Toward Toolkit Support for Integration of Distributed Heterogeneous Resources for Tangible Interaction

Abstract
Building tangibles-based applications for distributed computing contexts compounds the complexity of realizing tangible interfaces with the complexity of distributed computing. We present preliminary work toward a toolkit that represents steps toward decoupling these concerns. This toolkit enables users to build tangible applications for a range of computational contexts that vary in the number, type and locality of tangible interaction devices with minimal changes to source code.

Keywords
Tangible user interfaces, distributed user interfaces, tangible interaction toolkits

ACM Classification Keywords
H5.2. Information interfaces and presentation: Miscellaneous.

General Terms
Design, Human Factors
Introduction
Computing applications and services we use everyday are increasingly distributed and multi-granular. Datasets are becoming more numerous, massive and spread about the world. Processing elements are scaling both up and out as with the move toward multiple core processing and the proliferation of objects with embedded processors and network connectivity. But we face significant challenges to engaging these resources through user interaction. For emerging interactive paradigms such as tangible and embedded interfaces, the challenges of tangible interface development are compounded with the complexity of integrating distributed, heterogeneous resources.

Large-scale scientific visualization is an application domain for which distributed, heterogeneous computing is becoming more commonplace. Our collaborators build applications and services for visual analysis of huge volumes of scientific data (several gigabytes to petabytes), hosted on servers across the globe. We have co-developed several tangible interaction-based applications for distributed computing contexts, one of which we overview in this paper. From these experiences we identify several distributed tangible interaction concerns:

1. The concern of integrating diverse, physical interaction devices;
2. Concerns of mapping user input to desired behavior onto objects within the domain functional core;
3. The concern of coordinating interactive elements over several types of communication channels

We present a toolkit prototype that separates these concerns. This toolkit supports development of tangible-based applications that target a range of computing contexts within minimal code changes. This toolkit is characterized by:

- Flexibility via interaction resource proxies
- Abstract interfaces that encapsulate interactive behavior
- Network-transparent interaction device integration

Related Work
This paper builds upon related work primarily in the areas of tangible interface toolkits. Several tangible interface toolkits handle device management details and provide users with abstractions that present physical interaction devices as physical widgets or physical event generators [4, 7, 9, 12]. Several toolkits support integration of tangible input based upon computer-vision [8, 9, 11]; other toolkits support the integration of mechatronic-based tangibles [2, 4, 5, 12]. Our toolkit is agnostic to the underlying sensing technology and can support both vision and mechatronic tangibles via APIs for a class of tangible interactors [9, 13]. Virtual objects that exhibit behavior of a given modality can be bound to a given concrete interactor regardless of physical implementation.

Tangible interaction toolkits also differ in the range of computational contexts they can support. Several toolkits were designed with native support for localized interaction [4, 9]. Some toolkits employ communication mechanisms typical to ubiquitous computing systems [2, 10] in which distributed components are integrated without a priori knowledge. These interaction-programming approaches simplify the integration of many components. Other toolkits are based upon a client-server communication model, yielding higher
performance and simple designs for simple configurations [8]. Another approach is to hide network-programming details, resulting in a programming approach that more closely resembles familiar development environments [10]. Our approach seeks to support the construction of distributed tangible applications that yield scalable performance but not in apparent system complexity.

A Tangible Interaction Toolkit Prototype
Motivated by earlier experiences in developing tangibles for distributed, heterogeneous computing we are building a toolkit, tentatively called TUIKit. We are motivated by the possibility of an interactive system architecture that allows a developer to disregard whether some component is local or remote so long as it is appropriate, available and provides acceptable performance. We also envision an architecture that gives interaction designers the option of several components that provide a given function despite diverse implementations and forms.

Fig. 1 illustrates the core components of the current TUIKit architecture. This work-in-progress toolkit is based upon several core concepts:

1. Loose coupling through composition of proxies;
2. Generic APIs for overlapping capability providers;
3. Adaptors to access heterogeneous resources;
4. And communication transparency.

Loose coupling via composition of proxy objects
In our model, developers compose interactive applications by adding proxies to an interactive context. These proxies provide a gateway for accessing one or more resources. These resources currently include interaction devices (physical and virtual) and in the future other computational capabilities (e.g. data services, application domain objects and services). These proxies can be bound dynamically at runtime. For instance a developer may add to her interface composition a proxy for a rotary input that may be physically connected to her host computer and at another time wish integrate a dial connected to some other host via network.

Generic APIs for Interactive Resources
Our toolkit provides APIs that abstract implementation specific details for a given physical interaction modality. An interactive proxy provides a concrete resource by implementing that provider's interface. This approach builds on the approach of several tangible programming toolkits to provide abstract APIs for multiple tangible interaction device implementations [4, 5, 7, 10].

Adaptors for Heterogeneous Resources
While interaction resource proxies provide a generic programming interface, adaptors translate between the abstract interface for a class of interactive resource providers and a particular interaction resource implementation. It is through these adaptors we intend to expose features unique to a particular resource provider.

Communication Channel Transparency
Proxies are bound to the concrete resources they represent over an interaction message bus that provides a generic interface for several communication models (e.g. client-server, publish-subscribe, multicast, peer-to-peer) over several communication channels (network sockets, Unix pipes for inter-process...
communication, shared memory for intra-process communication). Using this approach resources for interaction are integrated in a network-transparent manner. Currently, TUIKit only uses client-server and publish-subscribe communication models, but we will explore other models in the future.

**Use Cases**

Here we describe two uses of TUIKit for development of distributed tangibles-based applications.

**Tangible for Large Scale Interactive Visualization**

Our collaborators are developing an environment for real-time, interactive large-scale data visualization applications. Many of the use-cases supported by this environment involve the use of remote distributed resources for data, and graphics rendering. Work sessions supported by these systems may involve multiple users at multiple locales engaging several visualization services instances. In earlier efforts to support remote interactive visualization, they found tangible interaction to have favorable properties, and so we were asked to design tangibles to drive user interaction. The tangibles we provided were based on the parameter interaction tray tangible, in which abstract parameters of visualization can be bound to individual physical controls [15, 16]. TUIKit was used to integrate remote user interaction, (Fig. 2a). An early demonstration of this was a winning entry in an international competition on innovative uses of large-scale computing resources in which a collaborator uses using tangibles from Shanghai, China to control giga-scale data visualization on resources in the southeastern United States [6].

**Surface Oil Spill Data Visualization Tangibles**

The April 2010 BP Oil Spill has sparked much research and public outreach activity. One effort in which we have been involved concerned the development of a Microsoft Surface application for visualizing geospatial data related to the BP Oil Spill (Fig 2b). TUIKit was used to explore the integration of tangible interaction techniques within the tabletop environment. Users could engage the application via a Surface-mediated tangible that supports physically constrained touch-based interaction as well as the manipulation of tagged objects to simulate physical controls. This same artifact can be mediated by an Apple iPad to provide similar functionality. We call these objects, casier tangibles, which are discussed in depth by Ullmer et al [15].

**Toolkit Implementation Status**

TUIKit consists of device drivers and plug-ins, a device manager, an interaction message bus and class library with bindings for python, Java and C#. TUIKit currently supports several mechanical-electronic (mechatronic) interaction devices including dials based upon several
types of sensors and an RFID reader, which were all developed using the Blades and Tiles modular interaction hardware toolkit [14]. This toolkit also supports commercially available physical interaction devices including the Griffin Powermate media control knob, and RFID modules by ID Innovations and Olimex. For the Surface Oil Spill application, we wrote an adapter to transform OSC messages generated by the touchOSC iPhone/iPad application. We are currently preparing TUIKit for release as an open source library.

def zoomOnDialEvent(self, event):
    delta = event.value - self.last_val
    self.zoom(delta)

def main():
    interCtrl = interactorController()
    dial1 = interCtrl.getInteractor('rotary1')
    vizCtrl = vizController()
    dial1.addListener(TUIKit.ROTATEDIAL,
    vizCtrl.zoomOnDialEvent)
    interCtrl.add(dial1)
    interCtrlr.start()

Figure 2 Code Snippet: Using TUIKit to control visualization

Discussion
The code snippet in Fig. 3 shows how one might integrate a rotary input to control the zoom camera function within a visualization application. Early versions of the TUIKit class library were used in two semesters of an introductory interface design and technology course. This library was used for integrating physical controls and tangibles with existing graphical user interfaces, and so there was a desire to integrate into programming environments familiar to the students. At the time of this writing, TUIKit has been used to integrate tangible input into applications built with Java SWT, Java Swing, Processing/Java, OpenGL/C++, and C#/Windows Presentation Frameworks (WPF).

TUIKit builds upon the innovations of many prior and parallel efforts for building applications that employ post-WIMP interaction techniques. Ultimately, we wish to realize system architectures capable of scaling in the number of users, devices and locales. In the future we will perform performance tests to evaluate how scalable TUIKit is. Our goal is to achieve responsiveness of at least 10 events per second per device (~100ms per device-event) and low jitter for interaction over high latency network connections. We think this is possible because the interaction message bus is based on communication infrastructure designed to handle 1000s of message per second [1]. The challenge will be providing developers with abstractions that allow them to more easily design and manage such large systems.

Acknowledgements
This work has been supported in part by NSF MRI-0521559, IIS-0856065, EPSCoR RII-0704191, and La. BoR LEQSF (2008-09)-TOO-LIGO Outreach. Thanks also to Alex Reeser, Landon Rogge, Andrei Hutanu and Jinghua Ge.
References


