Ephemeral Interaction Using Everyday Objects

James A. Walsh, Stewart von Itzstein and Bruce H. Thomas
School of Computer and Information Science
University of South Australia
Mawson Lakes Boulevard, Mawson Lakes, South Australia, 5095

james.walsh@setoreaustralia.com, stewart.vonitzstein@unisa.edu.au,
bruce.thomas@unisa.edu.au

Abstract

The ability for Tangible User Interfaces to enable the intuitive control of existing systems and adapt to individual users' usage scenarios remains an area of development. Previous research in customizable tangible interfaces has focused primarily on the offline creation by the original system developer, instead of offering extensibility to the end user. This paper presents our system to support the ad-hoc creation of 'disposable' UIs using both projected controls and physical objects. To support these controls, a software based patch panel enables data to be mapped to external systems, and from external systems back to the system itself. Using a projector, depth camera and 6DOF tracking system, users can create and map tangible/touch-based ad-hoc user controls to existing system functionality. This allows users to both quickly create new inputs for existing functionality, as well as create new arbitrary input devices from completely passive components.

Keywords: user interfaces, ephemeral, tangible, projected, extensible customizable, reconfigurable.

1 Introduction

Following the concept of Ephemeral User Interfaces (EUI) (Doring et al., 2013) as a temporary means of communication, we extend this concept to allow the user to construct disposable (ad-hoc) UIs to control existing systems and applications using physical objects (tangibles) and projected content in the environment. These temporary UI's are designed to support the creation of UIs to control a subset of a system's existing functionality for short term usage (minutes to hours). For example, when cooking in the kitchen, users often need to quickly create a timer based on the current recipe's task. This could easily be done by allowing the user to rotate a kitchen utensil on the counter to set the time. A projector displays the time, with adjustments made by further rotations. This leverages the affordances of objects available in a natural, tangible interaction.

The motivation for these interfaces comes from the need to enable a system with a known set of functions to adapt to the context and capabilities of the user at time of use, something the original designer cannot envision. For the kitchen timer scenario, we know that a timer will be required, but not the parameters or the context, given the user may have limited space or be limited to one handed interaction. Similarly, someone sitting on the couch can quickly draw a line armrest to control the volume/channel whilst resting their arm on the arm rest. As such, there is a need for ad-hoc controls that enable users to rapidly create controls based on the current context, ideally leveraging the affordances of the immediate environment. Previous touch-based systems (Akaoka et al., 2010, Henderson and Feiner, 2008, Xiao et al., 2013), have shown a need for ad-hoc interaction, but were designed for developers, excluded tangible interactions and did not support the integration with existing systems.

Whilst previous work (Akaoka et al., 2010, Avrahami and Hudson, 2002) has looked at creating interactive prototypes from passive materials, all interaction has been touch based, ignoring the geometric and spatial relationships that Tangible User Interfaces (TUIs) (Ishii and Ullmer, 1997) offer. Our work focuses on the end user, enabling them to utilize the proxemic relationships (position, orientation, visibility, etc.) of objects available in the immediate environment, as well as touch interaction as a means of input. We believe TUIs to be a key component in ad-hoc interaction that is yet to be explored.

In this paper, we explore the application of non-traditional tangible interaction to ad-hoc EUIs. We are interested in the use of everyday objects as props for supporting TUIs and the use of projectors to augment the user's workspace. Through the use of a software patch panel, such as Ballagas et al. (2004), we can isolate the UI from the application being controlled. This allows information to flow from the user defined UI, through the patch panel to the end application. This means the system to create the UIs and the systems/functions to be controlled can be developed independently. This paper makes the following contributions:

- we present extensions to an existing TUI architecture to support ephemeral touch and tangible UIs, including support for incorporating input from external systems,
- a paradigm to support ephemeral UIs for a wide range of existing GUI input controls, and
- a mechanism to develop ephemeral input devices from passive components using touch and geometric relationships.

The first contribution addresses the core system design to support ad-hoc controls and interactivity, including integration with existing arbitrary systems. We present an implementation of a software patch panel for our architecture that is capable of passing parameters to individual applications provided by the end user. This allows the user to integrate the ad-hoc controls with any existing system. This patch panel also allows inputs to be passed into the system from external systems, allowing the incorporation of input

methods not supported by the original system, as well as support for feedback loops.

The second contribution describes a comprehensive set of example interactions to control existing applications. These are based on the existing tangible controls and the results of a preliminary study.

The final contribution outlines methods for allowing the user to create interactive, novel tangible UIs on-the-fly from passive physical components that utilize both touch and the geometric relationships both of-and-between objects. This allows the user to create disposable controls, on demand.

Following this, our work focuses on 'what' functions to perform, not 'how' to do them, extending our previous work in tangible programming by demonstration (Walsh et al., 2013). We acknowledge a number of our concepts require ubiquitous and highly portable sensing and projection technologies. In this paper, we are only focused on the architecture and methods to support interactions.

The remainder of this paper is structured as follows: related work is discussed, identifying related projects and influential factors. A summary of the system and how it is used is provided, providing a number of example applications. A preliminary study that served as the initial design phase is then described, leading into a description of the system implementation and design. We then conclude with future work and final thoughts.

2 Related Work

Our work follows previous human computer interaction work relating to TUIs, reconfigurable UIs, and their supporting architectures.

Doring et al. (2013) presented the ideas of EUIs as UIs that have at least one element designed for limited time use. They defined a design space for EUIs incorporating a) materials, b) interactions (input vs. output) and c) aspects of ephemerality. Using this design space, our work is focused on selecting the right material for the job (a), primarily as a form of input (b). Their exploration of ephemerality came from the materials used (fog, ice, jelly, etc.). We however utilize multiple objects that when together, serve as an appropriate input EUI, but when split apart fulfil their original roles (c). *Despite objects being persistent on their own, it is their utility together that is ephemeral.*

2.1 Tangible UIs

TUIs utilize the affordances of physical objects, spaces and surfaces as an interface to digital functionality (Ishii and Ullmer, 1997). Fitzmaurice et al. (1995) began exploring TUIs as Graspable UIs, using 6DOF tracked 'bricks' to manipulate digital elements This allowed users to explore the advantages of bi-manual, spatial interaction with digital functionality. Despite the nature of ad-hoc interaction meaning we are surrounded by tangible objects, previous work has failed to leverage TUIs on an equal level to ad-hoc touch interaction.

The embodiment of TUIs led to the creation of Organic User Interfaces (OUIs) (Holman and Vertegaal, 2008), exploring non-planer displays that are both input and output. This embodiment blurs the distinction between input and output and closely mirrors the feedback loop that we experience in the real world with cause and effect (Sharlin et al., 2004).

Ullmer (2002) proposed a TUI architecture equivalent to the GUI Model-View-Controller (MVC) architecture, identifying three categories of TUIs; interactive surfaces, constructive assemblies, and Tokens and Constraint (TAC). These TACs utilized the unique affordances of individual objects as logical constraints on the object. For example, an elongated groove suggests placing an object in that groove to assign a value across a range. It is these kinds of affordances that this work hopes to leverage.

2.2 Reconfigurable TUIs

Akaoka et al. (2010) explored the creation of active prototypes from passive materials as DisplayObjects. Using markers to track a passive object, predesigned virtual content (buttons, displays, etc.), designated as inputs or outputs, can be dragged from a Physical-Virtual palette onto the object. Pressing 'Play' on the palette allowed users to interact with the device. More intricate interactions between input and output controls was possible using the computer to generate scripts.

Avrahami, and Hudson (Avrahami and Hudson, 2002) used push-pin enabled RFID buttons and sliders to prototype input devices, enabling reconfiguration of physical inputs for non-planar surfaces. Building on this, the BOXES project (Hudson and Mankoff, 2006) looked at using thumbtacks attached to a circuit board to trigger user-defined macros. Upon touching a thumbtack, the software could emulate a touch at a given screen coordinate or simulate any number of predefined mouse/keyboard inputs, essentially defining a macro. Using the tacks with cardboard and tape allowed users to quickly prototype button based interaction on physical prototypes.

Both Phidgets (Greenberg and Fitchett, 2001) and VoodooIO (Villar et al., 2006) explored configurable component based UIs. Both systems offered a number of input controls and could be reused and repositioned, with Phidgets using cables to connect to a PC and VoodooIO using push-pin components to link to a conductive communication layer in a foam substrate. The processing of the input into system functionality was an offline process done by the developer. Whilst both enabled ad-hoc reconfiguration, the user was limited by components for which they do not have an input device.

In exploring touch-based UIs, Light Widgets (Fails and Olsen, 2002) explored ubiquitous touch interaction using cheap, pervasive cameras. Using a PC application to select an input type and a region on a camera's viewport for the control to be located, users could touch that location to interact. Aside from the offline creation of the UI controls, there was no feedback to the user aside from whatever function was being controlled by that input. Tangible interaction outside of touch was also not supported.

Henderson and Feiner explored Opportunistic Controls (OCs) (Henderson and Feiner, 2008) to enable natural navigation of situated Augmented Reality (AR) systems,

whilst leveraging passive feedback from the environment. Buttons, dials, etc. would utilize physical surfaces and take advantage of the affordances of those surfaces, e.g. a dial using a rotating bolt. Given the focus on AR for mechanical instruction, the OCs were predefined using knowledge of the environment the user would be in (e.g. located in front of a certain model of aircraft engine). As a future direction, Henderson and Feiner (2010) identified the capability for a user to locate an object, select a widget type and specify the mapping for that object. This work directly addresses that void.

2.3 Interaction Toolkits

Whilst frameworks exist to abstract TUIs and facilitate easier access, they are primarily for the developer. WorldKit (Xiao et al., 2013) provided developers with a software framework that uses a projector/depth camera pair to enable pervasive interaction. By abstracting the sensing and projection system to provide the developer with simple events, the developer can easily create applications that respond to real world manipulation, such as touch input and object presence. Despite enabling pervasive interaction in the environment, the system cannot be customized given the controls and their functions are defined by the original developer of the application. Our work addresses this void.

The Papier-Mâché (Klemmer et al., 2004) project enabled the fusion of different sensor inputs, allowing the developer to focus on events, rather than hardware sensors. This is along a similar line to the Proximity Toolkit (Marquardt et al., 2011) in providing a set of abstracted proxemic events both within and between objects. Kjeldsen et al. (2003) abstracted visual input, but allowed the application to ask middleware for a given input (e.g. a button), and have that control be dynamically created given current context.

Hardy and Alexander (2012) provided a toolkit for developing interactive projected displays. Focusing on developers, it abstracts the projectors and sensing hardware to provide information about touch-based interaction. This allowed the developer to focus on the application content and interactivity rather than managing display surfaces and their relation to sensed input. Our work focuses uses a similar approach to enable UI creation by end users, rather than developers.

2.4 Summary

Despite work looking at reconfigurable touch and tangible interfaces, previous attempts have stopped short of enabling completely ad-hoc interaction for arbitrary TUIs, instead focusing on touch interaction, primarily with some offline component for the mapping of them to a function to control. Following on from WorldKit and the future work identified by Henderson and Feiner in OCs, this work seeks to enable end users to define tangible and projected controls for existing functionality, whilst also integrating existing systems as a form of input.

3 Using Our EUI's

Our implementation uses a projector and depth camera (Kinect) along with an Optitrack 6DOF tracking system, used to identify objects between frames. Using a combination of the Kinect and Optitrack retro-reflective marker trackers, we can detect touches, objects and contours (using the Kinect) as well the position, orientation and visibility of objects (using the Optitrack). In the future we envision that RGBD cameras combined with computer vision algorithms will replace the need for the 6DOF sensing technology currently used. The system runs ~56fps during use. The Kinect faces down onto a tabletop where all controls are initially authored.

To illustrate how to use the system, we shall use an example of navigating a slideshow using whiteboard marker. Under normal usage, the selection of the function/system to control would be based on the user's current context.

As a means to "boot strap" our system, we use a Griffin PowerMate (supporting a button, rotation sensor and blue LED) as the initial means of input, however we do not utilize the rotation function (the justification for the button is provided in Section 4). To provide feedback to the user regarding when the system expects input via the button or touch input, the button's LED glows (1Hz) when the button can be used. In the future, we will investigate other modes of engaging the system that do not require an external input device.

To create a new input control, the user first presses the button. The different functions available for control (defined and grouped hierarchically in an XML file, discussed later) are then displayed as buttons. The user then touches the function they wish to create a control for. In this example, the user would select the 'PowerPoint' group and select the 'Next Slide' function.

It is at this point the system requires an understanding of what application function the user requires a new control for. Depending on how many parameters the function requires (also defined in the XML), different input controls can be used. For example, setting the volume would require a parameter from a valuator. In our slideshow example, the "Next Slide" function does not require any parameters. The different options available for controlling

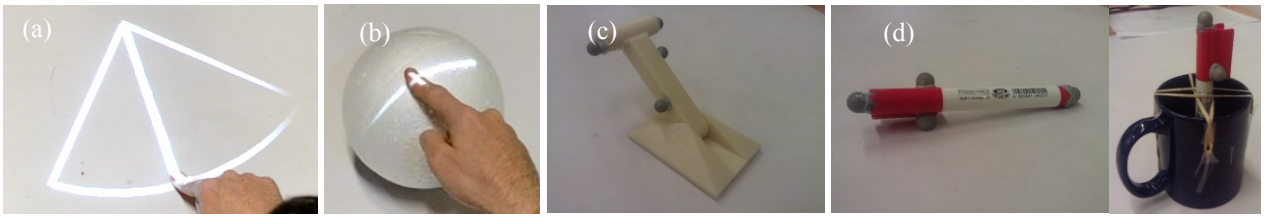

Figure 1: Interacting with dial control (a), slider control on an object (b), interactive lever (c) and improvised joystick (d)

that function are presented to the user as projected touch buttons. Upon selecting one, the user is guided through creating that control. For our "Next Slide" function, we select "Object Orientation" and place the marker we want to use on the table and then press the button to confirm the object selection. The system then prompts the user to orientate the object and the press the button. We hold the marker and point to the right side of the room and press the button. This links this orientation of the marker to the "Next Slide" function, allowing us to point the marker to the right to navigate to the next slide. This means we can now walk around the room, taking the user control (marker) with us, navigating the slides as required, something not possible with previous touch based systems. The same process can then be repeated for going to the previous slide. The whole process of creating a new control takes only seconds and single object can be used in multiple interactions simultaneously, e.g. a marker used as a joystick (Figure 1d) to define both an X and Y value.

When creating projected controls, the process involves the user using their finger to define that control on a surface. For example, to define a dial control the user touches the center of its location and drags out the radius and then continues dragging to define the size of the dial's arc/circle. Projected controls can utilize any physical objects as part of the interaction (e.g. a lever's handle can trigger a virtual button).

To edit controls already created, the user holds the button for more than one second. Projected controls then begin to wobble in a similar fashion to the press-hold-wobble interaction on mobile devices. Users can then touch and drag controls around the table, or drag them off the bottom of interaction area to remove them. To edit object-based interactions, the involved object is placed in the middle of the table, at which time the system presents buttons for each interaction involving that object. These buttons can then be dragged off the table to delete the interaction associated with that object.

3.1 Example Applications

We have created a number of example mappings to control different applications across a number of domains to demonstrate the system's functionality.

Video Editor: The user views the video on an external screen, with the system creating controls for the timeline and cutting/joining sections of the film. The most basic controller would be a slider with (at least) two buttons for cutting/saving the film (Figure 2), but could be more elaborate using a guillotine prop to 'cut' the film and another to join it. Different video clips to be split can be associated with different objects, allowing the user to rapidly switch between clipping/joining different files.

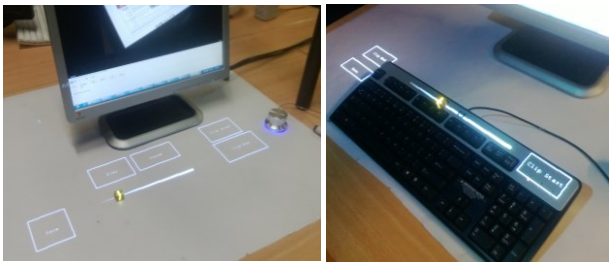

Figure 2: Ad-hoc video editing controls on their own (left) and supplementing the existing controls (right)

Audio Mixer: Allows the user to load and control media whilst adjusting individual audio channels and settings, creating an on demand, customizable DJ-style mixing board. What is novel is the user can create as many controls as required for the particular task, and destroy them when not required. Because the system is not limited to vertical or horizontal controls, the channels could be linked to dials, sliders and levers, etc., located at different positions and orientations surrounding the user, instead of having controls laid out in a linear fashion. Tangible objects provide persistence, visual feedback, and tactile feedback. The use of an application supporting MIDI mappings would enable integration with thousands of existing applications outside of just PC audio applications.

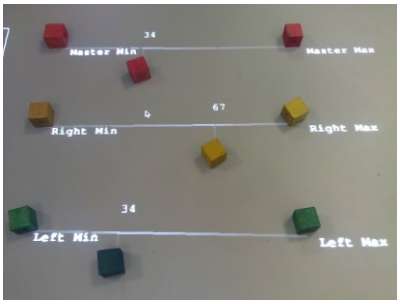

Figure 3: Basic multichannel ad-hoc audio control board

Game Controller: Allows customizable game controllers to be created. Given the ability to use passive objects as active input, users can use a child's pretend steering wheel as an ad-hoc means of controlling racing applications, such as with the AR simulation by Oda et al. (2007).

Since controls can quickly be created with arbitrary materials and turned into functional interfaces, another application is for developing user controls without having to integrate electronics with each iteration, e.g. using 3D printers. Figure 1c depicts a 3D printed throttle-style lever found in airplanes that could work as a functional input device for a flight simulator, without requiring modifications to the game. Despite this application not directly being ad-hoc, we can still leverage near-by materials to quickly create such controls in an ad-hoc manner, where inputs are dynamically created on the fly from passive materials.

We envision applications to developing large scale user controls. By utilizing a realistic simulator using the required mappings, we can design industrial control rooms whilst controlling a working simulation with passive input controls. This allows the user to experiment with configurations for different scenarios (e.g. day-to-day versus an emergency) inside the simulator, creating controls as needed.

4 Preliminary Study

A preliminary study was conducted in the initial design stages to evaluate how users would ideally create controls to interact with existing systems. This study was similar to

that used by Henderson and Feiner (2010) for OCs. Participants were given a number of everyday arbitrary objects (blocks, pens, smart phones, scissors, etc.) and asked to create UIs to control different tasks (selection, text entry, path definition, etc.) across different applications (both within and outside the users reach) using three types of UIs: touch, passive tangible and active tangible. Participants were surrounded by writable surfaces (whiteboard and paper covered surfaces) and asked to create controls for the tasks using the materials available. They were told to assume the system was 'all seeing' and asked to sketch out their ideas, experiencing a Wizard of Style evaluation. They were asked to describe the order they expected to be able to performed certain interactions, what navigation aids should be present and when/how to edit existing inputs, etc. Devices, menus and other content described was created using available materials. By evaluating the different types of input devices the users constructed from the available materials, as well as the manner and order in which they constructed them, we evaluated the types of ad-hoc controls the system should support, as well as how they expected to be able to create them within the system. Approximately half of the participants had a computer-science background.

When asked about the procedure for creating controls, participants responded that the system should enable the user to select the function to control first, followed by selecting the input device and then how that device is mapped to the function. It was mentioned that the main thing they were thinking about was what function to control, and thus needed to "offload" that information into the system as soon as possible. This supports the workflow suggested at the conclusion of the work on OCs (Henderson and Feiner, 2010). When asked how the user should be able to select the function to control from a large set, participants said that functions should be able to be grouped, with the user first selecting the function group, then selecting the function itself.

For the primary means of navigating the system, most participants wanted a different form of interaction than that supported by the system, i.e. use of a physical button instead of a touch-based button if interacting by touch. This was described as helping separate defining controls versus navigating the system. The workflow of the final system was followed these results.

Participants used both traditional touch controls (buttons, dials, etc.) and proxemic relationships (between and within objects). Occasional hybrids were created where a tangible object would interact with a touch-based control, triggering the input, in addition to the tangible object's own explicit input, e.g. a lever handle touching a virtual button.

The different types of touch-based controls and interactions using objects served as the first types of interactions that were implemented in the system. The study also served as inspiration for how the user should be able to navigate the system and the information flow between user and system for creating interactions.

5 Supported User Controls

By sensing different types of user actions with physical objects and extending touch interaction to use arbitrary surfaces, we can create a functionally comprehensive set of UI controls to enable the user to both control existing computer applications and create new input devices in their

		Physical						Touch
		Position				Orientation		
		Object Between Positions	Object in Position	Object is Visible	Object Proximity	Object in Orientation	Object Between Orientations	
		Continuous	*Boolean*	*Boolean*	*Boolean and Continuous*	*Boolean*	*Continuous*	*Boolean and Continuous*
Controls	**Button**		☑	☑	☑	☑		☑
	Radio Button	☑	☑		☑	☑	☑	☑
	Slider	☑			☑		☑	☑
	List Box	☑	☑			☑	☑	☑
	Spinner	☑	☑		☑	☑	☑	☑
	Menu		☑	☑		☑	☑	☑
	Tab		☑	☑		☑	☑	☑
Input Device	**Mouse**	☑					☑	☑
	Keyboard							☑
	Joystick	☑					☑	☑
	Steering Wheel	☑					☑	☑

Table 1: Mapping existing controls GUI/physical against how they can be controlled in the system (☑ Supported)

own right. Our initial demonstration of the concept employs the position, orientation, visibility or proximity of one or more physical objects and emulate touch interaction on objects and surfaces using a depth camera. By monitoring these types of interactions, we can support all the types of controls and interactions described in the preliminary study. Table 1 describes how our initial set of sensed user actions can be used to emulate and recreate a wide range of traditional system inputs. The column headings describe the different capabilities for input detection in our implementation, supporting both Boolean and continuous values. For a nullary function (one with no parameters, such as a button press) we sense one of the following properties of the object in relation to the sensed working space: visibility, absolute position (3D), and absolute orientation (all three angles), two objects within a set proximity as well as supporting touch-based buttons. For a single continuous value, the following relative geometric relationships are available: the position of an object relative to two 3D points, the orientation of an object relative to two defined start and end angles, distance between two objects as well as the value of a touch-based slider or dial. Functions requiring two parameters can utilize a projected touchpad.

To enable extensibility, external applications can provide input to the system as if it were native input. These external applications either call an application "SendInput.exe" or connect to the system via TCP socket to send data. The external system sends a keyword to uniquely identify the input as well as any parameters for that input, e.g. the value from a joystick as "joystick 23 60" for the X and Y values. In the case of calling SendInput.exe, the keyword and values are just passed as arguments when the application is executed. This simple approach allows external systems to integrate other capabilities not currently supported, such as gesture, voice, pressure, light, etc. with minimal code.

In Table 1, the checked boxes indicate particular GUI controls that are currently supported and implemented in either touch or tangible form. While we could conceive controls for every position in the table, the checkboxes represent the "sensible" interactions. Using data from 6DOF tracking and depth camera systems, we can map the input sensed as controls to both control existing applications and to emulate/create physical input devices.

5.1 GUI Control Substitution

As discussed previously, a user can quickly create new controls in only three-to-four steps to compliment/replace existing functionality currently controlled by different GUI elements. The top half of Table 1 includes lists how the input of various traditional GUI controls can be created using geometric relationships both within and between objects as well as virtual controls (buttons, sliders, etc.) projected by the system. We have developed a wide range of controls available to users as disposable UI elements. We elaborate on a subset of the developed EUI's based on exiting GUI controls here:

(Toggle)Button: To simulate a button, we can use any tracked object and create an interaction such that when the object is in a certain location (*Object in Position*), the associated button is 'pushed'. For toggle buttons, the persistence of the physical object's presence naturally supports the button's current state. To activate a push button with a physical object multiple times, the object has to be sensed, removed, and sensed a second time. Similarly, the orientation of the object may be used to indicate the state of a button, such as for the PowerPoint "Next Slide" example discussed previously (*Object in Orientation*).

Slider: The user can employ a tracked object(s) and define two positions as start and end positions. Depending on its current position between them (*Object Between Positions*), a value is passed to emulate a slider with that value. This slider is visualized as a projected path or a linear slider with two physical objects as end points (Figure 4). The projected path may be linear, a high order curved path, or an arc. The path can be tracked on 2D or 3D surfaces.

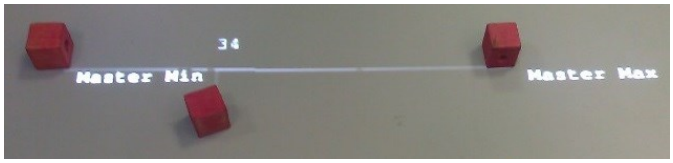

Figure 4: Objects used to as a slider control for volume

Radio Button: Given a set of options, we can use an object's orientation in a single axis (*Object Between Orientations*) to select an option. For each different orientation, a different radio button is selected (Figure 5). Since the object can only have one orientation at any time, only one option can ever be selected. This approach could also be used to emulate a dial. Likewise, the position of a single object may be employed to indicate which radio button is on.

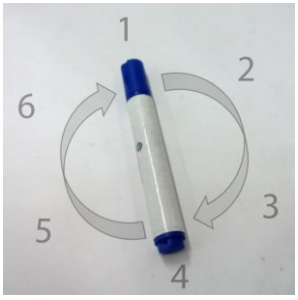

Figure 5: Object used as radio button (digital overlay added for illustration)

List Box: Given a list of items (similar to a menu), the user can use a tracked object, e.g. a pen, and set two different orientations on the table plane, the rotation of the object between those orientations can then be used to interpolate a value and select the appropriate index in the list.

Spinner: A spinner has a small set of discrete values. The user can use the proximity of objects (Figure 6) to set the

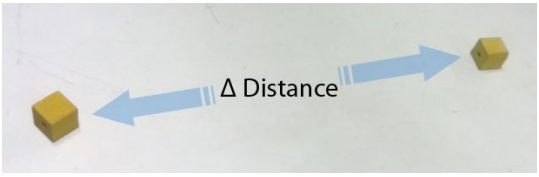

Figure 6: Using physical proximity as valuator (digital overlay added for illustration)

value for the spinner. The distance between two objects may be employed to set the value for an individual spinner value (*Object Proximity*). As the object moves further away, the value increases.

Menu: To emulate a menu system such as a pie menu, we can use the orientation of a fixed object at different rotations to select different items, similar to the radio button functionality. Rotating the object means selecting a different menu item. Given any number of different menu options, we can also utilize projected touch-based buttons.

Tab: To switch to different tabs, the user can associate different tracked blocks with each tab. To switch tabs, the user places one of the tracked objects into the workspace to make it visible (*Object is Visible*). The associated system tab is then selected.

5.2 Input Devices

Using the sensing capabilities of the system, the user can create active user controls from completely passive components. These UIs can supplement or replace existing input devices with user defined ones. We are interested in investigating controlling more complicated interactions instead of just emulating GUI-like elements. The bottom half of Table 1 outlines how some common input devices can easily be created using the system. The system also enables the user to create input controls in place of existing input devices, including the following:

Mouse: As per the Slider example above, the user can define a 2D area using two perpendicular sliders utilizing a common position (an 'L' shape). We can then use the position of an object for each slider to define both the X and Y position of the cursor. The control of the cursor can also be in relative scaled coordinates, similar to a touchpad device.

Keyboard: Using touch on the tabletop surface, a user could emulate a software keyboard.

Joystick: As a tangible example, using a simple whiteboard marker, we can define a joystick (Figure 1d) using the orientation across two different orientation axis (x-min left, x-max right, y-min down, y-max up) and immediately control any number of games. Using a cup and two rubber bands, we can quickly improvise a self-centring joystick capable of controlling applications.

Steering Wheel: We can use any circular object tracked with reflective markers and use the angle between two defined orientations (rotated left and right extremes) as the input value. This provides input akin to the Wii console's steering wheel controller.

Figure 7: Passive steering wheel used as an active input device using tracking markers (visible on top)

6 System Design

To support ad-hoc interaction, we extended the TAM architecture (Walsh et al., 2013). This work focused on programming the logic of tangible interactions, the 'how' of the interaction, whereas we focus on the 'what' of the interaction. One study participant described this as telling the system, "what to do, not how to do it". As such, our work assumes the system already has a set of predefined functions, and instead focuses on how control those functions at run time. In addition, the previous work does not support interacting with external systems and is designed for all interactions and feedback to take place within the system.

The previous architecture physical objects as InteractionObjects (InObjs) with associated Properties (position, color, touch points, etc.). Different Action objects evaluate the Properties according to a given criteria (e.g. rotation around an axis for a rotation input). Using that Boolean result, an Interaction object monitors when the Action occurs, and modifies any number of Properties of different objects as a result. By using the properties of physical objects (location, orientation, etc.) we can leverage them as input for existing systems.

We extend the architecture in four ways: 1) allow Actions to have some form of native representation to indicate their current state, 2) allow Actions to have Properties to communicate a non-boolean state to other components, 3) introduce support for VirtualPropeties as a way to communicate with external systems without incorporating any system-specific code in the core application, and 4) allow external systems to pass information into the system and use that information as an input for the internal patch panel as if it were a normal input from the user.

6.1 Allowing Actions to Have a Representation

Whilst the original architecture was focused on purely tangible interactions, this work has focused on a physical/projected hybrid. Actions that monitor input need to be able to report some kind of state, e.g. buttons not only need to register for a press, but have some representation (i.e. a projected button). Given the Action object evaluates input, it is the only component that is aware of the context of the value (i.e. is the value based on distance, rotation etc.?), it must be responsible for creating any representation for that Action. As such, we assign Action a method to render its state in some form, e.g. in our implementation using OpenGL. This representation is generated based on its current state. For example, a button would render the button display (changing if pressed), or lines to indicate the distance between objects, etc.

6.2 Allowing Actions to Have Properties

Given the purpose of the Action component to monitor the state of an interaction, we add any number of Properties to it to represent its current state and configuration. This value is then read as part of an Interaction, and used to update Properties of other objects.

6.3 VirtualProperties for External Functions

To integrate with existing systems, we require some external communication method. A VirtualProperty (VP)

was created to enable communication of a value represented inside the system to an external application. These VPs are associated with single Interaction. A VP takes in a string containing an application name (and required parameters, if any), to run when the Property is set. This string is passed when the VP is created, and could itself be a Property that can be edited at run time. It was thought that by executing an application instead of an API call, we simplify the system by excluding API libraries from the core system, without losing any functionality. This approach also allows the integration of existing applications that can be run/controlled using the command line and allows our system to be used by non-developers.

To format a user control's output as a valid form of input for a specific application, VPs contains a scaling and format setting for how the data should be transformed before the application is executed. This scaling operation includes: minimum, maximum, user defined range, and no modification options. For example, to use the rotation of an object as a 6-value radio dial (Figure 5), you would define a range of 1 (min) and 6 (max). The rotation of an InObj would then be transformed to the range 1-6. Using this, valuators such as sliders or dials can be used to give a value across any given range or even to a Boolean value.

For Actions that provide persistent values, e.g. using the position of an object, VP also contains a rate limiting timer (defined in milliseconds) to restrict the rate of execution for the related application. This defaults to zero for no limit.

All functions that can be controlled are defined (Figure 8) in an XML file, with functions grouped in any arbitrary manner (following the results of the preliminary study). Groups are used to define logical sets of related functions for a given task, meaning they may not all interact with the same application. Rather, they all interact with different applications to manage a common task. As such, functions can appear multiple times across any number of groups. Each XML definition defines a program to be executed when an attribute is set, along with how values pass to that program should be limited/scaled. A user-friendly name is also provided and used within the system to identify that mapping.

```
<output execute="joystick.exe setX %i" ratelimit="0"
scale="range" min="0" max="1080" name="Joystick X" />
```

Figure 8: Example mapping of a single axis of a joystick

6.4 Allow External Systems to Send Data

Our system allows external applications and systems to act as native input controls. For example, our current implementation only senses touch and geometric relationships as forms of input. Using this approach, an application that monitors pressure exerted on digital foam, or the user's voice or gestures can pass this information into the system as a valid control to be utilized by the user.

In a similar approach to enabling external functions, we use a string to integrate external systems for passing information into the system. Input from external systems is supported through a VirtualAction object. These triggers are defined in a similar XML format to outputs, defining a user readable name, keyword and what parameters are provided. This allows external systems to send a string via TCP in the format "*triggerkeyword parameter1value*". In addition to TCP, we wrote a simple application (SendData.exe) that, when executed, directly passes data arguments into the system as if it were sent via TCP.

An associated VirtualAction produces this value as if it were sensed natively, thus registering as a valid input (like a button, etc.). Because this interaction appears as normal input, it can be mapped back to external applications using VP's, only to be read in again. This ability to both read and write to external systems enables the creation of feedback loops (Figure 9), creating support for embodied OUIs.

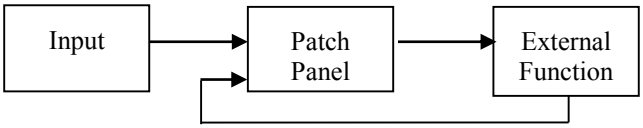

Figure 9: Information feedback loop

This approach simplifies adding new controls to the system, removing the requirement to write system-specific code. For example to integrate Phidgets, you could simply take the example programs and add one line of code to execute SendData.exe and pass in a keyword and the value from the Phidgets. Any new type of input not currently supported by the system (pressure, sound, light, voice, etc.) can easily be incorporated as if it were a native capability of the system. This also enables existing applications that can be controlled via the command line to be controlled via these ad-hoc controls, allowing the user to incorporate new functionality without writing any code, a point of difference when compared to previous homogeneous systems.

7 Future Work

A full evaluation of our implementation remains as future work. Whilst this work enables the ad-hoc creation of tangible controls for existing functions, further work remains in the area of temporal and direct manipulation interactions for virtual content. In addition to this, the object-based interactions do not currently support the full set of proximity based relationships between multiple objects (e.g. an angle between two objects). These could be implemented to ensure full support of proximity based relationships both within and between objects, as used in the Proximity Toolkit (Marquardt et al., 2011).

8 Conclusion

In this paper we have presented our system to support ephemeral interaction using everyday objects and an architecture to support their ad-hoc creation, including the incorporation of new types of input and functionality not supported by the original system. This system supports creating controls to simulate input from existing GUI controls as well as supporting the creation of novel tangible input devices made from passive components. Using design decisions employed from a preliminary study, we have presented an example system and techniques for enabling end users to create a wide range of arbitrary user controls, both tangibly and virtually, to control existing functionality.

9 Acknowledgements

The authors would like to thank the reviewers for their feedback and thoughts and regarding the paper.

10 References

AKAOKA, E., GINN, T. & VERTEGAAL, R. 2010. DisplayObjects: prototyping functional physical interfaces on 3d styrofoam, paper or cardboard models. *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction.* Cambridge, Massachusetts, USA: ACM.

AVRAHAMI, D. & HUDSON, S. E. Forming interactivity: a tool for rapid prototyping of physical interactive products. Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques, 2002 London, England. ACM 141-146.

BALLAGAS, R., SZYBALSKI, A. & FOX, A. 2004. Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments. *In:* ANDY, S. & ARMANDO, F. (eds.) *Second IEEE International Conference on Pervasive Computing and Communications.* Orlando, Florida.

DORING, T., SYLVESTER, A. & SCHMIDT, A. 2013. A design space for ephemeral user interfaces. *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction.* Barcelona, Spain: ACM.

FAILS, J. A. & OLSEN, D. 2002. Light widgets: interacting in every-day spaces. *Proceedings of the 7th international conference on Intelligent user interfaces.* San Francisco, California, USA: ACM.

FITZMAURICE, G. W., ISHII, H. & BUXTON, W. A. S. 1995. Bricks: laying the foundations for graspable user interfaces. *Proc. of the SIGCHI conference on Human factors in computing systems.* Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co.

GREENBERG, S. & FITCHETT, C. 2001. Phidgets: easy development of physical interfaces through physical widgets. *Proc. of the 14th annual ACM symposium on User interface software and technology.* Orlando, Florida: ACM.

HARDY, J. & ALEXANDER, J. 2012. Toolkit support for interactive projected displays. *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia.* Ulm, Germany: ACM.

HENDERSON, S. & FEINER, S. 2010. Opportunistic Tangible User Interfaces for Augmented Reality. *Visualization and Computer Graphics, IEEE Transactions on,* 16**,** 4-16.

HENDERSON, S. J. & FEINER, S. 2008. Opportunistic controls: leveraging natural affordances as tangible user interfaces for augmented reality. *Proceedings of the 2008 ACM symposium on Virtual reality software and technology.* Bordeaux, France: ACM.

HOLMAN, D. & VERTEGAAL, R. 2008. Organic user interfaces: designing computers in any way, shape, or form. *Communications of the ACM,* 51**,** 48-55.

HUDSON, S. E. & MANKOFF, J. Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. Proceedings of the 19th annual ACM symposium on User interface software and technology, 2006. ACM, 289-298.

ISHII, H. & ULLMER, B. 1997. Tangible bits: towards seamless interfaces between people, bits and atoms. *Proc. of the SIGCHI conference on Human factors in computing systems.* Atlanta, Georgia, United States: ACM.

KJELDSEN, R., LEVAS, A. & PINHANEZ, C. 2003. Dynamically Reconfigurable Vision-Based User Interfaces. *In:* CROWLEY, J., PIATER, J., VINCZE, M. & PALETTA, L. (eds.) *Computer Vision Systems.* Springer Berlin/Heidelberg.

KLEMMER, S. R., LI, J., LIN, J. & LANDAY, J. A. 2004. Papier-Mache: toolkit support for tangible input. *Proc. of the SIGCHI conference on Human factors in computing systems.* Vienna, Austria: ACM.

MARQUARDT, N., DIAZ-MARINO, R., BORING, S. & GREENBERG, S. 2011. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. *Proc. of the 24th annual ACM symposium on User interface software and technology.* Santa Barbara, California, USA: ACM.

ODA, O., LISTER, L. J., WHITE, S. & FEINER, S. 2007. Developing an augmented reality racing game. *Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment.* Cancun, Mexico: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

SHARLIN, E., WATSON, B., KITAMURA, Y., KISHINO, F. & ITOH, Y. 2004. On tangible user interfaces, humans and spatiality. *Personal and Ubiquitous Computing,* 8**,** 338-346.

ULLMER, B. A. 2002. *Tangible interfaces for manipulating aggregates of digital information.* Doctor of Philosophy, Massachusetts Institute of Technology.

VILLAR, N., BLOCK, F., MOLYNEAUX, D. & GELLERSEN, H. 2006. VoodooIO. *Proc. of ACM SIGGRAPH 2006 Emerging technologies* Boston, Massachusetts: ACM.

WALSH, J. A., ITZSTEIN, S. V. & THOMAS, B. H. 2013. Tangible Agile Mapping: Ad-hoc Tangible User Interaction Definition. *In:* SMITH, R. & WUENSCHE, B. (eds.) *Australasian User Interface Conference.* Adelaide, Australia: CRPIT.

XIAO, R., HARRISON, C. & HUDSON, S. E. 2013. WorldKit: rapid and easy creation of ad-hoc interactive applications on everyday surfaces. *Proceedings of the 2013 ACM annual conference on Human factors in computing systems.* Paris, France: ACM.