ROSS Toolkit: An Infrastructure and API for Building Interactive Environments

Andrea Bellucci
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911, Leganés, Madrid, Spain
abellucc@inf.uc3m.es

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

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

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

Abstract

Programming responsive and interactive environments is a complex task that requires extensive knowledge of hardware and software components as well as many hours of programming. To cope with these issues we developed the ROSS (Responsive Objects, Surfaces, and Spaces) Toolkit that provides an infrastructure and API to easily build heterogeneous networks of tangible devices and interactive surfaces. In this position paper we outline the design rationale, the current state of the ROSS Toolkit and the contribution for the ITS community.

Keywords

Toolkits; API; interactive environments, prototyping.

ACM Classification Keywords

H.5.2. Information Interfaces: User Interfaces – input devices and strategies, prototyping.

Introduction

Developing responsive and interactive environments involves bringing together heterogeneous devices and interactive surfaces in the physical space. This requires a concentrated effort in writing low-level software and communication protocols for the multitude of devices encountered [1]. The complexity involved in such an

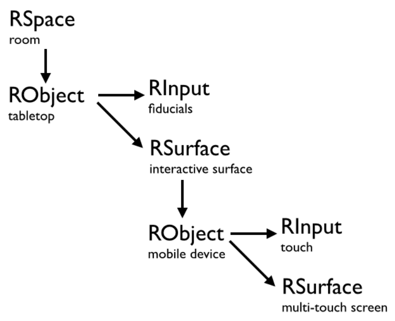


Figure 1. An example of the nested hierarchical structure: a mobile device is tracked on an interactive tabletop.

endeavor might deter researchers and designers from developing a compelling and engaging user experience. It is therefore critical for the ITS community to design simpler tools and abstractions for building applications that run across a variety of platforms and devices, as a part of their everyday research [9].

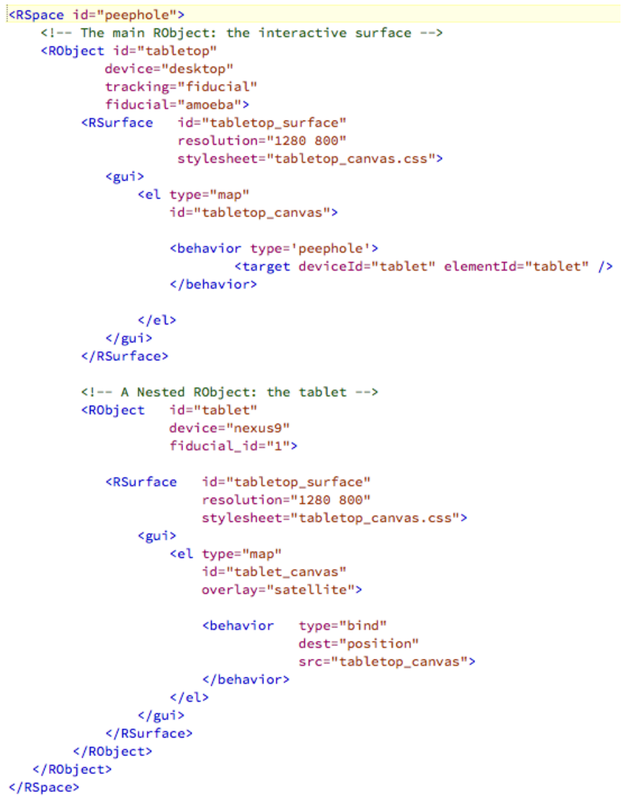
To this end, we developed the ROSS (Responsive Objects, Surfaces, and Spaces) Toolkit as a complete re-design of the original ROSS API (Application Programming Interface) [9] specifications to assist the development of responsive and interactive environments that integrates off-the-shelf mobile devices, Arduino-based custom tangibles, interactive surfaces, and different software platforms.

The toolkit exposes an API as well as tools for abstracting low-level communications between heterogeneous interactive surfaces and reducing the programming effort for sensor-based and spatial interactions between devices. The design of the ROSS Toolkit has been informed by (a) the experience gained from the previous design, (b) the needs emerging from the recent technical opportunities, and (c) the lessons learned from the development of interactive environments and cross-device interactions in two different research laboratories (Synlab at Ryerson University and DEI Lab at Universidad Carlos III de Madrid).

Design goals

Our design goals for the development of the ROSS Toolkit are the following.

(G1) Inclusivity: allow for easy inclusion of diverse hardware platforms as well as newer hardware platforms such as smartwatches and low-cost microcontrollers within a single interactive environment.



```
<RSpace id="peephole">
    <!-- The main RObject: the interactive surface -->
    <RObject id="tabletop"
            device="desktop"
            tracking="fiducial"
            fiducial="amoeba">
         <RSurface   id="tabletop_surface"
                     resolution="1280 800"
                     stylesheet="tabletop_canvas.css">
            <gui>
                <el type="map"
                    id="tabletop_canvas">

                    <behavior type='peephole'>
                            <target deviceId="tablet" elementId="tablet" />
                    </behavior>

                </el>
            </gui>
         </RSurface>

         <!-- A Nested RObject: the tablet -->
         <RObject   id="tablet"
                    device="nexus9"
                    fiducial_id="1">

            <RSurface   id="tabletop_surface"
                        resolution="1280 800"
                        stylesheet="tabletop_canvas.css">
                <gui>
                    <el type="map"
                        id="tablet_canvas"
                        overlay="satellite">

                        <behavior   type="bind"
                                    dest="position"
                                    src="tabletop_canvas">
                        </behavior>
                    </el>
                </gui>
            </RSurface>
        </RObject>
   </RObject>
</RSpace>
```

Figure 2. An example of XML descriptor file for a mobile device tracked on an interactive surface.

(G2) Low Threshold/High Ceiling [6]: provide high-level building blocks to implement complex use cases

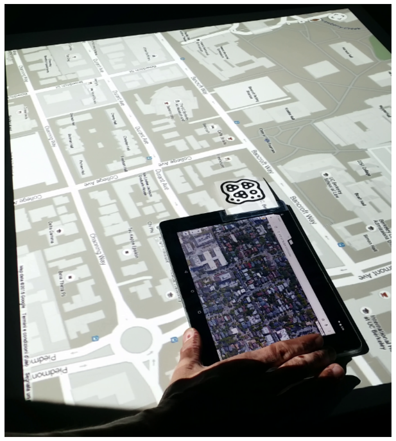


Figure 3. A sample application: the mobile device is tracked on an interactive surface exploiting the reacTIVision fiducial markers [8]. The mobile device displays an additional layer of information for the digital map rendered on the tabletop. For tracking the position of the tablet, we exploit the capability of the tabletop device to recognize reacTIVision fiducial markers and use the TUIO protocol [8] to transport all the relevant information, such as the position, speed, and rotation.

for novice developers (low threshold) while allowing low-level customizability for expert developers (high ceiling).

(G3) Technical Abstraction [1]: provide a broad range of functions and features out-of-the-box, e.g. high-level APIs for managing data from motion or depth sensors.

(G4) Extensibility: support extending the API both at the micro-level (adding new functionalities within the context of a single project) as well as at macro-level (extending the API for the entire ROSS community).

The ROSS Toolkit

The ROSS Toolkit provides a conceptual framework that allows designing interactive environments as hierarchical nested structures. Every device, screen, and sensor in an interaction space is mapped within a hierarchical structure, outlined in an XML descriptor file. This hierarchical tree (Figure 1) encapsulates relationships between various entities and determines how they interact. The XML descriptor (Figure 2) forms the basis for generating the application code and managing the communication between different devices, tangibles, and interactive surfaces.

Together with XML-based authoring, ROSS Toolkit provides: (1) an API packaged as JavaScript components within the Node.js environment [7]. This is exposed to the developers through the XML descriptor and JavaScript functions, (2) a parser to generate the source code (from the XML descriptor) to run the interactive environment, (3) a server running on Node.js (a cross-platform runtime environment) that hosts the client applications, registers the connected clients, and lastly performs the role of a connection and communication bridge between different devices, and finally (4) client applications that are served onto a device's browser environment and are mostly made up of JavaScript and CSS files. A client's UI is first rendered as a Jade file, an intermediate HTML template language for applications layout [3]. In the case of custom tangibles, the parser generates deployable and Arduino-compatible code.

The API allows implementing interactive environments using either an XML descriptor or JavaScript code. Authoring applications using the XML descriptor supports different levels of abstraction for simplifying the development process. Consider the hierarchical tree in Figure 1 for a sample setup of an interactive space (Figure 3). *RSpace* is an abstract container that encapsulates the whole space. Hierarchically, this is the root node of an application tree. As a part of the API, this provides spatial context (i.e. coordinate system, size, and scale of the interactive space) and function-calls for manipulating the coordinate systems for devices within the interaction space. *RObject* is a container class for sensors, tangible devices, and other components that constitute an interactive object. The *RSurface* class represents all display surfaces. It supports querying and manipulation of display output parameters (e.g. screen resolution, orientation) and touch input parameters (e.g. number of simultaneous touch points, granularity of touch points). The *RInput* and the *ROutput* classes represent sensors and actuators respectively. They allow developers to control the I/O channels: registering for sensor events, modifying their parameters, and driving the actuators. Registering for sensor events can be done using event listeners (traditional approach) or templates to directly link a sensor input with an actuator output or a function parameter.

Positioning

The contribution of our work for the audience of this workshop is twofold.

(1) The experimentation with the ROSS Toolkit will highlight what kind of meaningful interactions are favored/precluded by its design rationale and features. This way, we will be able to investigate the different facets of the infrastructure problem [1] and reflect on design drivers and implications to build shared infrastructures for tangible tabletops and interactive surface, e.g. level of technology abstraction, end-user development, inclusivity, the "low threshold/high ceiling" trade-off, breadth of API coverage and extensibility.

(2) ROSS Toolkit uses and builds upon TUIO to simplify cross-device communication. However, the TUIO protocol presents limitations when it comes to manage data from heterogeneous sensing sources. With our solution, which allows developers to extend TUIO in order to encode and transport custom event data, we aim to start a discussion on what features are needed for novel communication abstraction mechanisms to support the interaction between multiple tangible devices and surfaces in a space.

Acknowledgements

This work has been supported by the Canada Research Chairs program and the 2015 UC3M postdoctoral mobility scholarship.

References

[1] Keith Edwards, Mark W. Newman, and Erika Shehan Poole. 2010. The infrastructure problem in HCI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10). ACM, New York, NY, USA, 423-432.

[2] Saul Greenberg. 2007. Toolkits and interface creativity. *Multimedia Tools and Applications* 32, 2: 139–159.

[3] Jade Template Engine. 2015. Retrieved September 20, 2014 from http://jade-lang.com

[4] Martin Kaltenbrunner and Ross Bencina. 2007. reacTIVision: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction* (TEI '07). ACM, New York, NY, USA, 69-74.

[5] Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST '11). ACM, New York, NY, USA, 315-326.

[6] Brad Myers, Scott E. Hudson, and Randy Pausch. 2005. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact*. 7, 1: 3–28.

[7] Node.js JavaScript Runtime. 2015. Retrieved September 20, 2014 from https://nodejs.org

[8] TUIO Protocol. 2015. Retrieved September 20, 2014 from http://www.tuio.org

[9] Andy Wu, Sam Mendenhall, Jayraj Jog, Loring Scotty Hoag, and Ali Mazalek. 2012. A nested API structure to simplify cross-device communication. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction* (TEI '12), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 225-232.