A Framework Interweaving Tangible Objects, Surfaces and Spaces

Andy Wu, Jayraj Jog, Sam Mendenhall, and Ali Mazalek

Synaesthetic Media Lab
GVU Center, Georgia Institute of Technology
TSRB, 85 5th St. NW
Atlanta, GA 30318, USA
{ andywu, jayraj.jog, smendenhall3, mazalek }@gatech.edu

Abstract. In this paper, we will introduce the ROSS framework, an integrated application development toolkit that extends across different tangible platforms such as multi-user interactive tabletop displays, full-body interaction spaces, RFID-tagged objects and smartphones with multiple sensors. We will discuss how the structure of the ROSS framework is designed to accommodate a broad range of tangible platform configurations and illustrate its use on several prototype applications for digital media content interaction within education and entertainment contexts.

Keywords: tangible interaction, API, interactive surfaces

1 Introduction

In the past decade, researchers have begun to address the needs of developing interaction techniques that can more seamlessly bridge the physical and digital worlds. The vision of Tangible Media [1] presents interfaces that integrate physical objects with interactive surfaces and responsive spaces that embody digital information. Yet working with diverse sensing and display technologies is difficult and few tools support this type of development. Our research lab focuses on tangible interaction and sensing technologies that support creative expression bridging the physical and digital worlds. In the last few years, we have developed several projects based on a draft framework, the Responsive Objects, Surfaces and Spaces (ROSS), a unified programming framework that allows developers to easily build applications across different tangible platforms.

Tangible interaction research is a relatively young field, and as such application designers and developers working in this space face a number of challenges. The sheer number and variety of sensing techniques, interface devices, form factors and physical settings envisioned by researchers present a complex space from the perspective of generalizability of development tools. To further complicate the situation, many of the sensing and display technologies used by tangible interfaces are still in the process of evolution and maturation, and have not yet reached the state of stability and robustness.

2 Related Work

User interface toolkits enable programmers to rapidly create applications for graphical user interfaces (GUIs) that make use of traditional input/output devices. Tangible user interfaces combine non-standard hardware and software. A tangible toolkit allows developers to develop applications rapidly without having detailed knowledge of the underlying protocols and standards.

2.1 Multiouch Toolkits

Multitouch applications have become quite popular lately. There are many software toolkits that support multitouch gesture interaction and object manipulation. A lot of them share the same protocols, such as TUIO or Open Sound Control (OSC). These toolkits mostly use a client-server architecture. In other words, these toolkits acts as servers that dispatch multitouch messages to client applications.

libTISCH [2] (Library for Tangible Interactive Surfaces for Collaboration between Humans), is a development framework for tabletop computers. It provides a number of features such as gesture recognition, GUI widgets and hardware drivers for various tangible interaction input devices and technologies (FTIR, DI, Nintendo Wii, Microsoft Kinect). libTISCH employs a layered architecture that consists of four layers: hardware abstraction, transformation, interpretation and widget. The hardware abstraction layer takes raw data from the input devices regarding the positions of objects, finger touches etc. and produces data packets. The transformation layer converts the raw data into corresponding screen coordinates, which then cause the interpretation layer to generate events as appropriate. The widget layer listens for events and reacts accordingly.

ReacTIVision [3] is a computer-vision based system designed for interactive tabletop displays. It tracks not only finger touches but also objects tagged with computer vision patterns, called fiducial markers. ReacTIVision defines its own tangible interaction protocol, the TUIO protocol to communicate with client applications. The TUIO is an open framework based on Open Sound Control – an emerging standard for interactive environments. The TUIO protocol and its API have become a standard for the interactive surface community. Several of the early ROSS applications adopt the TUIO protocol. Another similar project, TWING is a software framework that has been use to develop the Xenakis table [4], a multi-user interactive tabletop display to create music through objects and finger gestures.

2.2 Physical device toolkits

Physical device toolkits often provide predesigned hardware components and software APIs for rapid prototyping. Phidgets [5] are electronic components analogous to software widgets, allowing construction of complex physical systems out of simpler components. Phidgets provide the physical components and the software API that lower the barrier for novices to control electronic devices through computers. Similar to Phidgets, the Microsoft Gadgeteer [6] is a prototyping platform

for rapid constructing and programming. However, it also integrates 3D CAD support for developers to design the physical container for the electronics. Introducing a CAD library to the toolkits helps developers build embedded system products even more efficiently. Toolkits for developing Augmented Reality (AR) or Virtual Reality (VR) applications have the capability of detecting objects in the 3D space. They are not toolkits for physical devices but still worth mentioning here. The ARToolkit [7] is a computer vision based system that tracks fiducial markers attached to objects. Another toolkit, DART [8] is designed for rapid development of AR or VR applications through GUI.

2.3 Integrated toolkits

Prior work on tangible application development has shown that both these approaches are important in evolving the area of TUI design, and we thus feel they should not remain entirely separate from one another. In order to further support these developments, the toolkits for shared tangible display surfaces need to support communication with other kinds of devices as well.

The iROS platform [9] supports an interactive workspace that incorporates technologies on many different scales, from personal devices to large table and wall displays. The iROS utilizes a dedicated server to dispatch customizable messages between different devices. The Papier-Mâché toolkit [10] provides an event-based model for application development using RFID-tagged, barcoded, or computer vision identified objects. The empirical study of Papier-Mâché also discovered several necessary features that influenced our tangible toolkit design, which are ease of use, facilitating reuse and modular code structure.

Our previous work on the Synlab API [11] has integrated different types of tangible platforms into a shared framework. The Synlab API is based on the API of the TViews [12] media table, a multi-user interactive platform that supports real-time tracking of tagged objects. We have extended the Synlab API to include additional technologies and platforms, including RFID object tracking, multi-touch surfaces, and displays such as a tilting table and spinning screen.

TwinSpace is a generic framework for developing collaborative cross-reality environments. It allows the connection of heterogeneous clients (mouse + keyboard clients, interactive tabletop display, instrumented spaces, etc.) to a shared virtual environment. The TwinSpace architecture is presented in detail in [13]. As a platform for prototyping and experimentation, the TwinSpace infrastructure permits a range of techniques for integrating physical and virtual spaces. Events generated by sensors or input devices in a physical space can be published and reported in the virtual space. Furthermore, TwinSpace offers facilities to define spatial and structural correspondence between physical and virtual spaces (for example, the meeting space can be mapped to a corresponding region in the virtual space). Spaces may also be connected through functional ‘contact points’ whether the real and virtual spaces correspond spatially or not (for example, a tabletop interface might present an interactive top-down view over the virtual space).

3 ROSS Infrastructure

Working with emerging sensing and display technologies is a challenging task for application developers. In order to integrate heterogeneous tangible interfaces we provide the ROSS programming framework that is based on a nested abstraction of responsive objects, surfaces and spaces, for developing tangible user interfaces. This serves to unify the application development process across a broad range of the TUI space, and can accommodate the continued evolution of the underlying technologies and devices.

3.1 Relationship between Objects, Surfaces and Spaces

The relationships between the three cornerstones of ROSS, objects, surfaces and spaces, can be understood simply by talking about their respective roles. A ROSS object is any device that is capable of running a ROSS application. Thus, peripheral devices such as cameras, microphones, mice etc. are not ROSS objects, while computers, mobile phones, tablets, tables and microcontrollers are ROSS objects. A ROSS surface usually refers to any 2-dimensional space with which it is possible to interact. This could mean touchscreens, touch-sensitive walls, tabletop computers or any other surface through which it is possible to provide input to a ROSS application. ROSS objects can be placed on ROSS surfaces.

A ROSS space is any 3-dimensional space in which one or more ROSS objects or surfaces are located and within which it is possible to interact with a ROSS application. ROSS spaces abstract the various problems of locating a ROSS object spatially, in addition to being a means of input through gestures, sounds, light patterns etc.

3.2 ROSS Design Goals

Tangible tools need to share common functionality across different applications, provide standards that can support development across different kinds of platforms and devices, and most important of all is the ease of use. ROSS has the following design goals:

Designed for applications
The design of ROSS API is through a practice-based approach. In other words, we have built and continue to build applications based on the concept of ROSS and improve the API from the developers' feedback. This means that ROSS API should be appropriate for a wide variety of contexts, applications and platforms.

Platform independence
Although ROSS API is currently available in Java only, we aim to ensure that it will eventually be available in other popular programming languages. ROSS API

introduced ROSS XML, a file that can be easily processed by most programming languages.

Rapid development and scalable structure
ROSS is envisioned to work in a rapidly changing and evolving environment. It is also designed for critical network conditions. ROSS passes only the most essential messages for real-time communication. Using UDP for the transport layer achieves all of these goals. Besides, there is little connection maintenance overhead compared to TCP, and the lack of flow control and acknowledgment are ideal for real-time data transfer. In addition, ROSS can scale well without maintaining persistent connections.

Work under resource constraints
Many of the applications for ROSS involve one or more embedded systems, e.g. mobile phones or single chip microcontrollers. Since these devices have memory and/or energy constraints, ROSS is designed with these constraints.

3.3 ROSS Structure

The core of the ROSS framework rests on the notion that the different objects, surfaces and spaces in a responsive environment can exist in nested relationships with respect to one another. The concept of nested objects, surfaces and spaces is most easily illustrated using an example, as follows.

Consider an RPG game that runs on an interactive table. Users interact with the game through tagged physical pawns and finger touches on the sensing surface. We can first think of the interactive table itself as a responsive object that provides a nested sensing surface. Each tagged pawn is another kind of interactive object that is in a nested relationship to the sensing surface. Now imagine we replaced the pawns with android phones. These objects now each provide their own nested sensing surface, on which finger touches can again be seen as nested objects. The phone might also provide interaction through touches, buttons or other sensors, which can be considered as nested controls that operate on the interactive object.

ROSS devices have an XML file that provides more information about those devices. A ROSS device can be just a ROSS object or also have a ROSS surface or be in a ROSS space.

Table 1. A sample ROSS XML structure. The server-side XML indicates that the server table needs to listen to events from *Android* phones. The client-side XML indicates that the client phone needs to listen to events from a *TuioTable*.

Server-side XML (table)	Client-side XML (Android phone)
`<Device type="TuioTable" id="0">` `</Device>`	`<Device type="TuioTable" id="0">` `</Device>`

<Subscription type="Android"> </Subscription>	<Subscription type="Android"> </Subscription>

As you can see, the ROSS XML file describes the structure of the ROSS device to which it belongs. The XML encapsulates the nested structure of the device perfectly. This particular device (a cellphone) is a ROSS object that also has a ROSS surface. Other devices (such as cameras) have the ability to understand ROSS space and their relationship to it.

3.5 Cross network communication

The ROSS API is designed as a distributed networked system. Each node of a ROSS network acts as a client, receiving information about other I/O devices, and also as a server, distributing information about the I/O devices that it contains.

Each node contains XML file that holds a description of its I/O devices. By trading this information with other nodes, each node knows the device capabilities of the other nodes in the network. This file also contains information on what devices the particular node wants to listen to. When a node parses a remote XML file, it looks through this information and subscribes the remote node to the I/O events of any local matching devices. The I/O events or state of a particular device are transmitted to subscribers via a UDP protocol. A ROSS node both listens to, and broadcasts its IP address from a multicast group. Upon receiving the multicast packet, another ROSS node will establish a TCP/IP connection with the packet sender. The nodes then exchange their ROSS XML files.

Due to the modular driver/XML parsing architecture, to construct a new ROSS application one only needs to gather all the necessary device representation objects and drivers, write an appropriate XML device structure and have the application listen for new devices and register for their I/O events. No networking code is needed for sending and receiving I/O events. The ROSS Manager handles it.

4 Applications

We have shown that it is possible to specify a broad range of different platforms using the ROSS structure. We describe some selected example applications below.

4.1 Moons Over You

Moons Over You [14] is one of the earliest projects that experimented with the ROSS infrastructure. In the project room, a moon appears projected on the wall for each person entering the room, and each person's moon then follows them around as long as they remain in the space. At the same time, these moons can be seen on the surface of the interactive tabletop display. Users at the table can visualize people's movements by looking at the changing image of a moon-filled sky on the table's

surface. They can also reach into this tabletop sky, and playfully re-arrange the moons with their finger touches. The moons respond to these finger touches as well, and people thus get the sense that their moon is playing tricks on them as it runs away. The left figure of Fig. 1 illustrates how the server captures and interprets the movement of users and sends the ROSS messages to a client dedicated for visualizing the actions through the wane and wax of moons. The right photo of Fig. 1 shows several moons based on the users' locations and movement.

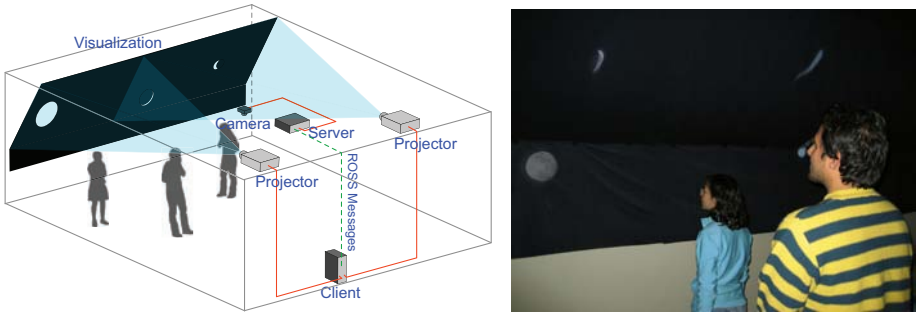

Fig. 1. The system diagram of Moons Over You and the users' interaction.

4.2 ROSS Ripple

ROSS Ripple is an initial concept and technology demo built with the ROSS API. It is an exploration of embedded and synchronized interactive surfaces. Ripple connects an interactive tabletop and Android phones with fiducial markers attached to their backs. When the user touches any of the surfaces, ripples of color emanate from the touch position. When placed on the table, a phone appears to become part of the table surface. The backgrounds of the table and phone align. Ripples created on the table surface run across phones. Ripples created on the phone surfaces run onto the table. If a phone is lifted off of the table, then touches on the phone have a one-to-one "synchronized" spatial relationship with ripples on the table surface.

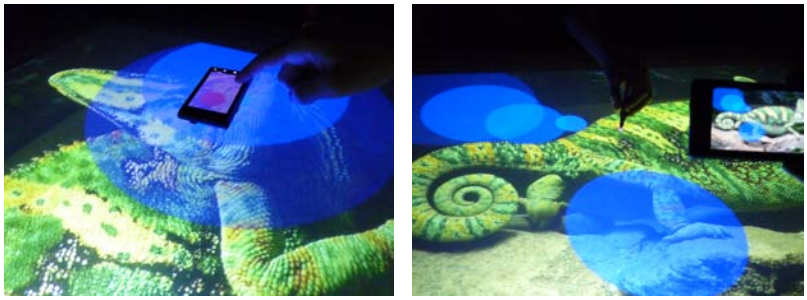

Fig. 2. The interactions of ROSS Ripple

In the left photo of Fig. 2, a mobile phone is placed on an interactive tabletop display. The blue ripples on the tabletop are generated by touches on the tabletop

while the purple ripple on the cellphone screen is generated by a touch on the cellphone screen. These two types of ripples are interweaved because of the nested structure of ROSS. The screen of mobile phone acts like a small window on the tabletop that reveals the tabletop image blocked by the phone. In the right photo of Fig. 2, the mobile phone has left the tabletop surface. The cellphone screen and the tabletop display show exactly the same image. The act of placing one surface onto another embeds that surface into the surface beneath.

4.3 Ross Paint

ROSS Paint is a sketching application that utilizes multiple phones as touch input devices to paint onto a monitor or screen. Lines are drawn between touch inputs from each phone to produce a unique line drawing effect. ROSS Paint explores how to integrate multiple input devices into collaborative artwork. Users do not merely interact separately on the same surface. The nature of the tool forces them to synchronize their activities to produce a desired effect. Fig. 3 shows the pattern when a user draws random curves on two phones.

Fig. 3. ROSS paint with two cellphones.

4.4 Kinect applications

We propose a class of applications that employ the Kinect sensor manufactured by Microsoft. The Kinect allows reconstructing the 3D scene of a space. It consists of IR and visible light cameras and microphones, and can be combined with sophisticated image recognition software to detect fiducial markers, objects, and even perform facial recognition. A ROSS space can be defined in terms of the Kinect's location in it. It can act as the "origin" of the coordinate system and the other devices can locate themselves with respect to each other in this system. Already networked Kinects are used in pairs to build 3D models of a room. They may be used to perform perfect object recognition and thus obviate the need for fiducial markers.

Kinect-based ROSS application – Impromptu tangible objects

Kinect is used to recognize shapes of objects. This, combined with the in-built microphone can be used to "teach" a table to recognize objects as tangibles. By further associating them with pre-existing objects whose properties are stored in a database, it is possible to use an arbitrary object as a proxy for any other object, for the sake of convenience (as shown in Fig. 4). For example, if a user wants to discuss the possible positioning of various types of billboards on a network of highways with some associates, and all she brought are pens, pencils and notepads. She could conceivably train the table to recognize each pen or pencil as a different type of billboard. When a pen is placed on the table, the table could display an image of the corresponding billboard next to the pen.

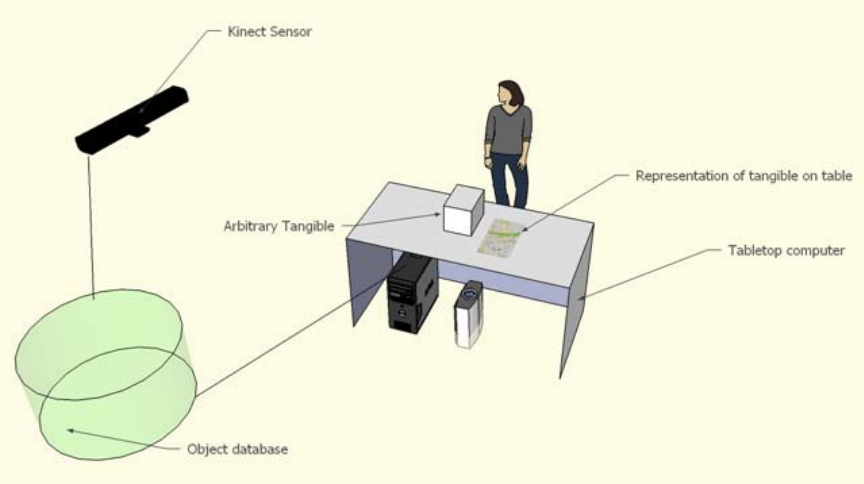

Fig. 4. The concept diagram of ROSS Kinect.

5 Summary

This paper has demonstrated the capability of the ROSS API through several applications. In order to test out and develop a breadth of applications and interaction concepts on emerging tangible platforms, creators need easy development tools to work with. The ROSS API has integrated heterogeneous resources, from multiple objects and surfaces to spaces. In the future, we plan to continue extending the ROSS API to support a broader range of platforms and technologies, and to support application development in different programming languages. The current version of the ROSS API is an engineering prototype. Working with emerging sensing and display technologies is a challenging task for application developers. In order to support continued growth of tangible interfaces, we need to improve the ROSS API from an engineers' prototype to an easy-to-use programming tool.

Acknowledgments. We thank all the application developers who used and provided feedback on ROSS in its early stages.

References

1. Ishii, H. and Ullmer, B.: Tangible bits: towards seamless interfaces between people, bits and atoms. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 234--241. Steven Pemberton (Ed.). ACM, New York, NY, USA (1997)
2. F. Echtler. libTISCH: Library for Tangible Interactive Surfaces for Collaboration between Humans. http://tisch.sourceforge.net/
3. Kaltenbrunner, M., Bencina, R.: reacTIVision: a computer-vision framework for table-based tangible interaction. In Proceedings of the 1st international conference on Tangible and embedded interaction, pp. 69--74. ACM, New York, NY, USA (2007)
4. Bischof, M., Conradi, B., Lachenmaier, P., Linde, K., Meier, M., Pötzl, P., André, E.: Xenakis: combining tangible interaction with probability-based musical composition. In: Proceedings of the 2nd international conference on Tangible and embedded interaction, pp. 121--124. ACM, New York, NY, USA (2008)
5. Greenberg, S. Fitchett, C.: Phidgets: easy development of physical interfaces through physical widgets. In: Proceedings of the 14th annual ACM symposium on User interface software and technology, pp. 209--218. ACM, New York, NY, USA (2001)
6. Villar, N., Scott, J., Hodges, S.: Prototyping with Microsoft .NET Gadgeteer, In: Proceedings of TEI '11, ACM, pp 377--380. ACM, New York, NY, USA (2011)
7. Kato, H., Billinghurst, M., Poupyrev, I., Imamoto, K., Tachibana, K.: Virtual object manipulation on a table-top AR environment. In: Augmented Reality, International Symposium on, pp. 111--119, IEEE and ACM International Symposium on Augmented Reality. IEEE Press, New York (2000)
8. MacIntyre, B., Gandy, M., Dow, S., Bolter, J.D.: DART: a toolkit for rapid design exploration of augmented reality experiences. In: Proceedings of the 17th annual ACM symposium on User interface software and technology, pp. 197--206. ACM, New York, NY, USA (2004)
9. Ponnekanti, S. R., Johanson, B., Kiciman, E., Fox, A.: Portability, Extensibility and Robustness in iROS. Proc. First IEEE international Conference on Pervasive Computing and Communications, pp. 11--19. IEEE Press, New York (2003)
10. Klemmer, S.R., Li, J., Lin, J., Landay, J.A.: Papier-Mâché: toolkit support for tangible input. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 399--406. ACM, New York, NY, USA (2004)
11. Mazalek, A.: Tangible Toolkits: Integrating Application Development across Diverse Multi-User and Tangible Interaction Platforms. In: Let's Get Physical Workshop, 2nd International Conference on Design Computing and Cognition, Eindhoven University of Technology, Netherlands, July (2006)
12. Mazalek, A., Reynolds, M., Davenport, G.: TViews: An Extensible Architecture for Multiuser Digital Media Tables. In IEEE Computer Graphics and Applications, 26(5), pp. 47--55. Sep/Oct 2006. IEEE Press, New York (2006)
13. Reilly, D.F., Rouzati, H., Wu, A., Hwang, J.Y., Brudvik, J., Edwards, W.K.: TwinSpace: an infrastructure for cross-reality team spaces. In Proceedings of the 23nd annual ACM symposium on User interface software and technology, pp. 119--128. ACM, New York, NY, USA (2010)
14. Lee, H.J., Wu, C.S.(A.), Shen, Y.T., Mazalek, A.: Moons over you: the poetic space of virtual and real. In ACM SIGGRAPH 2008 posters, Article 24, 1 pages. ACM, New York, NY, USA (2008)