which help to develop multitouch applications but require a particular hardware/software configuration.

GestureWorks [2] and *Arena* [8] are frameworks which provide generic and cross platform functionalities, like gestures recognition, to develop touch applications, but they are not able to manage more than one application being launched at the same time or multiple application enclosed in different windows.

Glassomium [7] is a project which aims to go beyond these limitations using web technologies. Glassomium provides multiple windows management, (limited) gestures recognition usig TUIO , allows for rotations, scaling and dragging of windows/apps but the project is currently at beta, developed by a single person (0 contributors and only 2 forks on github), that do not seem to be enough robust and frequently updated to be used for commercial solution. Moreover, applications developed with Glassomium are limited inside a browser window, and cross-windows communication is not supported. *Xplane* overcomes these limitations.

Description of the framework

Xplane is a framework, which allows to speed up the development of applications for multi-touch surface providing a set of features common to multi-touch applications like windows disposition, gestures

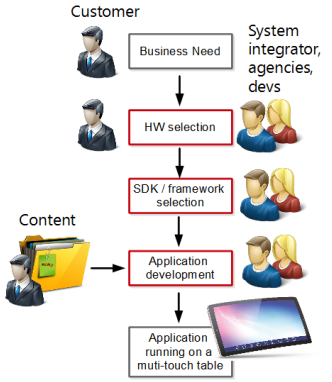

Figure 2: Traditional process to design and developed a multi-touch application.

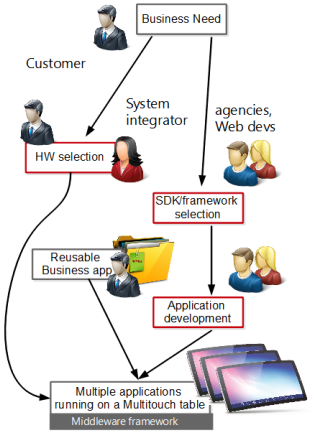

Figure 3: Process to design and developed a multi-touch application using *Xplane*.

recognition and interface orientation and windows management.

Xplane changes the process of designing a tabletop applications. As discussed in Section "Related Works", many software frameworks are bound to a particular hardware solution (see Figure 2). Our solution works with any operating system, therefore, the choice of hardware and software configuration are completely independent and can be moved further in the process since the developed applications can run on different surfaces (see Figure 3). Moreover, *Xplane* allows to encapsulate web applications into widgets suitable for multi-touch surfaces, therefore already developed web applications can be easily adapted and re-used to multi-touch interactive surfaces. Moreover, we do not ask the developer to know any particular language or technology bound to the particular hw/sw equipment, he or she only needs to know how to use the Javascript API provided by our framework.

Xplane uses *WebViews* to separate contents from the interaction widgets. A *WebView* is a component, in particular a *view*, that displays web pages. Using a WebView it is possible to embedded HTML pages inside an application. This component uses the *WebKit* rendering engine to display web pages and encapsulates all the functionality needed to interact with the user and with other components within the table (e. g., gesture management and recognition, window orientation, button to close the window, the stack of visible objects, etc.) and widgets for the visualization of media items like videos and images. Therefore, the developer only need to specify which is the web page to render.

Contents can be arranged and personalized using the CSS standard language. But the provided interaction is very poor since the user can touch the interface, but the touch is interpreted like the movement of a mouse pointer. To manage portability of touches and gesture recognition, we use the TUIO protocol [3] which allows the transmission of an abstract description of interactive surfaces, including touch events and tangible object states. This protocol encodes control data from a tracker application and sends it to any client application that is capable of decoding the protocol.

The recognition of the gestures is managed extending qTUIO [1], a library which implements a TUIO listener on a local UDP socket and recognizes gestures made with one finger. Since the user usually move windows and objects with the whole hand, we extended this library to recognize and manage also gestures which involve more one fingers or both the hands.

Communication between different windows

Concurrent interactions by more than one users is another important issue to be considered. As an example, the users can compete for space on the surface. For these reason, when a new window is opened (even by a new user or not), this operation can require the resize of all the other windows already present on the table. Otherwise, actions from a particular user may affect the behavior of the windows of other users. To allow the easily implementation of applications with this kind of features, *Xplane* implements a communication protocol between the different *WebViews* which managed the windows.

Let us consider as example, an application with a map, e. g., a map of a city with the list of its museums,

rendered with HTML5 on a WebView. When the user touches a museum the application opens a new window, with the web site of the museum, and the user can interact with this window, resize it, or move across the table. If the user touches the “go to the map” button on the new windows, the initial window with the map is moved over the current window of the user.

To implement this behavior, we developed a communication protocol between the *WebViews* implementing the windows, which allows the developer, using our Javascript API, to change the content or the behavior of a window on the base of the behavior/user interaction on another windows. The software layer *Xplane*, implemented using the C language to address performance issues, acts as a windows manager and carries the messages. The Javascript API allows to enlarge, resize, minimize, close, rotate or move a window, in response to a user interaction, also on other windows.

Discussion and conclusion

In this paper we present *Xplane*, a software layer for fast development of applications running in separate windows, handling communication between them. The framework is based on modern web technologies to adress portability.

The novelty of our approach consists on different issues, first of all the possibility to use (and possible re-use) web pages to decrease the time spent to develop the multi-touch applications and to learn a particular language bound to the chosen hardware. Using *Xplane* a content editor can create simple applications for tabletop. Moreover, our framework create applications that can run in all tabletops, i. e., they are independent from the hw/sw configuration of the tabletop. Finally, no other software framework provides the possibility to run more than one application into independent windows and supports communication between them.

Future works will be dedicated to the implementation of an API to manage Near Field Communication (NFC).The idea is to save the state of the user, in term of opened documents and windows, and which is the window currently active, and to re-create the entire workspace at the correct state, every time that user approaches the system.

References

[1] Belleh, W., and Blankenburgs, M. qTUIO Library. http://qtuio.sirbabyface.net/.

[2] Ideum. Gestureworks Core. http://gestureworks.com/pages/core-home.

[3] Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. TUIO Framework. http://www.tuio.org/.

[4] Microsoft. Surface 2.0 SDK. http://msdn.microsoft.com/en-us/library/ff727815.aspx.

[5] Microsoft. Walkthrough: Creating Your First Touch Application. http://msdn.microsoft.com/en-us/library/ee649090.aspx.

[6] SMART Technologies. SMART Table SDK. http://downloads01.smarttech.com/media/products/sdk/smart-table-sdk-summary.pdf.

[7] Toffanin, P. Glassomium Project. http://www.glassomium.org/.

[8] Unedged. Arena Multitouch Platform. http://arena.unedged.com/.