Responsive Objects, Surfaces and Spaces (ROSS): Framework for Simplifying Cross-Device Communication

Aneesh P. Tarun
Synaesthetic Media Lab
Ryerson University
Toronto, Ontario, Canada
aneesh@ryerson.ca

Ahmed Sabbir Arif
Synaesthetic Media Lab
Ryerson University
Toronto, Ontario, Canada
asarif@ryerson.ca

Andrea Bellucci
Department of Computer Science
Universidad Carlos III de Madrid
Leganés (Madrid), Spain
abellucc@inf.uc3m.es

Ali Mazalek
Synaesthetic Media Lab
Ryerson University
Toronto, Ontario, Canada
mazalek@ryerson.ca

TEI'15, Jan 15–19, 2015, Stanford University, Stanford, California, USA.
Workshop on Interactive Infrastructures – Towards a Language for Distributed Interfaces

Abstract
Responsive Objects, Surfaces, and Spaces (ROSS) is a framework that enables developers to easily build applications for heterogeneous network of tangible devices and platforms. This position paper outlines the current state of the ROSS framework, lessons we have learned, challenges ahead, and how the ROSS framework facilitates the vision of interactive infrastructures.

Author Keywords
Toolkits; API; Cross-Device Communication; Tangible User Interfaces.

ACM Classification Keywords
H.4.3. Communications Applications.

Introduction
Tangible User Interfaces [3] have embodied Mark Weiser's vision of Ubiquitous Computing [6] and successfully argued that the physical instantiation of digital media is necessary and advantageous for the seamless interaction with the digital world. Within the last decade, interest and research into TUIs has grown

exponentially, leading to the development of hundreds of prototypes exploring interfaces and interactions for TUIs. TUIs are also gaining mainstream attention with the availability of commercial systems such as Reactables [4] and Sifteo Cubes [5].

Developing TUI prototypes involves bringing together heterogeneous devices and platforms that include both off-the-shelf as well as custom hardware and software systems. This requires a concentrated effort in writing low-level device driver software and communication protocols for the multitude of devices encountered. The complexity involved in such an endeavor might deter researchers and developers from building a compelling and effective tangible user experience.

As a research group focusing on developing tangible media applications, we identified the need for a "tangible toolkit" that would lower the barrier for setting up ubiquitous environments. Commercial availability of tangible platforms further increased our necessity of a unified tangible framework for application developers. We developed the *Responsive Objects, Surfaces and Spaces* Application Programming Interface (ROSS API) [7][8] for developing applications for heterogeneous tangible platforms and devices.

ROSS API

ROSS API is designed to utilize combinations of different interaction platforms in a single unified application. The simplest way to enable this is by abstracting low-level connection and communication commands. However, this approach does not work well as the number of devices and variations in their capabilities increases. This approach also stifles developers from thinking about their complex application designs involving different platforms and devices. ROSS API tackles these challenges by introducing a new way of thinking about the inherent structures within application spaces.

With ROSS API, all interactive surfaces, spaces, and objects are organized in a hierarchical nested structure that parallel the real world structuring of surfaces, spaces, and tangible artifacts. This nested structure enables querying and navigating from one entity to another within an interactive system.

ROSS API's core class structure includes the following classes: responsive spaces (RSpace) representing interactive 3D spaces; responsive surfaces (RSurface) representing interactive 2D surfaces; responsive active or passive objects (RObject) tracked within an RSpace or RSurface; nested standard controls (RControl), part of an RObject. Each instance of these classes has a unique identifier and tracks unique properties.

This high-level abstraction serves as an interface between the applications and the device drivers and communication protocols. This approach enables developers to think beyond tedious low-level control and communication programming.

Consider an example tangible application with an interactive tabletop with a mobile phone and a puck. Users can interact with the tabletop and the objects on top of it. Sensors in the room track the location of tabletop. Figure 1 shows how the ROSS framework can represent this setup as a nested structure.

An XML file represents the nested structure of an application developed by ROSS API. Application

developers write XML files by hand. This information forms the basis for routing and device information. XML files also provide a quick and easy way to alter the system architecture and input methods.

The nodes in a ROSS application automatically detect one another via multicast packets and communicate their input device configurations. The Open Sound Control (OSC) protocol forms the basis for reporting events between different ROSS layers in real-time.

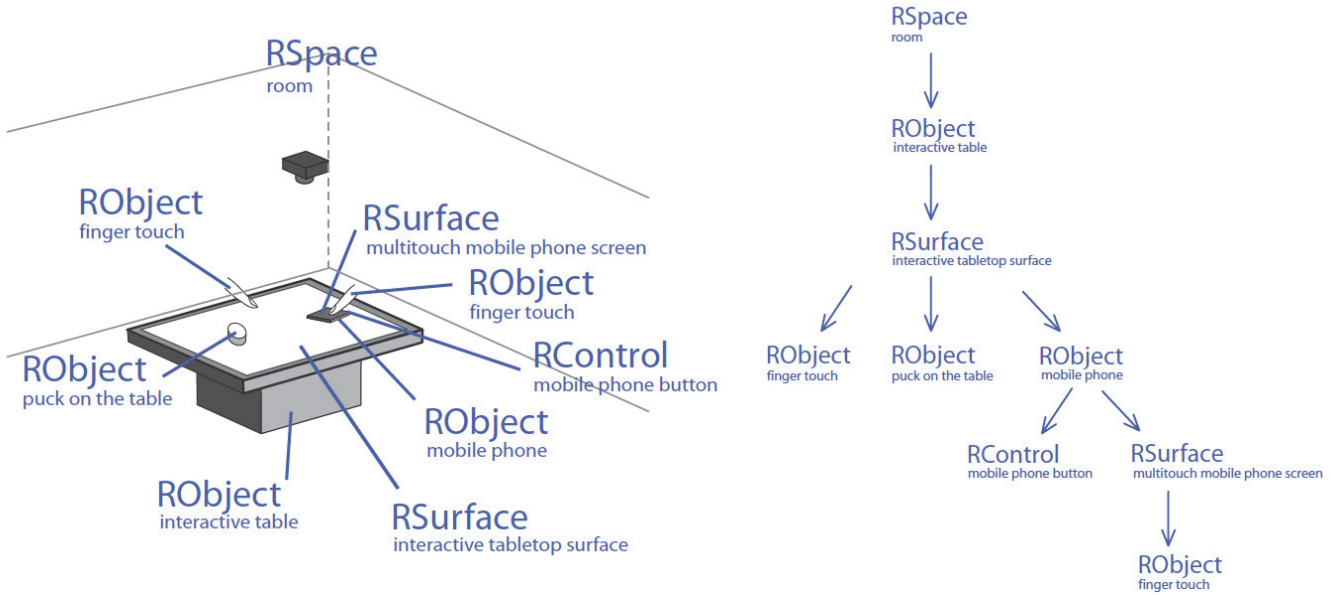

Figure 1. An example representation of ROSS API's nested structure. Left: ROSS components in a tangible tabletop application within an interactive room. Right: Representation of the room as a nested API structure.

Challenges & Future Directions

Newer technical opportunities have emerged since our initial ROSS design specifications. We have identified several areas of improvement based on these opportunities. We believe that the outlined future directions will also encourage for pushing the API to be relevant beyond TUIs.

(1) Inclusivity: The changing landscape of interconnected devices and protocols includes newer hardware platforms such as smart watches, low-cost computers, and microcontrollers having limited processing power and memory. We identify two possible ways to support such devices in a rich interactive environment. Adopting Javascript as the primary interface may be one possible way to support

myriad of devices and platforms cropping up. In addition, we consider providing centralized virtual storage and/or processing power to a distributed set of low-footprint devices as an exciting and new way to think about enabling and empowering ubiquitous devices.

(2) Ease of Design: We identify the need to simplify the use of ROSS API by developers of different expertise levels. The current version of the API favors developers who think in an Object Oriented Programming (OOP) style. There is definitely scope for supporting different programming paradigms within the current framework. Another barrier for entry, for novice developers, is the tedious and time-consuming process of preparing a large XML file (for any tangible application). A possible solution is to provide a graphical interface for rapidly configuring device ecosystems and automatically generating ROSS XML files.

(3) Ad-hoc Grouping and Ownership: ROSS API supports grouping of devices using XML configuration files. However, this approach does not allow for ad-hoc grouping of spatially and temporally collocated devices. We see an opportunity in supporting ad-hoc groupings in collaborative and unplanned scenarios.

(4) Disjoint Infrastructures: Our discussion of ad-hoc grouping raises the question of disjoint sets of interaction ecosystems and how they can communicate with one another. We believe that there is opportunity to explore issues of ownership, control, and security within the larger framework of nested objects, surfaces, and spaces.

(5) Adaptive User Interfaces: Currently the API focuses on abstracting the hardware and communications aspect of heterogeneous devices. However, research threads in multi-device interfaces [1][2] show opportunities for contextually adapting UIs for a network of devices. Current frameworks, including ROSS API, do not support easy deployment of adaptive UIs for multiple devices. We think that there is scope for supporting the deployment and management of UIs across devices within the ROSS framework.

(6) Larger Scope: ROSS API was motivated by the need to easily develop applications for various tangible frameworks. However, there is sufficient overlap between TUIs, AR interfaces, and Wearables to position the framework within the larger scope of Ubiquitous Computing. A conscious effort in this direction will ensure a more inclusive framework that can evolve with the changing landscape of devices, platforms, and interaction paradigms.

Workshop Goals

With ROSS API we have taken a first step towards enabling application development for heterogeneous tangible devices. We hope to engage in a discussion of the relevance of ROSS API in the wider context of Ubiquitous Computing. We believe that the ROSS framework can be leveraged to support the evolving landscape of interactive infrastructures.

Acknowledgements

This work has been supported in part by Qualcomm as well as the SSHRC CRC Program.

References

[1] Girouard, A., Tarun, A., & Vertegaal, R. DisplayStacks: interaction techniques for stacks of flexible thin-film displays. In CHI '12. ACM (2012), 2431-2440.

[2] Hamilton, P., and Wigdor, D., J. Conductor: enabling and understanding cross-device interaction. In CHI '14. ACM (2014), 2773-2782.

[3] Ishii, H. and Ullmer, B. Tangible bits: towards seamless interfaces between people, bits and atoms. In PCHI '97. ACM (1997), 234-241.

[4] Reactable. http://www.reactable.com

[5] Sifteo. http://www.sifteo.com

[6] Weiser, M. The computer for the 21st century. *Scientific American 265*, 3 (1991), 94-104.

[7] Wu, A., Jog, J., Mendenhall, S., and Mazalek, A. A framework interweaving tangible objects, surfaces and spaces. In HCII '11. Springer (2011), 148-157.

[8] Wu, A., Mendenhall, S., Jog, J., Hoag, L. S., and Mazalek, A. A nested API structure to simplify cross-device communication. In TEI '12. ACM (2012), 225-232.