OnObject: Programming of Physical Objects for Gestural Interaction

by

Keywon Chung

B.F.A., Carnegie Mellon University, 2001

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Keywon Chung
Program in Media Arts and Sciences,
School of Architecture and Planning
August 6, 2010

Certified by .....
Hiroshi Ishii
Muriel R. Cooper Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by.....
Pattie Maes
Chairperson, Departmental Committee on
Graduate Students Program in Media Arts and Sciences

Abstract

Tangible User Interfaces (TUIs) have fueled our imagination about the future of computational user experience by coupling physical objects and activities with digital information. Despite their conceptual popularity, TUIs are still difficult and time-consuming to construct, requiring custom hardware assembly and software programming by skilled individuals. This limitation makes it impossible for end users and designers to interactively build TUIs that suit their context or embody their creative expression.

OnObject enables novice end users to turn everyday objects into gestural interfaces through the simple act of tagging. Wearing a sensing device, a user adds a behavior to a tagged object by grabbing the object, demonstrating a trigger gesture, and specifying a desired response. Following this simple Tag-Gesture-Response programming grammar, novice end users are able to transform mundane objects into gestural interfaces in 30 seconds or less. Instead of being exposed to low-level development tasks, users are can focus on creating an enjoyable mapping between gestures and media responses. The design of OnObject introduces a novel class of Human-Computer Interaction (HCI): gestural programming of situated physical objects.

This thesis first outlines the research challenge and the proposed solution. It then surveys related work to identify the inspirations and differentiations from existing HCI and design research. Next, it describes the sensing and programming hardware and gesture event server architecture. Finally, it introduces a set of applications created with OnObject and gives observations from user participated sessions.

Thesis Supervisor: Hiroshi Ishii

Title: Muriel R. Cooper Professor of Media Arts and Sciences

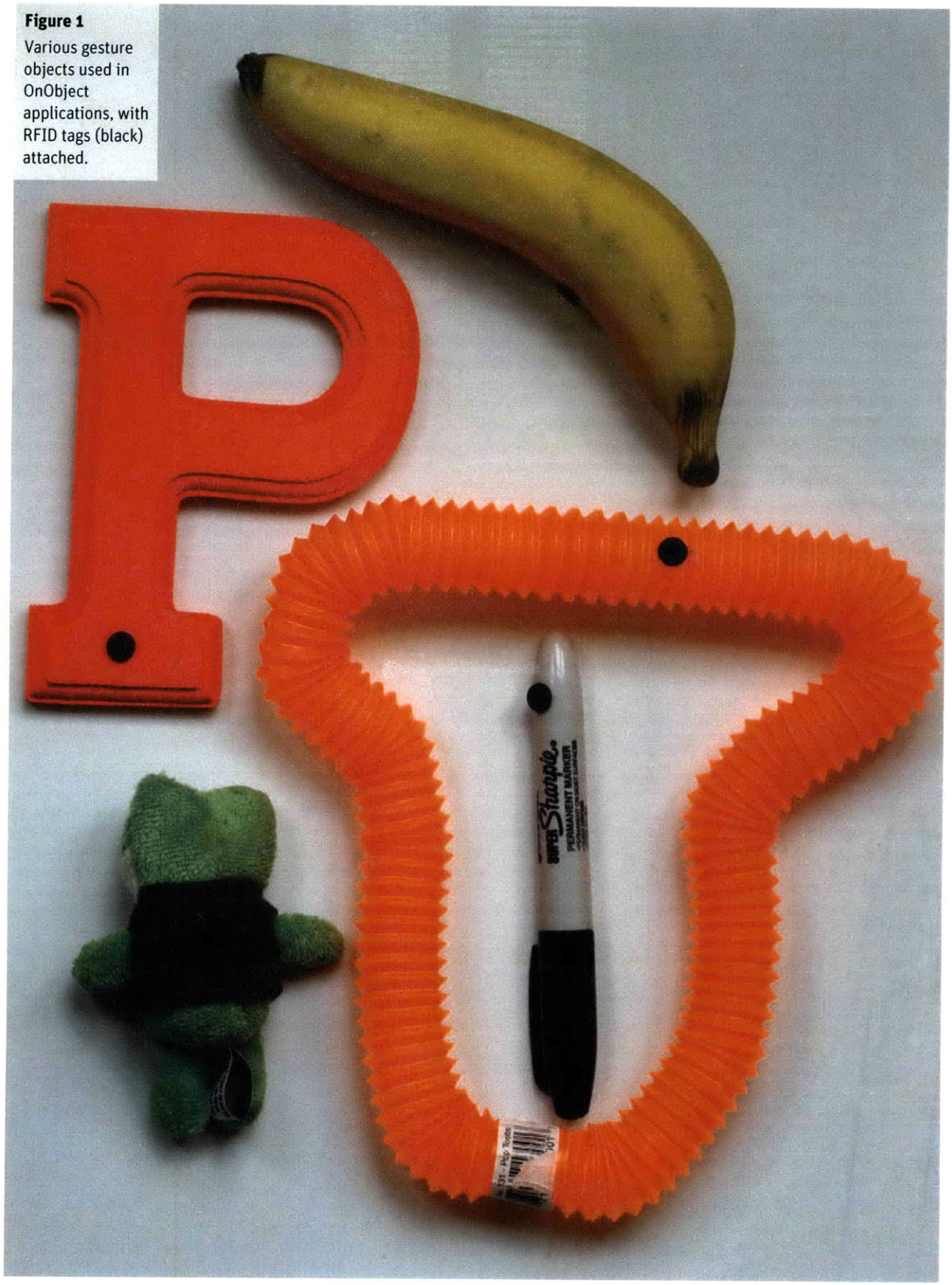


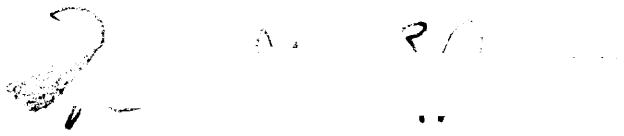
Figure 1
Various gesture objects used in OnObject applications, with RFID tags (black) attached.

OnObject: Programming of Physical Objects for Gestural Interaction

by

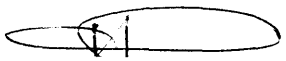
Keywon Chung

The following people served as readers for this thesis:



Thesis Reader .....

Hiroshi Ishii
Muriel R. Cooper Professor of Media Arts and Sciences
MIT Media Lab



Thesis Reader .....

Pattie Maes
Muriel R. Cooper Professor of Media Arts and Sciences
MIT Media Lab

Thesis Reader .....

Desney Tan
Senior Researcher and Research Manager, Microsoft Research
Affiliate Assistant Professor, University of Washington

Acknowledgment

A big gratitude goes to my advisor Professor Hiroshi Ishii, thanks to whom I have had invaluable two years of exploring and executing my personal vision of bottom-up, user-defined uiquitous computing. Thank you for lending valuable family play time as well.

And I would like to thank...

... Thesis readers Professors Pattie Maes and Desney Tan, for their concrete suggestions and letting me get a glimpse at their research aesthetics.

... Collaborators who helped make this idea come alive: Michael Shilman, Chris Merrill, E Roon Kang, Min Lee, and Bosung Kim; technical advisors Rich Fletcher and Selene Mota, for walking me through the RFID antenna design and the Wockets gesture recognizer; logistics maven Lisa Lieberman and Sarah Hunter, who took care of the many underlying needs.

... Colleagues and friends, for their research, prototyping, and morale guidance: Cati Vaucelle, Daniel Leithinger, Jean-Baptiste Labrune, Adam Kumpf, Sean Follmer, Xiao Xiao, Jinha Lee, Jaewoo Chung, Kimin Jun, Jaebum Joo, Sigurður Örn, Carnaven Chiu, Peggy Chi, Richard The, Sey Min, Seth Hunter, Andrea Colaço, Inna Lobel, Sohin Hwang, Jaekyung Jung, Kyunghee Kim and Nicole Shumaker.

... My parents and family in Korea who supported my learning, and my parents-in-law in California who tolerated my absence, thank you.

And again my husband, machine learning teacher, personal trainer, and proof reader Michael Shilman: let's celebrate the end of the long distance tunnel, the soju is on me.

Content

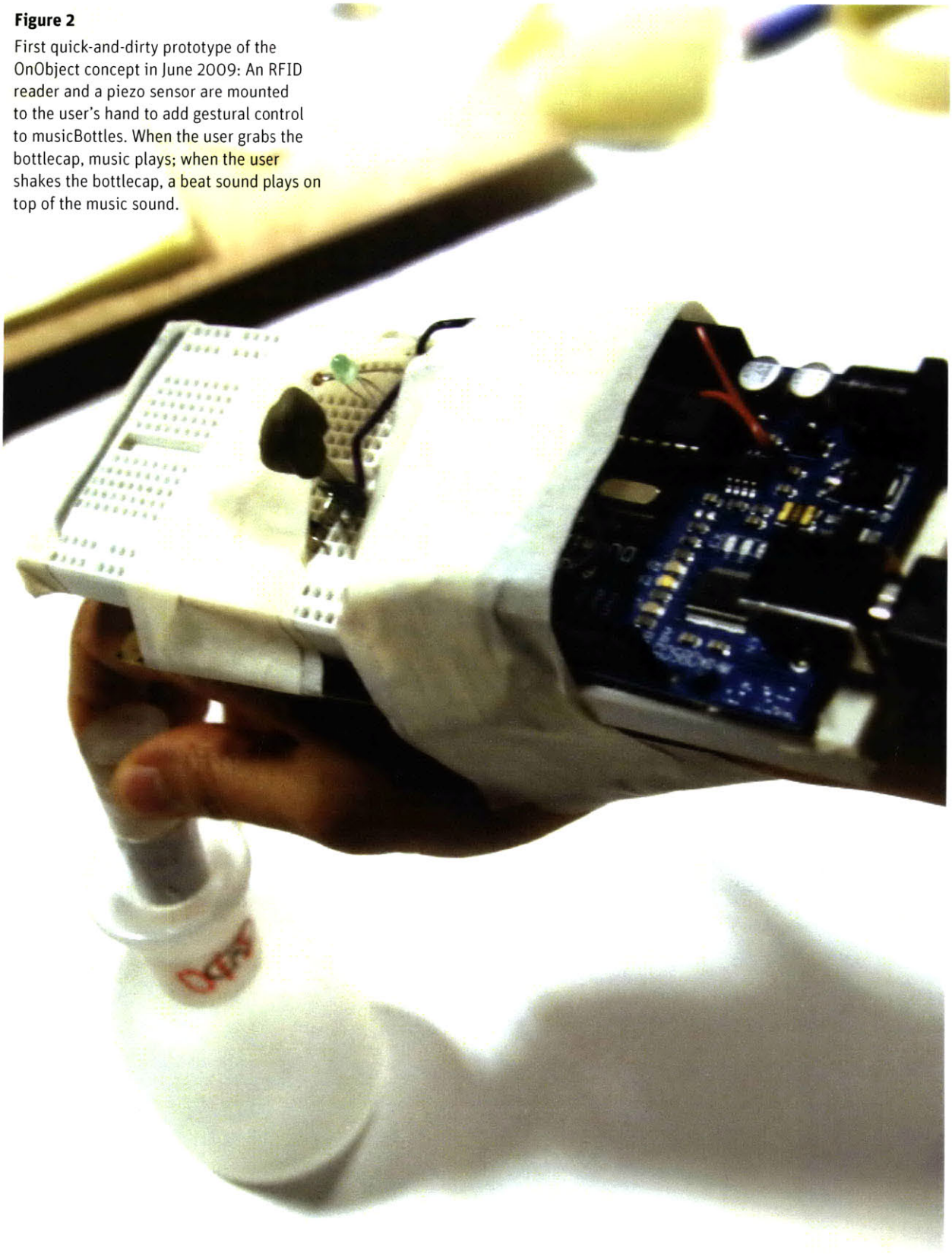
Chapter 1

The Idea

This chapter outlines the challenges this research aims to address, introduces the OnObject approach used to solve the problems, and highlights the novel interaction and research contributions from the approach and implementation.

Figure 2

First quick-and-dirty prototype of the OnObject concept in June 2009: An RFID reader and a piezo sensor are mounted to the user's hand to add gestural control to musicBottles. When the user grabs the bottlecap, music plays; when the user shakes the bottlecap, a beat sound plays on top of the music sound.



1.1 The Problem

While many Tangible User Interface systems employ sensor-equipped physical objects, they do not easily scale to situated everyday objects: the process requires significant modification or custom fabrication of the host objects, hardware assembly, and software coding by skilled individuals. Such involved development process not only limits participation from novice end users, but results in pre-meditated mapping of input and output events and user interaction that cannot be easily modified to reflect the end user's wishes and circumstances.

1.1.1 Interacting with Situated Objects

Historic and recent Tangible User Interfaces (TUIs) have incorporated common physical objects, from Marble Answering Machine, musicBottles, to I/O Brush, reacTable and Amagatana+Fula [17, 9, 20, 11, 12]. Everyday objects are not only easy to acquire and understand, but also can be more meaningful because they are repeatedly used in context and are associated with certain notions and experiences individual users value. Nevertheless, it has remained elusive to develop of a method for end users to easily co-opt situated objects and couple their states with digital information, whether a sensing platform is used or the sensing hardware is embedded inside the objects.

Objects on a Sensing Platform

Ever since Ishii and Ullmer introduced phicons to allow users to manipulate graphical user interface (GUI) elements through specifically designed physical objects [34], many initiatives have endeavored to generalize platform-based sensing to everyday objects. musicBottles showed that tabletop sensing can be used for emotive experience with a poetic coupling between opening of the bottle and curated music. More recently, many platforms including reacTable, Trackmate and G-Speak have co-opted everyday objects into the user experience by marking them with tags detected by externally-referenced vision and sensing systems [11, 36, 37].

Marker-based systems scale relatively well to a larger number of objects, as you can co-opt a new object by applying a marker to the object and modifying the software to produce desired response to the marker. However, the user experience often lacks the immediacy and physicality of action-based interaction, as the objects are used as simple tokens to abstract information and their physical affordances are not actively leveraged other than the object being moved and rotated as a whole. Besides, objects have to be removed from their natural context to be placed within the sensing area, unless the user possesses a large enough sensing platform to encompass their natural activity areas — a setting that is costly and immobile.

Objects as Custom Housing

A particularly compelling subset of Tangible Interfaces consists of a common physical object housing sensing hardware. These interfaces are intuitive and often delightful thanks to their clear physical affordances and user expectations. Those who encounter the I/O Brush can immediately understand that the interface is to be grabbed by the handle and painted with thanks to their prior experience with paintbrushes. Users are then pleasantly surprised to see the novel capabilities of the brush, as it is augmented with electronic hardware and software. Amagatana + Fula is another recent example where sensor-embedded umbrellas produce sword-like sound effects when swung. Wearable computing projects also consist of garments housing hardware components.

However, while these interfaces take the form of everyday objects, the physical artifacts have to be specially fabricated to enclose the sensing hardware (I/O Brush and wearables) or significantly modified to house them (Amagatana). The augmented objects made with this approach are essentially one-off creations. This approach is far from supporting end users to couple actually situated objects with digital information as they desire.

Input-Output (Re)Mapping

With both approaches mentioned above, an important hurdle is configuring the behavior of a newly-opted physical object. When sensing hardware is housed in the object, every new object requires design and development, fabrication, and testing of the sensing hardware and the housing. This lengthy process renders the quick adaptation of new objects impossible. Even if a sensing platform is used to detect a new object immediately, the user still has to delve into low-level tasks of application coding or analysis of sensor data. The same difficulties arise when user wishes to simply change the mapping between input and output events. As a result, many Tangible Interfaces can only offer pre-defined output responses that cannot be updated or modified reflecting the changing wish or circumstances of the users. With such rigid interaction, adaptation of numerous situated physical objects for varying user desires remains unattainable.

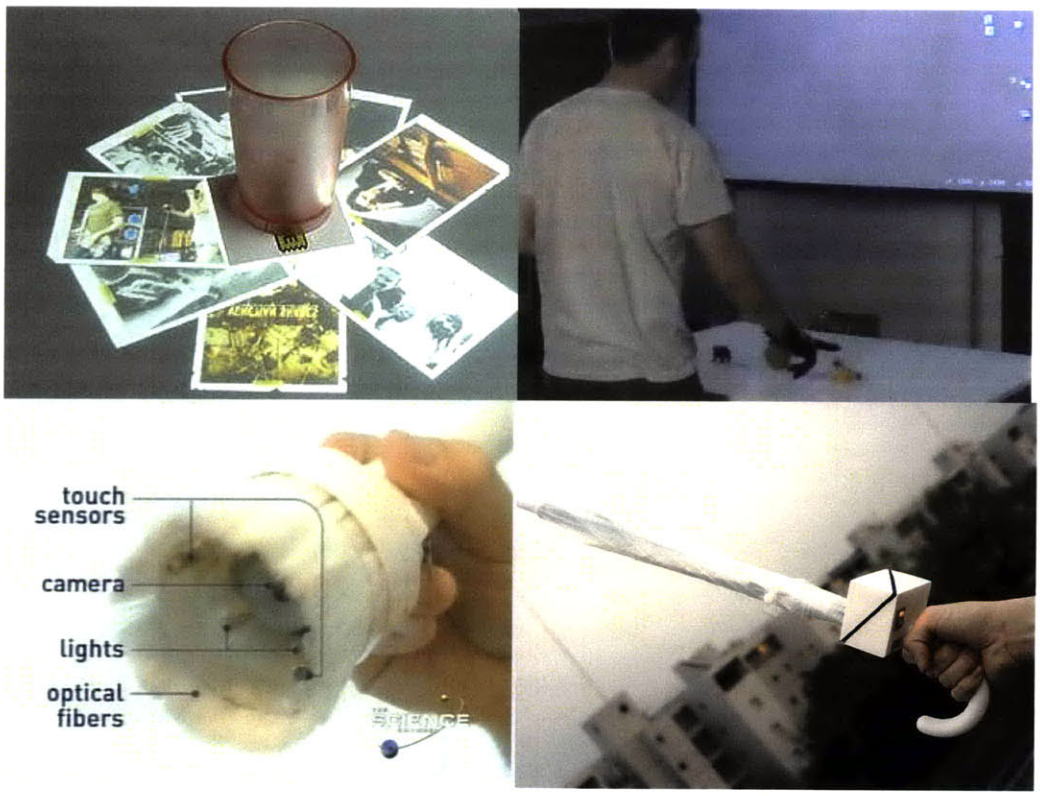


Figure 3 Objects on sensing platforms: Reactable (top left), G-Speak (top right). Objects housing sensing hardware: I/O Brush (bottom left), Amagatana (bottom right).

1.1.2 System Design Criteria

What does it take for end users to create interactive experiences with everyday objects around them? In the scope of this work, we focus on the following four qualities that are not currently met by the common TUI development process.

Low Entry Barrier

It is difficult to expect end users to learn and cope with the coding, analysis and project management tasks to get the seemingly simple enjoyment of hearing sound effects when they open a bottle or sword sound when they swing an umbrella. Echoing this observation, Klemmer et al. noted "developing Tangible Interfaces is problematic because programmers are responsible for acquiring and abstracting physical input. This is difficult, time-consuming, and requires a high level of technical expertise... Consequently, only a small cadre of technology experts can currently build these UIs" [33].

Scale and Propagation

Once one object has been incorporated into an interactive application, it should be considerably faster and easier to add another one like it or a slightly different one. It is worth taking the time to learn how to create a mouse-over animation in Adobe Flash or Processing if one can easily duplicate the files and repurpose them for later use. As stated above, creating another sensor-embedded object like I/O Brush or Amagatana costs considerable efforts, and the task of input-output mapping for the new objects remains too involved for end users to casually engage in.

Room for Quick Experiments

In many creative tasks, creators are encouraged to capture the virtues of momentary ideas and the motivation behind them with a quick sketch, voice notes, or prototypes. When end users have an idea for an interactive object, it is important that they can see it manifested without prolonged distraction. Unfortunately, creators of TUIs must go through multiple phases of tasks away from their original moment of ideation, tending to low-level tasks. Even recently developed toolkits such as Exemplar or d.tools expose users to raw sensor data, forcing users to change their context [6, 7]. The complex and

disjointed process of building a functional interface prohibits timely validation of the initial idea and motivation.

User-Defined Mappings

Despite the original vision of Tangible Bits where TUIs would "augment the real physical world by coupling digital information to everyday physical objects and environments [38]," the rigid TUI design process does not allow end users to define the mapping between input and output events. Dey et al. explained their reasoning behind creating a CAPpella, a system designed to empower end-users in building sensor-based applications [3]:

- End-users have more in-depth knowledge about their activities environments than any developer.
- If only a developer can control system behavior, the user will be unable to evolve the system when her environments or activities change.
- Especially in a context-aware application, user are best suited to build and configure an application to do what they want when they want it.

Carvey et al. asserted that allowing users to define the mapping between physical object and digital output has been often overlooked in the TUI research, and that "for simple everyday tasks that vary and are constantly changing, specifically customized physical objects may not be desired or even practical. [35] "

1.2 Research Goals

OnObject aims to provide a way for novice end users to rapidly transform situated physical objects into individualized gesture interfaces in the context of use, without a lengthy development process.

Rapid Incorporation of Situated Objects

We stipulate that current development process of Tangible Interfaces prohibits end users to quickly co-opt situated objects into interactive experience, and that two challenges lie at the core of this problem: rapid mapping of input user action and system output, and re-mapping of the input and output at will. The goal of OnObject is to provide an end-user-friendly method of mapping input events on a situated object to an output event on the computer.

Simplicity and Expressivity

In order for such a method to be end-user-friendly, the process of creating a TUI application out of physical objects needs to be extremely simple, yet the resulting interaction has to be engaging, satisfying the interaction ideals of Tangible Interfaces with a focus on direct manipulation and bodily engagement.

Many projects have recently created token-based applications with situated objects. For example, the weight-based "Amphibian" platform by Carvey et al. and RFID-based bowl container by Martinussen et al. both trigger playback events when the object is placed on them [35, 40]. While these are intended to be simple to use and the resulting interaction can be amusing or convenient, the binary form of input events (on/off the table) does not provide continuous manipulation and lacks in the dimension of bodily engagement.

OnObject aims to employ bodily motion and direct manipulation of objects as the input events — for novice end users to transform situated physical objects to gesture interfaces in a matter of minutes.

1.3 OnObject Approach

OnObject consists of ID tags easily attachable to physical objects, a hand-worn sensing device, and a set of software to recognize the object in user's hand and the motion gestures the hand makes.

1.3.1 Programming of Physical Objects

Object-oriented graphical user interfaces (GUIs) are a staple of modern computing. Multimedia objects with properties, behaviors, and actions were once the product of complex custom code. After years of development in GUI toolkits, design tools, and programming by demonstration, modern software tools like Flash, Processing, and Scratch [41, 18, 21] make it easier than ever to create rich, on-object user interfaces, often with little or no coding.

An essential concept of GUI programming and object-oriented programming in general is states of an object. Computer programmers can add states to on-screen objects, so that a button knows when it is clicked, and a file knows when it's dragged. While some man-made physical objects like collapsible or folding products have multiple states [42], physical objects do not allow to be "programmed" with new states that can be tracked.

1.3.2 Co-opting via Tagging

How can physical objects be made programmable and recognized by the computer? Physical "markers" like barcodes or infrared dots are used by externally referenced sensing platforms to label physical objects to be a part of the interactive application.

This approach is relatively simple and scales well to increasing number of objects. The notion of "attaching" behavior to graphical objects is already widely used in interactive GUI programming as well, where event listener methods are added to a class or instance of graphical objects. A "tagging" approach is also advantageous for rapid experiments with situated objects, as tagging does not require modification of the object itself and is often reversible. Because of the simplicity associated with physical and online tagging, users are comfortable tagging more objects and reorganizing them as their circumstances change.

1.3.3 OnObject System

OnObject avoids using a spatially-constraining tabletop platform or occlusion-prone and costly computer vision systems. Instead, it aims to position the sensing at the actual locus of the interaction: the user's hand. How can each object and their states be sensed using the hand, and how can digital behaviors be mapped to those states?

Recognizing Object ID and States

The OnObject user wears a device on her hand equipped with a short range 13.56MHz Radio Frequency Identification (RFID) reader to recognize tagged objects she is holding in her hand. The RFID tags most frequently used in OnObject applications are 9mm-diameter plastic tags, easily applicable to most objects using tape or glue. Each tag returns a unique 16-byte ID number.

Also included in the device is one tri-axis accelerometer to recognize the motion gestures user makes with the tagged object in hand (Figure 4). The device sends the RFID tag and accelerometer data to a nearby computer, where the tag ID, gesture ID, and the status of a pushbutton on the device can be used to interact with various applications. In addition to detecting the grab and release of a tagged object, a gesture recognizer has been developed to detect a set of motion gestures – shake, swing, thrust, tilt, circle, and fanning – using a combination of decision tree and Hidden Markov Model. Figure 5 shows the raw RFID and accelerometer data sent from the device.

Figure 4 The OnObject device senses objects user grabs by an attached RFID tag and detects motion gestures the user makes with their hand.

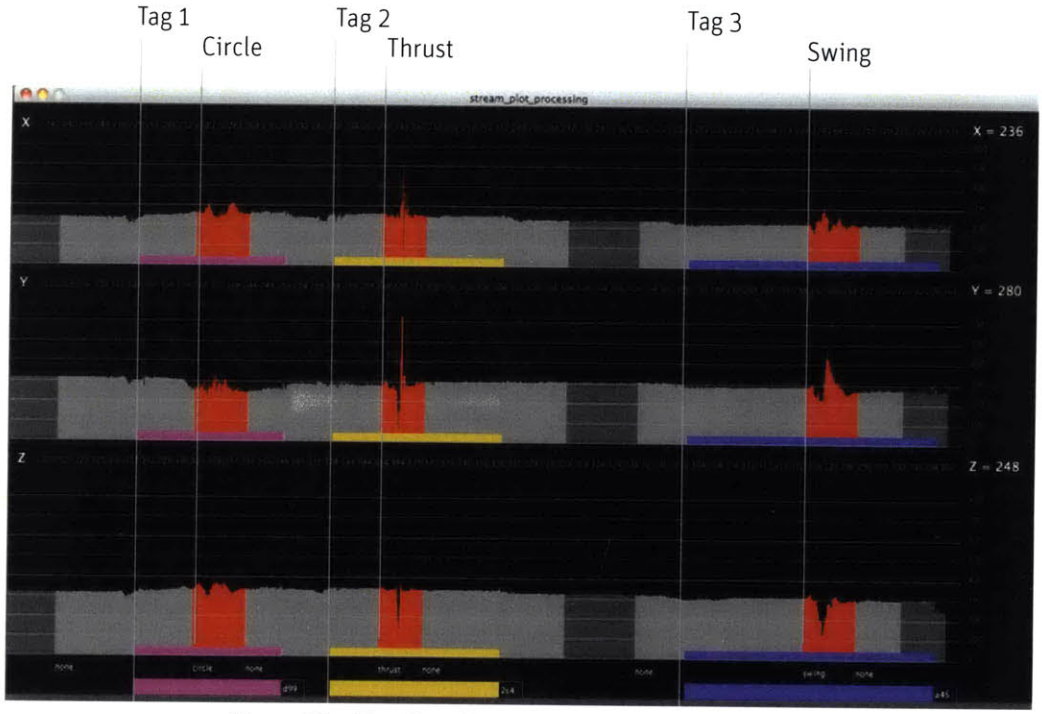


Figure 5 Tag ID and acceleration data coming from the device.

Figure 6 The pre-defined gesture set that can be recognized with OnObject software.

Attaching Digital Behaviors to Object States

For a given tag, the user can program the desired output for each gesture trigger via demonstration, using one of the methods below.

Device Only: For a primary set of applications, the programming is done using only the hand-worn device equipped with a button and a microphone in the following series of steps:

- Tag: The user attaches an RFID tag to a physical object.
- Gesture: With object grabbed by the tag, the user demonstrates one of the trigger gestures and a default sound feedback plays.
- Response: The user holds the button down, records sound response to the gesture by speaking into the microphone, and releases the button.
- Play: When user performs the gesture from then on, the recorded sound plays.

On-Screen Tool: For other applications, the programming is done using the device and a GUI tool. To make a shake gesture on a bell trigger a ripple video to play, user can take the following steps:

- Tag: The user attaches an RFID tag to the bell and grabs it by the tag. The new tag appears on the GUI tool screen on a nearby computer.
- Gesture: The user performs a shake gesture, which is displayed on the screen when recognized.
- Response: The user picks the ripple video file on the GUI screen.
- Play: The user creates an interactive installation where a ripple effect is projected in the room when the bell is rung.

Keystroke Mapping: Alternatively, the above programming can be done by editing a configuration text file where each tag and gesture ID is mapped to an operating system-wide set of keystroke events, so that a large number of software applications can interact with physical objects.

Figure 6 depicts the pre-defined gesture palette, including tag detection gestures and motion gestures.

1.4 Novel Interactions

See Table 1 for novel scenarios and interaction techniques enabled by OnObject. These scenarios will be further describe in the following chapters.

1.5 Contributions

Simplified Method of Co-opting Situated Objects

Using tagging and hand-located sensing platform, OnObject combines a quick and low-commitment induction of new physical objects with practical mobile sensing. This combination supports real life situations thanks to its small form factor and reasonable cost. Through the choice of scalable tag sensing and simple motion sensing, OnObject supports multiple degrees of bodily engagement from simple grabbing and releasing of an object to a series of motion gestures.

Self-contained Programming Flow

Following the Tag-Gesture-Response flow, novice end users including preschool children are able to program physical objects to respond to gestural triggers. Device-only programming allows users to maintain their focus on the object and their hand – the locus of the interaction – without being distracted by complex development tasks. The on-screen programming tool allows user gestures to trigger other audiovisual and Internet events than user-recorded sound, while the keystroke mapping tool extends the response further to various actions in most software applications available on the user's computer.

Gesture Server Architecture

The OnObject software architecture abstracts the raw device data to tag ID, gesture ID, and button state for a large number of custom and existing applications via direct or RESTful application programming interface (API). A fixed gesture palette recognized by machine learning algorithms and heuristics produces reliable results.

Table 1

Examples if novel applications and interaction techniques created with OnObject.

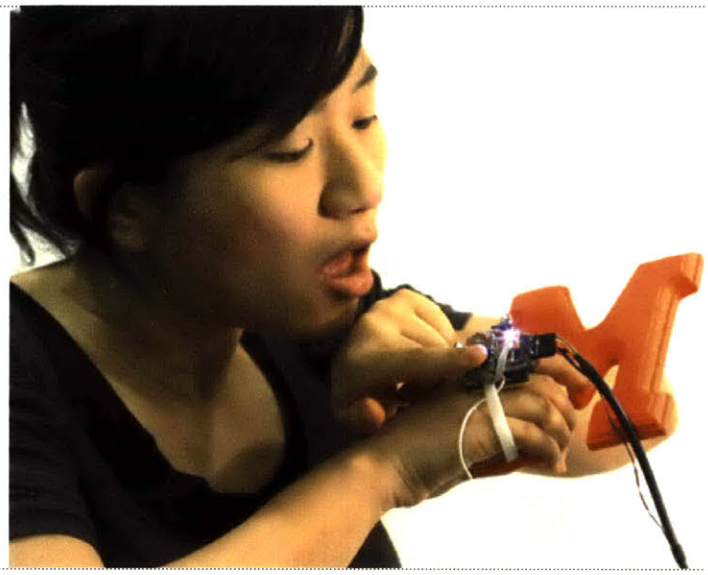
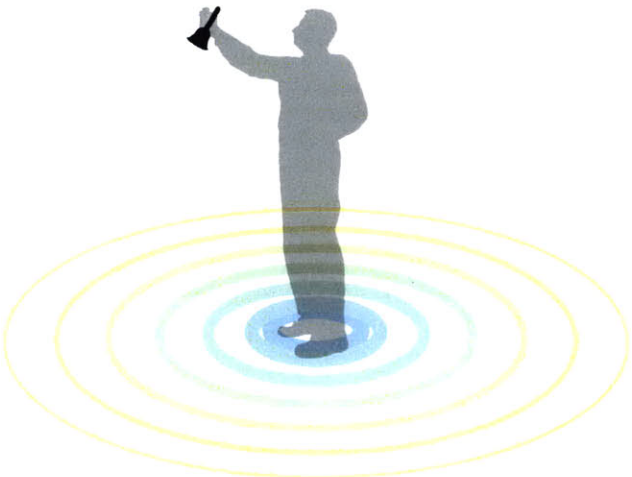
1	**Program sound for gestural triggers** Turn wood alphabet letters into talking educational device in less than 10 seconds.	
2	**Create interactive books on plain sheets of paper** Tags are attached to hand-drawn characters or scenes, where voice and narrations are recorded.	
3	**Easily create interactive installations** Using the on-screen tool, an artist can quickly create an installation where performer shakes a bell and a ripple video projection plays.	

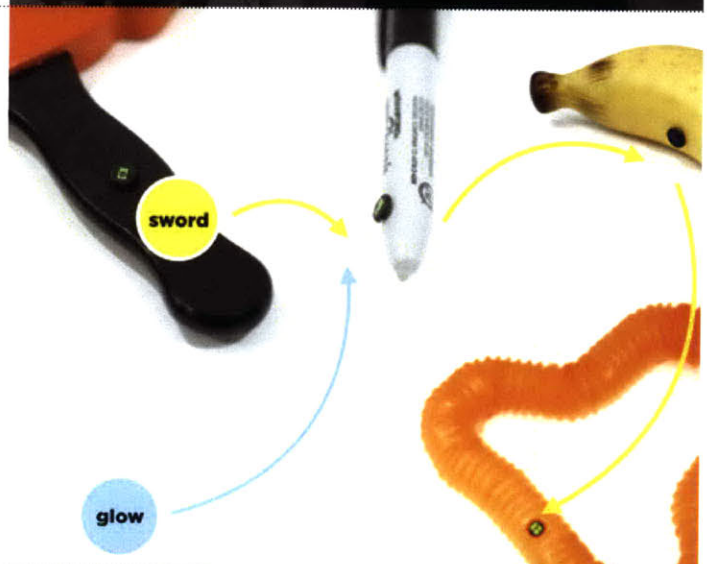
Table 1 continued

4 Play videogames with situated objects

Virtual swordfight with a physical Sharpie pen.

5 Copy-paste programming between objects

Program one object, propagate to another, then modify it as necessary.



6 Create interactive prototypes as you brainstorm

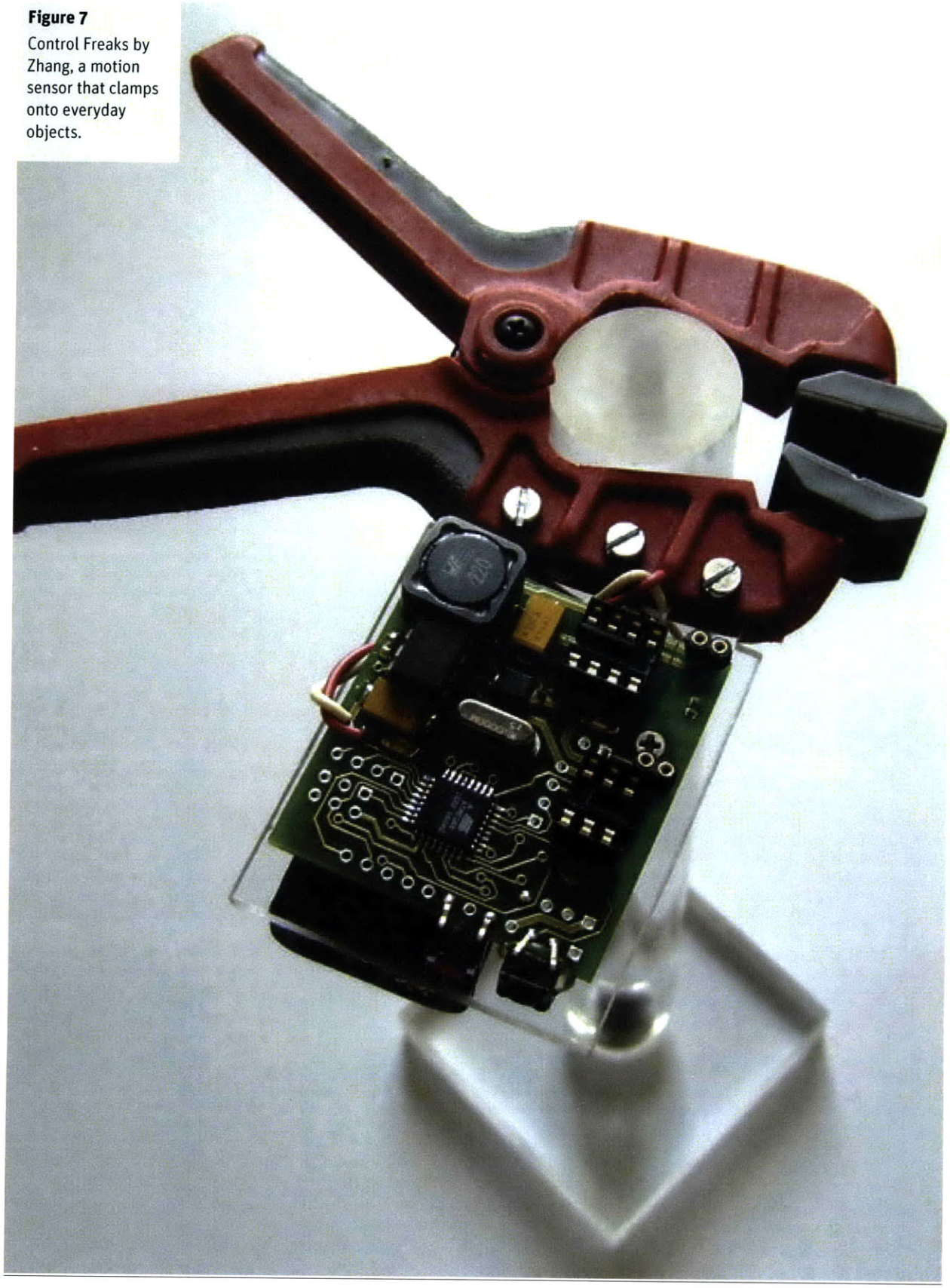
9 toy concepts were prototyped in clay and plastic by one user within an hour.

Chapter 2

Related Work

OnObject relates to multitude of previous research, yet differentiates itself by its scalability to situated objects and the instant gratification of TUI creation.

Figure 7
Control Freaks by Zhang, a motion sensor that clamps onto everyday objects.



2.1 Positioning and Differentiation

In this chapter, we examine the inspiration behind OnObject's states-oriented approach, then survey existing approaches in object and motion sensing, and conclude that OnObject merges wearable sensing and programming by demonstration for rapid creation of TUIs.

OnObject's architecture and implementation of gesture recognizer bear resemblance to activity recognition systems including Wockets/ MITes, ReachMedia, Mobile Sensing Platform (MSP), Kim's and Berlin's toolkits among others [31, 4, 45, 13, 2]. However, unlike activity recognition systems, OnObject is designed for interactivity and extemporaneous end user programming.

Compared to other sensor-based prototyping tools such as Examplar, Thumbtacks, and a CAPpella [6, 30, 3], OnObject offers significantly more abstracted representation of the gestural language and rapid in-situ object programming — its palette-based approach is usable by casual users, such as parents and children, without switching context for low-level activities like writing code, wiring, or watching raw input.

OnObject relates to the gestural programming by example pioneered by Curlybot, and the definition of Gesture Object Interfaces introduced with Picture This! [64, 23], while scaling the gestural interaction to any situated objects in our environment with the design principle of appropriation by attachment introduced by Zhang and Hartmann [26].

2.2 Multi-State Objects in Product Design

2.2.1 Collapsibles

While object-oriented thinking and state machines have been an instrumental concept in computer science, the concept of multi-state physical objects and products has long existed. According to design educator Per Mollerup, "collapsibles" are man-made smart objects such as umbrellas, eyeglasses, newspapers or rubber boats possess "two opposite states, one folded and passive, one or more unfolded and active." Ubiquitous in our environment, collapsibles are "man-made accommodations to change. [42]"

Mollerup's definition of multi-state objects inspired OnObject's state-driven approach to physical objects, where programmable states are added to a physical object in user's hand.

2.2.2 Interaction Design

Influenced by common use of software, today's interaction designers are also appropriating state model to design the interaction flow of electronics and software products, claiming that "designers should be able to design [products'] logical entities too. [43]"

OnObject extends this notion and aims to add, remove and modify logic to existing objects while allowing users to creatively leverage the physical properties like the shapes or materials and states of everyday objects.

Figure 8 Book cover of Mollerup's "Collapsible": Umbrella as an example.

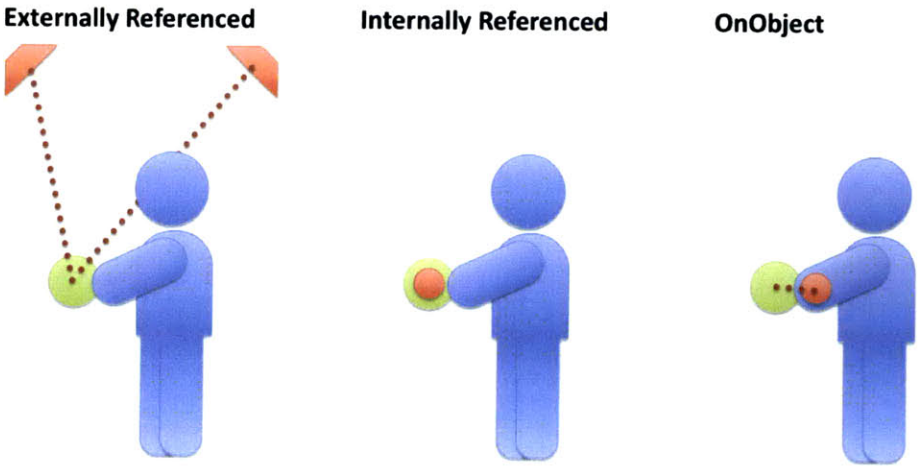


Figure 9 Comparison of sensing methods.

2.3 Motion Sensing Technologies

2.3.1 Verplaetse's Classification

In his 1996 article, Verplaetse stated "one of the current goals of technology is to redirect computation and communication capabilities from within the traditional computer and into everyday objects and devices – to make smart devices. One important function of smart devices is motion sensing. [29]"

In order to track body and object and their movements, many systems employ infrared, video, or other methods of computer vision. The G-Speak system used by Zigelbaum's g-stalt application uses a Vicon motion capture system to track passive IR retroreflective dots that are arranged in unique patterns on plastic tags placed on gloves or glued onto objects [37]. The Sensor Bar for the Nintendo Wii platform is required when the player is controlling up-down, left-right motion to point to menu options or objects on screen using the Wii Remote [46]. reacTable and Trackmate are two tabletop systems that track the identity and movement of objects placed on the top surface with cameras optically tracking fiducial markers attached on the objects [11, 36].

These tracking techniques fall into what Verplaetse classified as *externally-referenced* methods. Verplaetse asserts that these methods are physically removed from the moving object of interest, prone to occlusion and noise sources. In addition, these techniques require physical installation in the scale of a webcam-enclosed box in the case of Trackmate, to a room-sized motion capture system for G-Speak.

Internally-referenced sensing devices, on the other hand, "attaches directly to the moving body of interest and gives an output signal proportional to its own motion with respect to an inertial frame of reference. [29]" According to Verplaetse, two types of sensors comprise inertial sensing: accelerometers for sensing translational accelerations and gyroscopes for rotational rates.

2.3.2 Mixing Internally- and Externally-Referenced Sensing

The aforementioned Wii in fact takes a hybrid approach, utilizing both externally-referenced (infrared Sensor Bar for positioning) and internally-referenced (accelerometer in Wii Remote) sensing, with optional proprioceptive capability with add-on attachment devices (accelerometer in Nunchuck and gyroscope in MotionPlus). This mixed approach allows the Wii system to provide rich user experience without requiring high precision equipments that are costly and complicated to setup and use.

OnObject's motion sensing is internally referenced to the point of interaction — the user's hand — as it does not require external installation of sensing hardware and therefore can be used in a variety of environments, including mobile contexts. On the other hand, the object tracking is external to the objects, as the objects do not contain the sensing devices themselves.

The combination of RFID tags on objects and sensors on the user's body minimizes the burden of fabricating objects with embedded sensors, while the self-contained setup makes it applicable to a significantly wider variety of contexts than using externally-referenced methods.

2.4 RFID Tracking

2.4.1 RF Interaction for End Users

Various forms of RFID are widely used for public transit, authentication, and tracking of products, books, and livestock. Products like Touch-a-Tag and Violet Mir:ror [47, 48] utilize it as a way to achieve pervasive interactivity. Using RFID to recognize everyday objects for interactivity and entertainment is a commonly used technique in HCI. Recent examples include a bowl-like container that plays media content associated with the objects placed inside [40], and Sniff, a stuffed dog with an RFID reader hidden in the nose [28].

2.4.2 Engaging RF Interaction

With their Bowl interface, Martinussen et al. conducted the a series of tests with a 2-year-old child to find usability and engagement level of physical tokens made from situated objects [40]. Child was able to use and enjoy existing toys or new objects as tokens that were pre-mapped to movies with clear association: e.g., a Mickey Mouse-doll = a Mickey Mouse movie, but was confused with physical tokens used for navigation.

Interestingly, "odd" tokens with unclear association (e.g., a stick that plays a new movie every time child stirs it into the bowl) turned out to be very engaging and conversational. However, it was the "homemade" tokens mapped to content created by the child and her parent that proved to have the most potential: child enthusiastically presented the tokens and content to the audience.

OnObject research confirms that preschool children are able to use token objects with binary associations. Furthermore, a play session reveals that a 3-year-old is able to map

audio content (the recording of her voice) to a tagged object in less than 30 seconds, guided by a facilitator. Anecdotal observations from OnObject demonstration and user sessions also suggest users of "homemade" and self-mapped physical interfaces tend to present them with much enthusiasm and some theatrical quality.

Above all, OnObject's approach enables novice users and children to quickly create user-defined mapping that Martinussen et al. deemed the most engaging, without resorting to complex coding and wiring tasks.

2.4.3 Believable RF Interaction

RFID tag reading is typically used to create discrete, transactional user interactions. Swiping a cardkey to open a door is purely functional and transactional. However, some RF-enabled interfaces are designed to allow contextual meaning and believability: Sniff achieves more emotive user experience by integrating the reader functionality with the sniffing metaphor of a dog [28]. The unique form factor of reader-equipped nose is put to contextual interpretation by the child's action of carrying the dog and the gesture of holding it out to sniff objects of interests. musicBottles is another example that integrates RF sensing with the bottleneck form factor, where user gestures revealing and obscuring the tag are well matched with the embodied action of corking and uncorking the bottle [9].

Figure 10 Examples of believable RF interfaces: Sniff (left), musicBottles (right).

From these examples, design principles for more believable user interaction and higher semantic matching may be derived:

- Integration of form factors with the technical function. In the case of RF technology, map presence and absence of a tag to a contextually plausible states (e.g., open vs. closed bottle, sniffing vs. not sniffing through nose).
- Accompanying user gestures that are contextually meaningful. The actions user takes to transition between above states are contextually plausible and open to personalization (e.g., uncorking a bottle, holding the dog out toward a flower).
- Emotive response. The results produced by the motion in both cases are emotive, like dramatic music, haptic and sensory experience.

OnObject opens up this particular design space to the end users. It is up to the end user to determine where to place RFID tags, how to transition between tag presence and absence, and how to interpret the motion in their context. For instance, a user may decide that playing with a stuffed crocodile is more fun when it is grabbed by the snout, so she can map sound effects to the talking, singing or eating of the animal.

2.4.4 Beyond Associative Media

In his Papier-Mâché research, Klemmer classified TUIs into spatial, topological, associative and forms applications. In associative applications, "physical objects serve as an index or 'physical hyperlink' to digital media. [33] " The aforementioned Marble Answering Machine, Amphibian platform, Bowl, Sniff, and Mir:ror are examples of associative applications.

OnObject employs tagging as in most associative media. However, in addition to the tag-sensing gesture (grab and release), OnObject users can add more motion gestures while the tag is present or absent. Grabbing and opening the mouth of the crocodile can trigger one sound, but shaking and trembling the mouth can trigger another, such as threatening hissing sound.

2.5 Multimodal Sensing with Mobile and Wearable Devices

2.5.1 Inertial Proprioceptive Mobile Devices

Verplaetse explains that proprioceptive devices "have a sense of themselves, particularly a sense of their own motions. Embedded with inertial sensors, these devices are capable of autonomously sensing their own motions and orientations and reacting accordingly." [29]

Graspables take a unique approach in re-configuring the screen of a digital handheld device depending on how the user grabs it with her hands. Using capacitive sensors to detect where user grabs the device, and an accelerometer to determine the angle, it changes the screen from phone layout to a camera, PDA, or gamepad layout and vice versa. [49]

Spearheaded by the Apple iPhone, many mobile phones are now equipped with multiple sensors including accelerometers and an internal compass. The iPhone's sensors detect ambient light level, proximity from the device to user's body, and the orientation of the device, in addition high-fidelity multitouch input on its display [50].

While these devices are able to sense their own states and movements, their primary function is communication and information display, as opposed to OnObject's purpose of TUI creation and configuration of user's environment. These screen-centric devices also are intended to be the focus of the interaction and occupy both user's hand and foreground attention. The OnObject device, on the other hand, augments user's hand

during the interaction with physical objects in hand. The result of the interaction is contingent to the various objects user handles.

2.5.2 Wearable Sensors

In sensing modality, ReachMedia by Feldman et al. and daily activities detection kits developed by Kim et al. and Berlin et al., and Mobile Sensing Platform by Choudhury et al. bear similarity to OnObject for utilizing RFID and accelerometer to detect user actions [4, 13, 2, 45]. A ReachMedia user retrieves information services associated with an object at hand by navigating a hierarchical menu with motion gestures and listening to audio feedback. Kim's toolkit aims to classify activities of daily living using wireless triaxial accelerometers and a glove-embedded RFID reader. Similarly, Berlin et al. developed a prototype that senses user interaction data for several days, focusing on optimizing RFID antenna's reading range, data logging methods, and long-term deployments.

ReachMedia's hardware is based on MITes wireless sensor platform [67]. Its open source successor Wockets [31] uses decision tree classifier on featurized accelerometer data to recognize daily activities such as walking up the stairs or brushing teeth. OnObject's local gesture recognizer uses a feature set based on Wockets.

2.5.3 Multimodal Sensing as a Creative Platform

OnObject, however, differentiates itself from for its primary function: it serves as a platform for users to convert physical objects to gestural interface and configure them, while Kim's and Berlin's toolkits are designed to track days-long routine activity recognition, and ReachMedia focuses on retrieving personalized information from objects. OnObject suggests a novel use of the sensing modalities, for end users to attach interactivity to their surroundings and express themselves as well.

2.6 Gesture Object Interface

2.6.1 Vaucelle's Dolls

Tagged objects used in OnObject applications act as *Gesture Object Interfaces* based on Vaucelle's definition involving "gesture recognition during object manipulation [23]." Vaucelle's Picture This! system introduces camera- and accelerometer-equipped dolls children use as actors in the video narrative they construct. Children also use predefined shaking and tilting gestures with the dolls in hand to perform basic video recording and playback tasks. The dolls in this case are actors in the scene that are gesture-animated in user's hands, but are also used as controllers for movie creation tasks. Similarly, tagged objects like the folding fan, a lump of clay or stuffed animals co-opted in OnObject become Gesture Object Interfaces once a tag is applied.

2.6.2 Rapid Creation of Gesture Object

Creating Gesture Object Interfaces take working knowledge in hardware, programming and fabrication. Using OnObject, however, designers and casual users can quickly create them by tagging any number of objects. Additionally, users can conceive and create never-before-seen Gesture Object Interfaces from blocks or clay and put them in action, taking advantage of the iterative rapid prototyping process.

Figure 11 Picture This!: Dolls are equipped with accelerometer and camera.

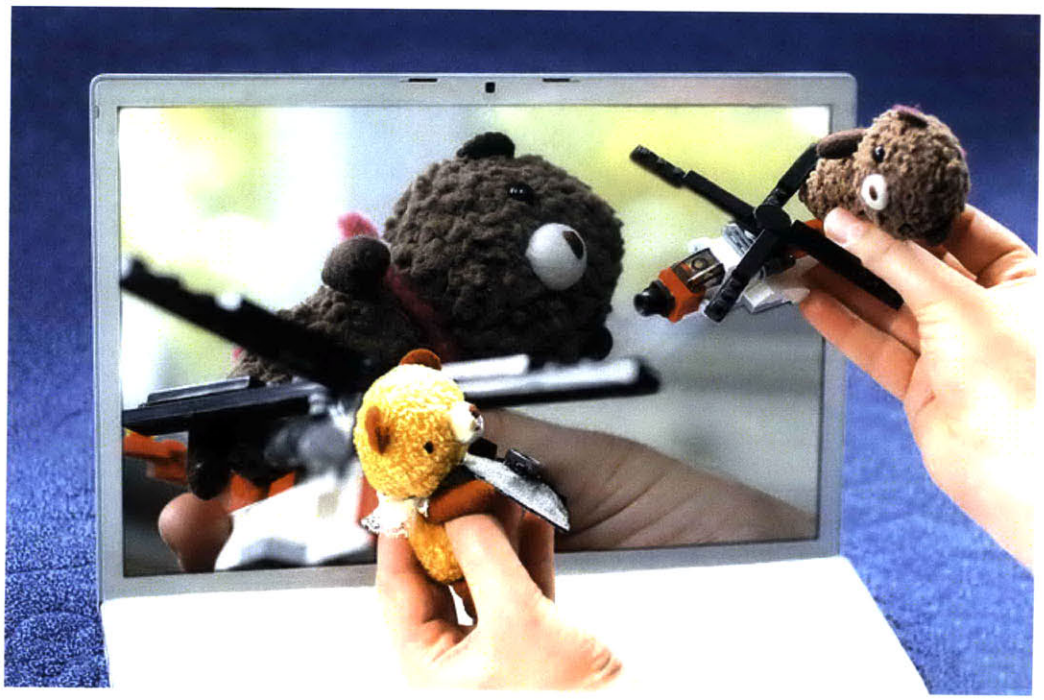


Figure 12 Picture This!: Using gesture commands, movies are recorded from the character's perspectives.

2.7 Form- and Interactivity-Giving Tools

2.7.1 Toolkits and Constructive Assemblies

Many toolkits have been developed in recent years for creating interactive artifacts for prototyping, education, and entertainment. These include d.tools, Arduino, and reacTable [7, 1, 11]. Some assembly tools attempt to give both form and interactivity. Two recent examples are Topobo, a 3D constructive assembly system with the ability to record and playback physical motion, and Senspectra, an augmented physical modeling tool for sensing and visualizing structural strain [19, 14].

2.7.1 Casting Interface from Anything

The primary differentiation and contributions of OnObject lie in the appropriation aspects of the system. Instead of providing a set of construction pieces, OnObject is founded on the belief that users are the resident expert of their own surroundings, similar to what Dey et al. argue in a CAPpella [3]. With OnObject, the user gains the ability to both quickly create motion sensing interface with any situated object, and to

create new forms of interactive artifacts by adding tags in sculpted or on assembled objects. Figure 13 shows how one can use existing objects such as a pen or coffee mug, or construct their own gesture object interface from blocks or clay.

Figure 13 Tagging an existing objects (top) vs. creating a makeshift gestural object from blocks (bottom).

2.8 Pervasive Tracking and Gaming

2.8.1 Appropriation through Attachment

OnObject's primary function of attaching interactivity to existing physical objects shares inspiration with Control Freaks, a clamp-shaped motion sensing device that clamps to everyday objects to turn them into game controllers (Figure 7), and its integration with Exemplar to design custom gestures [26]. OnObject adds two contributions to this line of "appropriation through attachment" approach:

- The ability to use the identity of the "host object," the physical object where sensor is attached, to produce different outcome for the same motion input. In OnObject's case, the tag is not sensor in itself, but a physical link to the sensor in user's hand.
- The ability for end users and developers to determine the outcome and design their own game using the device, on-screen tool or configuration text file.

2.8.2 Augmented Objects

Amagatana + Fula features a sensor-augmented umbrella that user can handle like a sword for dramatic audio output [12]. While its focus of adding interactivity to a mundane object relates to OnObject, Amagatana is rather similar to existing TUI objects like I/O Brush as the umbrella is significantly modified to house the sensing device. With OnObject, a game designer can sense the swinging of an umbrella by simply attaching a small tag to an existing umbrella without adding a device or battery, leaving the umbrella still usable for its original purpose.

2.8.3 Motion Tracking for Augmented Environments

Other projects have explored motion tracking assuming an intelligent environment. Both XWand and Wiimote are wireless UI devices that enable styles of natural interaction in intelligent environments equipped with sensing capabilities [25, 46].

OnObject is a programming platform to transform mundane environments to interactive ones. Instead of using an arbitrary controller object, OnObject facilitates using real world objects as gestural interfaces. While it does not provide location tracking, OnObject offers a low-overhead, low-commitment method to appropriate situated objects for interactive applications one by one.

Compared to G-Speak or Microsoft's Project Natal that track empty-handed gestures, OnObject takes a more structured, grammatical approach. Application events are contingent on the semantics of the user actions that consist of the physical artifact user is handling (the object) and user's motion (the verb). As the sensing is located at the point of contact, users are led to focus on the embodied manipulation of objects and physical engagement, instead of worrying about whether they are "seen" correctly by the external machine (Figure 14).

Figure 14 G-Speak's locus of interaction is external to the body space, where user stays aware of their visibility to the machine vision (left); OnObject users are led to focus on the contact point between their body and the physical environment – their hand (right).

2.9 Programming by Demonstration

2.9.1 Mapping by Demonstration

Hartmann et al. point out programming by demonstration in ubiquitous computing has been explored in many projects including a CAPpella and Exemplar [3,6]. Both projects specifically feature pattern recognition by demonstration, to aid users in building a custom pattern recognizer. Unfortunately, building the recognizer is only a small part of the programming and development process, and users are still exposed to many low-level tasks.

In OnObject's case, demonstrations are used throughout the programming process, for selection of particular gesture triggers during both programming and usage. First, the user literally maps her augmented hand to a particular object by grabbing it. Second, as OnObject provides a recognizer with a predefined set of gestures, Media I/O users can configure the mapping by demonstrating the trigger gesture of their choice among the provided set.

2.9.2 Rapid Creation of Applications

Comparing a CAPpella, Exemplar, and OnObject in terms of required user engagement and skills in the workflow reveals the different focus each tool has (Table 2). As a prototyping tool, OnObject focuses on easy application creation by novice end users

given a set of gesture vocabulary, rather than converting flexible set of sensor input into custom input vocabulary. As a CAPpella and Exemplar users choose and setup the input sensing devices, create custom recognizers and then develop hardware or software applications, these tools invite users who wish to be engaged in the end-to-end design and prototyping of sensor-based interaction. In comparison, OnObject caters to a wider range of casual end users who want to quickly create mapping between a combination and series of pre-defined events on a wide set of objects, test and modify the in minutes.

Table 2
Comparison inspired by Hartmann's "Designing sensor-based interactions" [6]

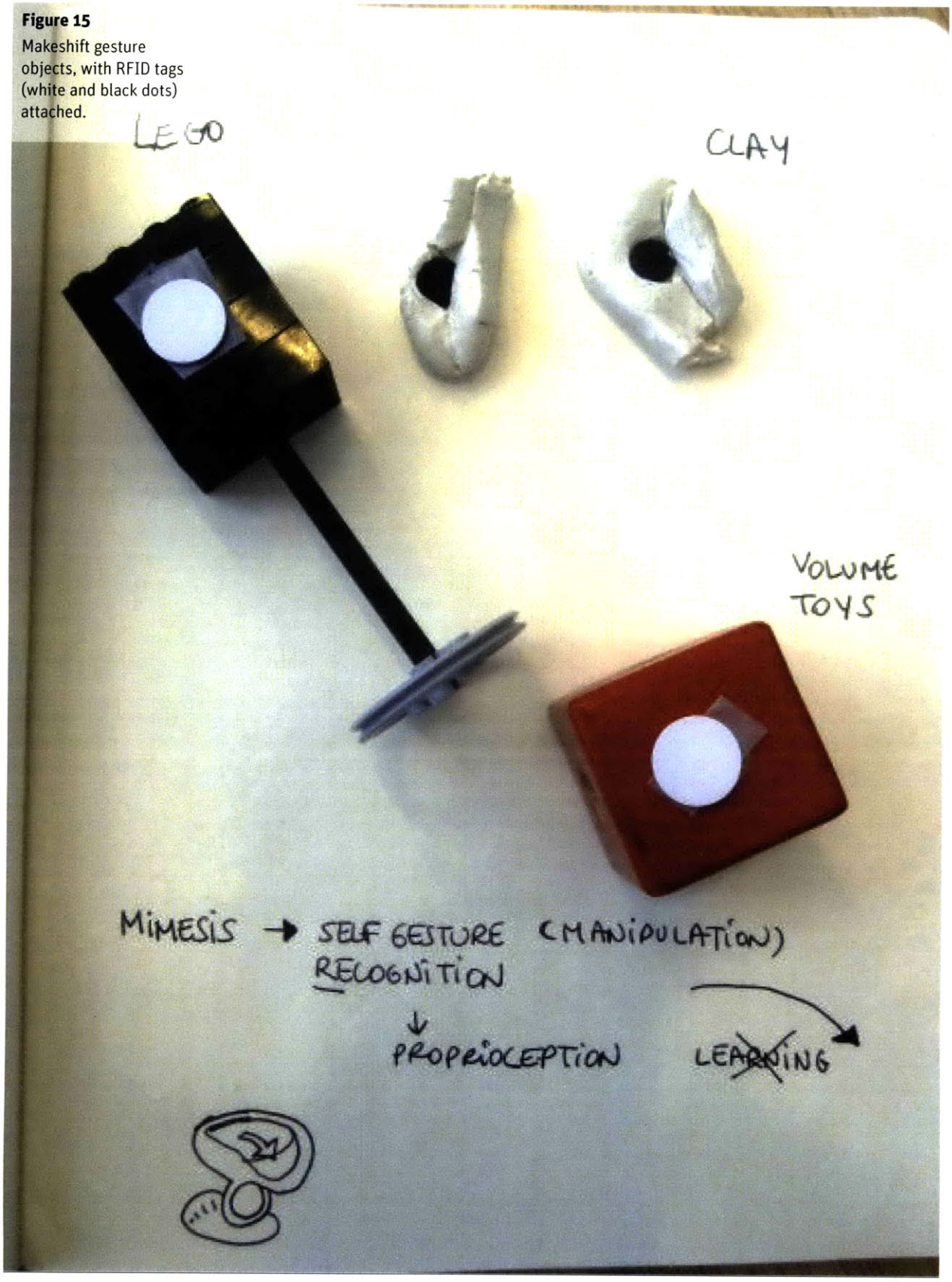
	a CAPpella	**Exemplar**	**OnObject**
TYPICAL USER	End user: Meeting-attending information worker	Product and interaction designer, hobbyist	**Casual end user: Parents and preschool children**
USER GOAL	Control my environment and devices based on my activities	Iteratively design sensor-based interaction	**Turn situated objects into gestural audio interface**
ASSUMED USER KNOWLEDGE	GUI movie editing, instrumenting a phone	JAVA, Phidgets or Arduino	**None, basic GUI to map additional responses**
INPUT MODALITY	User modifies phone, lighting devices and computer software and connects them to a CAPpella on the computer	User creates input devices and connects it to Exemplar running on the computer	**Attach a tag to an object, wear device, connect the device to the computer**
INPUPT PATTERN RECOGNITION	User creates recognizer from provided GUI	User creates pattern recognizer using provided GUI	**Gesture recognizer is built-in**
MAPPING BETWEEN THE PATTERN AND OUTPUT RESPONSE	User annotates input pattern and output pattern on provided GUI	User maps to OS-wide keystroke or mouse events using provided GUI	**User records sound response into device after performing gesture trigger**
APPLICATIONS	Actions on devices and hardware connected to a CAPpella	User creates output hardware/software application	**User specifies with provided GUI**
TRAINING THE RECOGNIZER	Days to weeks after the annotation	Set of train samples are recorded by users prior to mapping	**Recognizer is built in advance with training samples and provided**
RECOGNITION ACCURACY	50-93%	Unknown	**90-100%**

Chapter 3

System Design and User Experience

Combining the mobility of a hand-worn device, the reliability of a fixed gesture palette, the scalability of electronic tagging, and the clarity of the Tag-Gesture-Response flow, OnObject is able to provide a simple yet engaging method for end users to turn situated physical objects into a gesture interface.

Figure 15
Makeshift gesture objects, with RFID tags (white and black dots) attached.



3.1 Form Factors and Affordances

3.1.1 Tags

The RFID tags used for OnObject come in various sizes and shapes, 9-20+mm in diameter, 0.8-2mm thick, and in various colors including white, black and transparent.

Due to their small and slim form factor, the tags allow for many novel uses:

- **Multiple tags on one object:** Each part of the object can be programmed separately to behave differently. (Figure 17-1)
- **Augmenting transparent and fragile objects:** Even glass objects can be augmented with no modification and minimal visual interference. (Figure 17-2)
- **Objects created with tags:** Moldable objects can be created with tags embedded (Figure 17-3)
- **Body as a gestural interface:** Washable tags can be embedded into garments to control digital media with bodily contacts (Figure 17-4).

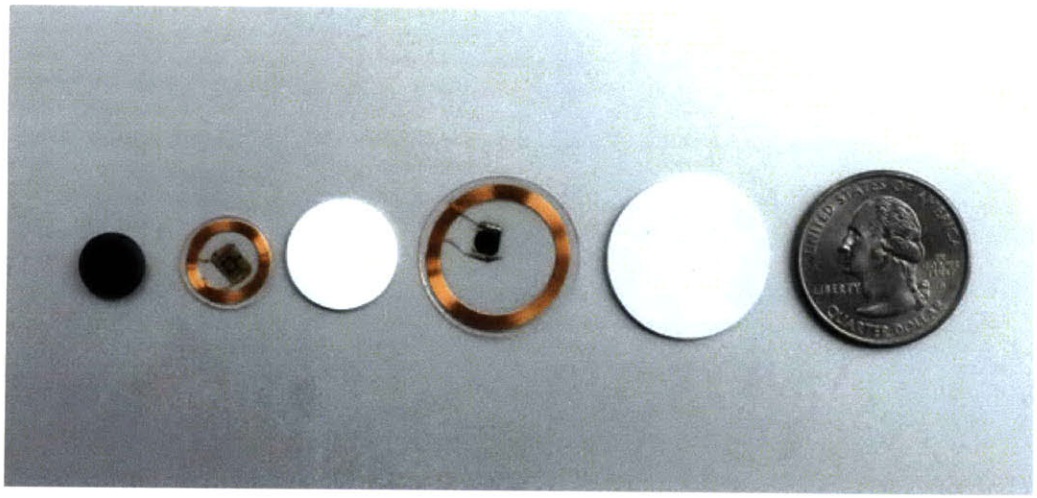


Figure 16 Variety of OnObject-compatible RFID tags, compared to a U.S. quarter coin.

Figure 17
Novel uses of small and slim tags.

3.1.2 Bare Device with Off-the-shelf Components

Most applications presented in this research use hand-worn devices approximately 40 x 40 x 25mm in dimensions. The RFID sensing is done through the reader's antenna located on user's palm, while the rest of the device sits on top of the user's hand.

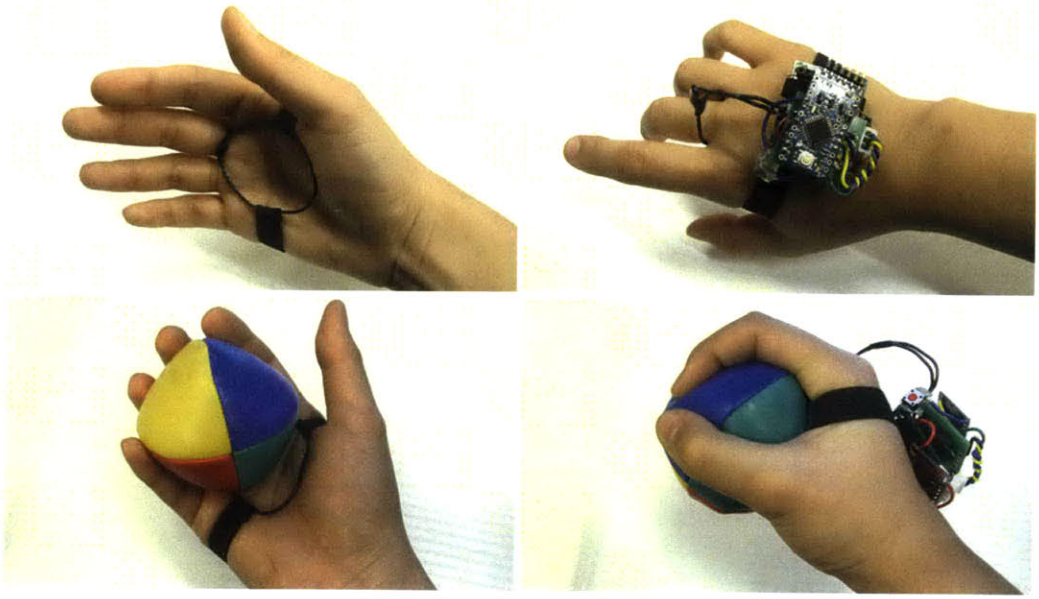


Figure 18 Device form factor.

Some of the devices use an internal antenna embedded in the RFID reader; others employ a custom-designed external antenna with 38-44mm diameter, cast in a 52mm-diameter silicone disc. When tags come within the antenna area, the 16-byte tag ID is detected by the reader.

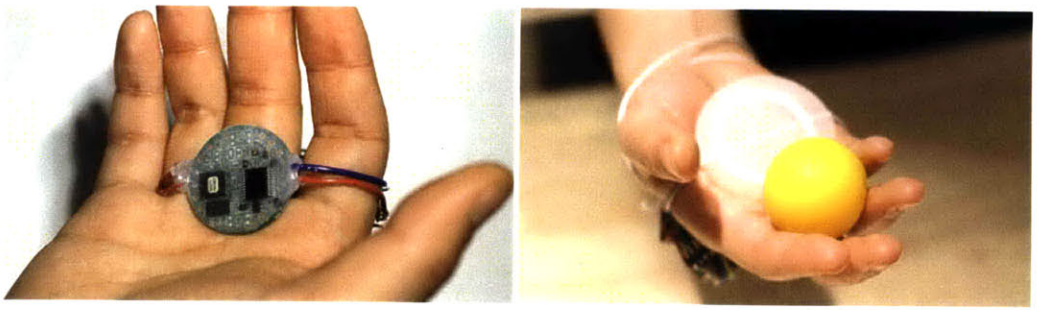


Figure 19 Internal antenna (left), external antenna (right).

The devices are composed of an RFID reader, a tri-axis accelerometer, an Arduino microcontroller board, three indicator lights, and a pushbutton. A microphone was added to enable sound recording for response.

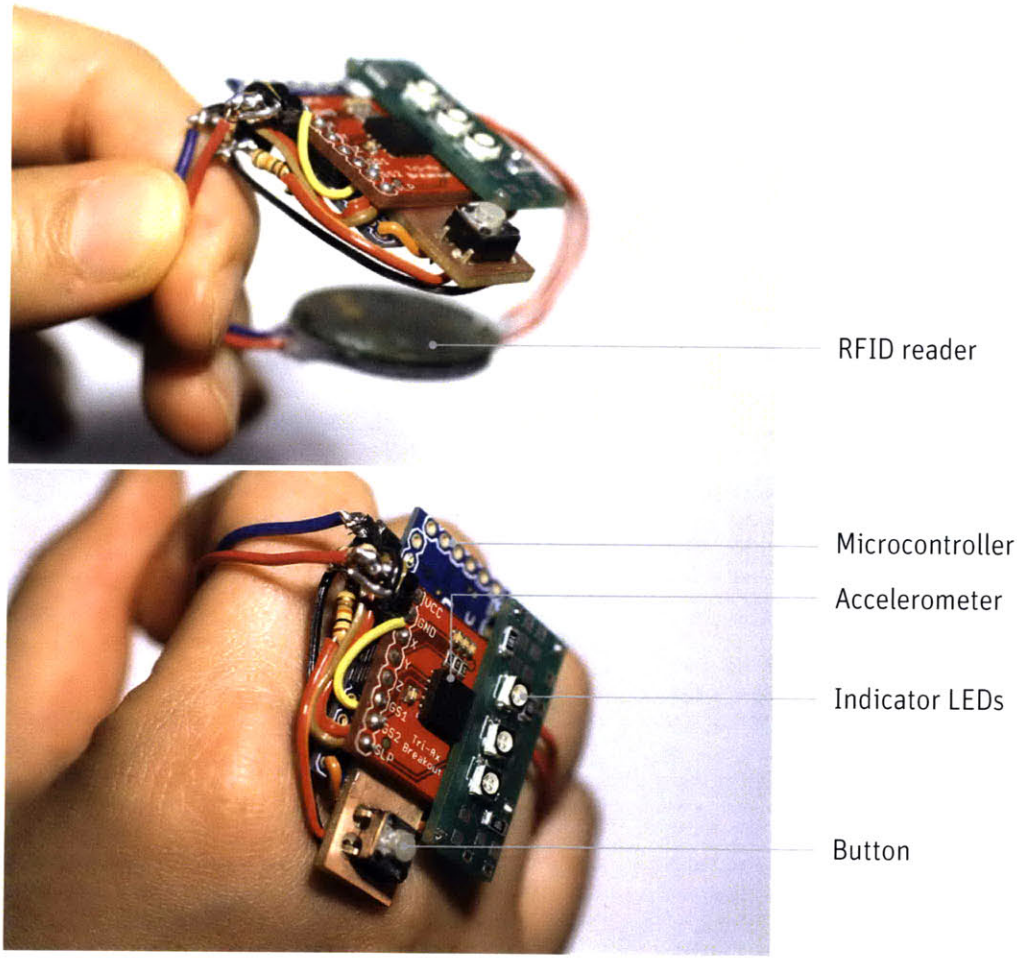


Figure 20 Anatomy of an OnObject device.

3.1.3 Communication with a Nearby Computer

Most devices use serial-to-usb cable to receive power from the compter and transfer data to and from the computer. A wireless version of the device transfers the data to and from the computer via Bluetooth serial port. The wired version has been mostly used due to its reliability and convenient power management.

3.1.4 Ring with Custom PCBs

We have been avoiding the glove form factor because gloves are strongly associated with immersive tracking and virtual reality applications, and completely enclosing user's hand may contradict the low-commitment, exploratory appropriation of graspable objects. In fact, the components of the device can be further compacted into a set of printed circuit boards (PCBs) in an oversized ring form (Figure 21). The PCBs and overall shape of the ring device have been designed, and has yet to be assembled and programmed with the firmware.

Figure 21 Prototype of a ring-shaped device with custom PCBs.

3.1.5 Garment-embedded

The current device components can also be rearranged and integrated with wearable garments. For example, a bathrobe or tracksuit can be equipped with an OnObject device so that the wearer can record sound into different parts of her body, and play sound effects, music, even her own voice when she massages her body or exercises (Figure 22).

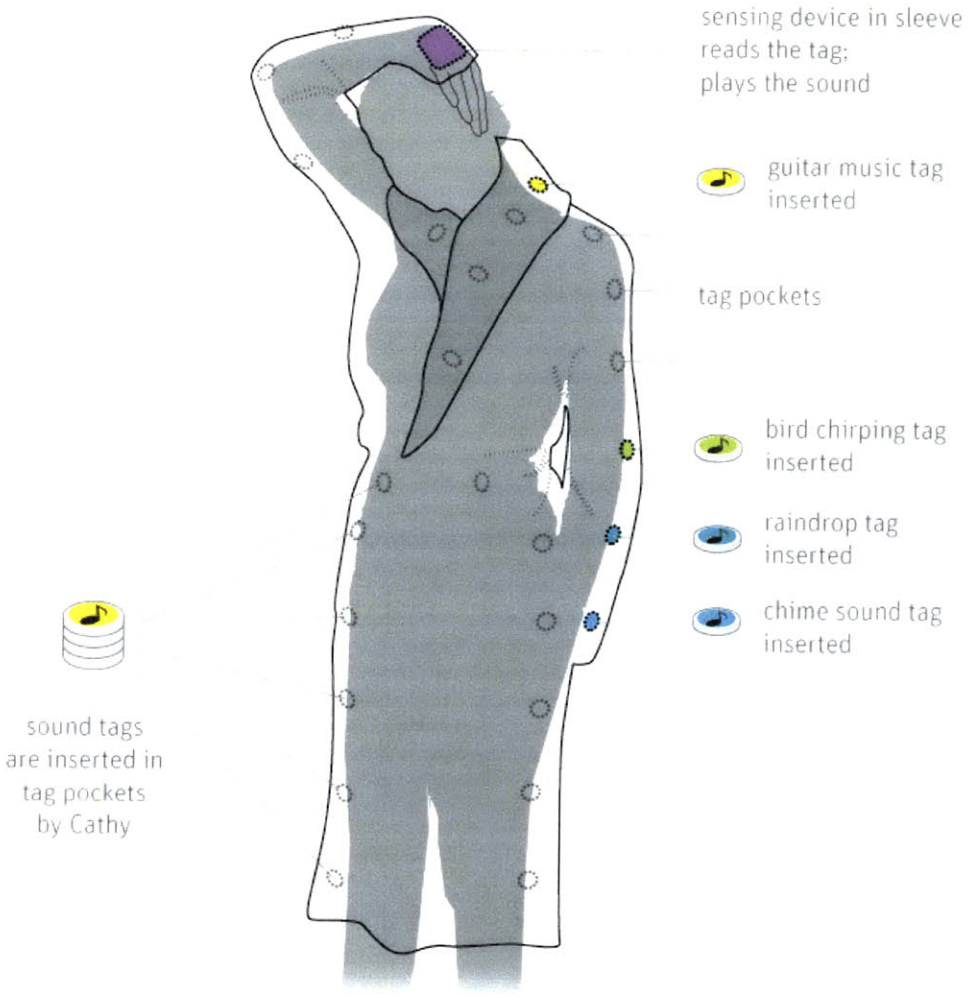


Figure 22 Garment-embedded application in development.

3.1.6

Controls and Feedback

Indicator LEDs

Currently three LEDs are located on the top side of the device to indicate when a tag is detected (top, Figure 24-2), button is pressed (middle, Figure 24-3), and a gesture segment is detected (bottom). With motion gestures where the hand is moving fast, users can see a bright trace of their motion as shown in Figure 23.

Button

The device is equipped with a pushbutton used for sound recording. Currently the same button is used to trigger copying and pasting in some applications, and the conflict can be resolved by either adding another control (another button for copy-paste) or adding another type of button press event (double-press, long and short press).

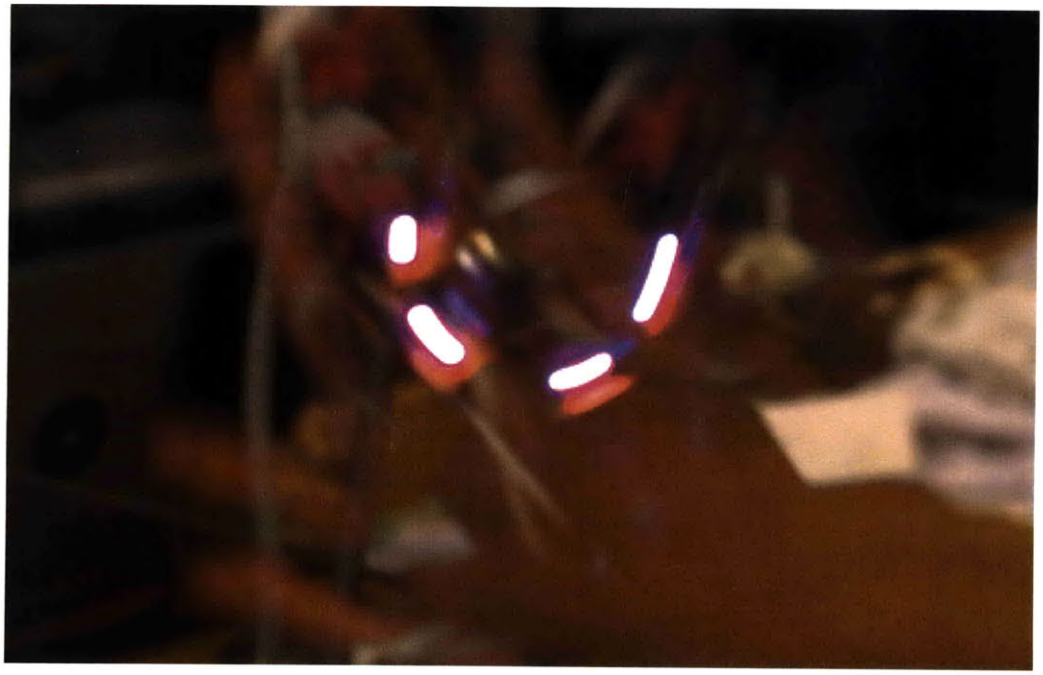


Figure 23 Persistence of vision effect with the bottom LED when user makes a circling motion.

Figure 24-1
LEDs are not on when idle.

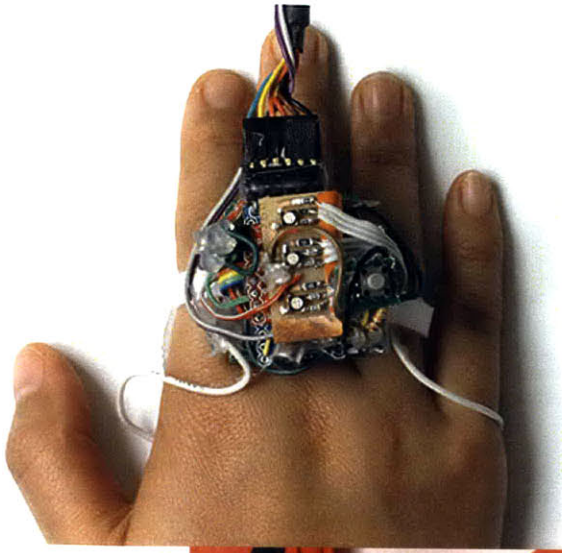


Figure 24-2
Top LED is on when a tag is detected.

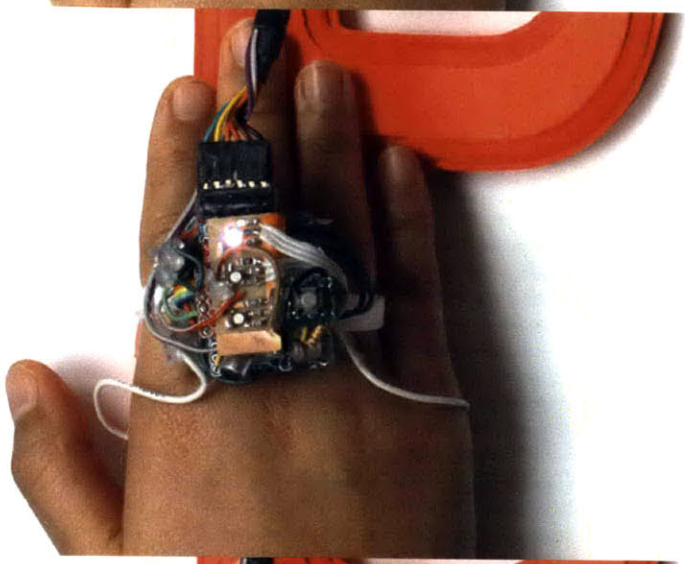
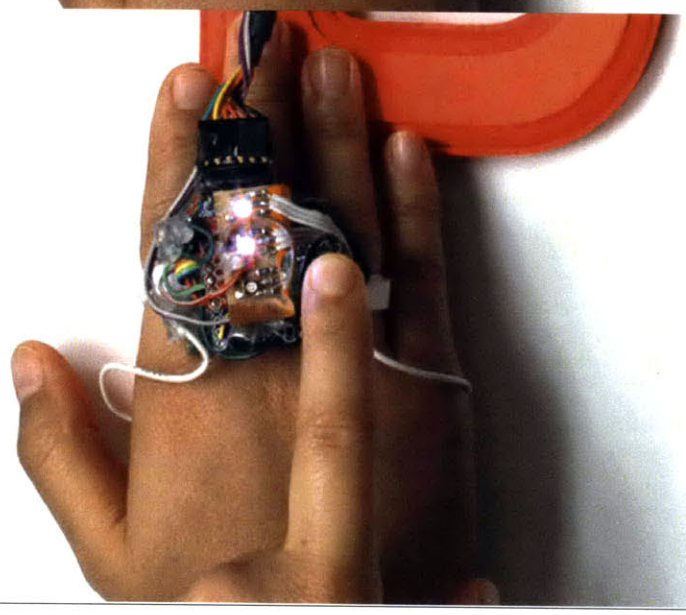


Figure 24-3
Middle LED is on when user presses the button.



3.2 Out-of-Box Experience

As an entry point to the user experience, the OnObject platform provides an "out-of-box" experience of interacting with a tagged object without writing a single line of code or performing a single mouse click.

For example, consider the water glass shown in Figure 25. It is particularly difficult to embed sensors in a transparent object like this glass. However with OnObject, user can add interactivity to this glass immediately by simply attaching a tag to the glass and then grabbing it. When the reader recognizes the tag, a subtle chime plays.

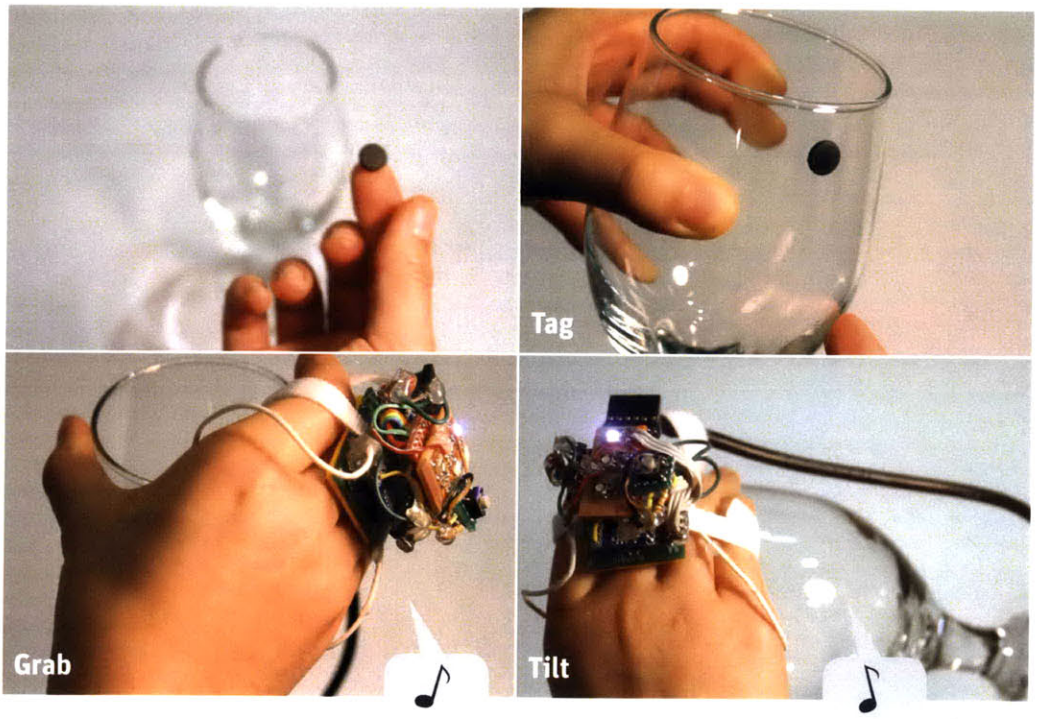


Figure 25 First user experience.

For each new tag, OnObject plays the default sound response: chime when grabbed, and boink when shaken. System-provided default values constitute a shortcut, and can help novice users learn the system [16]. This design helps users verify the system is functioning and enjoy immediate interactivity without a laborious setup or learning process.

3.3 Tag-Gesture-Response Flow

OnObject's goal is to make gesture and application design as easy as tagging an object, making a gesture, and specifying a response. Upon learning the premise of the system through the out-of-box experience, the user can start configuring each tag, following the Tag-Gesture-Response (TGR) flow.

3.3.1 Programming Sequence

Tag

An application can consist of many tagged objects, and a user can add a new tagged object to the application at any time by attaching an RFID tag to a physical object.

Gesture

With object grabbed by the tag, the user demonstrates one of the trigger gestures and default sound feedback plays. By demonstrating the gesture, user can remember how it feels for later repetition.

Response

The user holds the button down, records sound response to the gesture by speaking into the microphone, and releases the button.

When the user performs the gesture from then on, the recorded sound plays. The mapping between the tag, gesture, and response is application specific — the same grab gestures on the tag can trigger different responses depending on the application.

See Section 5.3.1 for a typical example of this flow.

3.3.2 Simplifying Assumptions

As with any intelligent interface, Tag-Gesture-Response is a tradeoff between system flexibility, recognition accuracy, and burden on the application designer, who is also the end user in the case of OnObject. We aim to minimize the burden on the user without sacrificing system performance by making simplifying assumptions concerning the system's expressiveness.

Fixed Gesture Palette

First, we assume a fixed palette of gestures. Although OnObject's underlying recognition engine learns from a set of training examples, adding training to an end-user application introduces complexity to the system: to the user interface, to the user's mental model, and to actually coaxing the system into constructing a highly accurate, usable recognizer. For instance, a CAPpella, a system for creating user-defined pattern recognizers, reports 50-93% accuracy, while uWave achieved 93.5% accuracy on a pre-defined gesture set with user-dependent accelerometer input [3,15].

OnObject's bundled gesture set includes grabbing and releasing a tag, shake, tilt, circle, swing, thrust, fan, and idle non-gesture motion or "background noise" (Fig 26). Separation and joining of two tags can also be added to the set depending on the firmware of the RFID reader. The gestures were selected based on their broad applicability (shake, tilt, grab) and desired applications (swing, thrust).

Because we collected a training set with 100 or more samples per gesture, the recognizer achieves 94-97% accuracy. While the gesture palette can be replaced, modified, or extended by developers to adapt it to a new application domain, it is important to the OnObject design, given the state of the art in gesture recognition, that the end user is given a fixed palette.

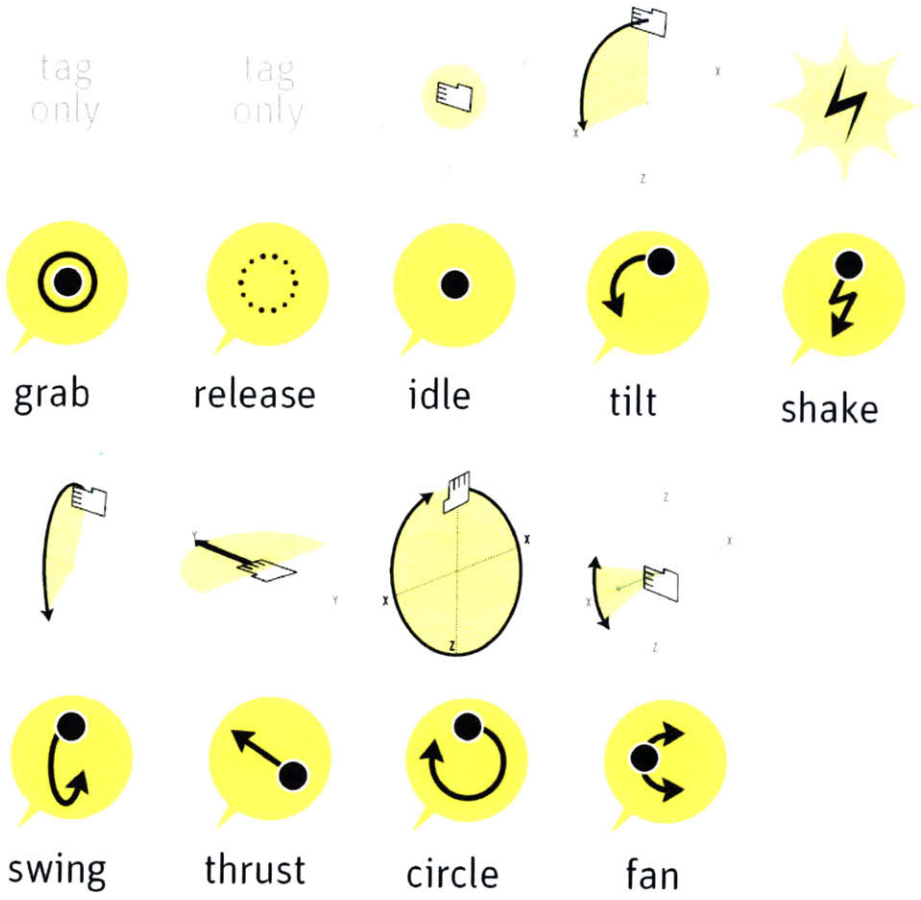


Figure 26 Current gesture palette.

Tag Extensibility

While a fixed gesture palette affords high accuracy recognition, it limits the number of actions that can be performed by a single object. However, programming physical objects does not require significant modification other than attaching an RFID tag via glue, tape, or clips. This makes the number objects in the system arbitrarily extensible, allows for experimental appropriation of our physical surroundings, and makes the system widely applicable despite a fixed gesture set.

Interpreted Gestures

Moreover, by adding new tags either on separate objects or on different parts of the same object, users can contextualize the same gesture in multiple ways. The shake

gesture for example, can be perceived as hopping when performed on a stuffed kangaroo, crashing of a toy car, or clapping on the opposite hand. Tilt can mean closing of a book when performed on the back cover, or looking for a bookmark when applied to the spine of the same book.

Constrained Recognition

In fact, the tag modality can be a powerful way to complement the recognizer. By constraining the set of gesture alternatives for any given tag, we can perform an easier recognition problem, which we call constrained recognition, and which can signfiicantly reduce the recognition errors depending on the configuration. For example, when there is only one active gesture, constrained recognition reduces the N-class recognition problem to a 2-class (idle vs. active gesture) problem.

Contingent Accuracy

The final assumption that helps the Tag-Gesture-Response flow is the notion of contingent accuracy. The primary user-centric criteria for gesture design are the application behavior and the physical affordances of the tagged object. However, as recognition accuracy decreases, the user experience degrades accordingly. Contingent accuracy allows end-users to easily incorporate recognition accuracy into their gesture design without having to understand the details of the underlying recognition technology.

Because the system contains a fixed set of gestures, but only a subset of these gestures will be active for any given tag, it is possible to precompute the expected recognizer accuracy with any active subset of gestures. Using these precomputed values, we can visualize the impact of the user adding a gesture to a tag before it has been added, and integrate this visualization into the programming flow.

This is illustrated above in the Media I/O scenario (Figure 27-2), in which gestures are color-coded according to contingent accuracy. Gestures with high contingent accuracy given the currently active gesture set of a tag are shown to be "easy" and highlighted in green. Gestures with medium and low contingent accuracy are shown to be "medium" and "hard" and highlighted in yellow and orange, respectively. The visualization is simple for users to understand and react to. For example, a user might employ a tag extensibility to exploit the accuracy benefits of constrained recognition if the current tag is becoming too crowded with gestures.

3.4 Creating OnObject Applications

OnObject provides three ways to support end user programming of physical objects:

- **Device only:** Screen-free mapping of gestural trigger and audio feedback by recording sound directly into the device, often user's voice. Created for novice users including preschool children to achieve simple programming within 30 seconds.
- **Media I/O:** On-screen tool to map gestural trigger to audio, visual, video, and Web browser events with more playback options. Designed for common computer users to create media playback applications within 5 minutes.
- **KeyMapper:** Configuration text file user edits to control other programs on the computer that takes keyboard events. Created for artists, prototypers and hobbyists to create customized gestural applications that work with a wide variety of existing software (PowerPoint, TextEdit, iTunes, etc.) or user-developed software (developed with Flash, Processing, MaxMSP, Python, etc.)

3.4.1 Device Only

For a primary set applications, the programming is done using only the hand-worn device equipped with a button and a microphone in the following series of steps:

Tag: User attaches an RFID tag to a physical object.

Gesture: With object grabbed by the tag, user demonstrates one of the trigger gestures; default sound feedback plays.

Response: User holds the button down, records sound response to the gesture by speaking into the microphone, and releases the button.

Play: When user performs the gesture from then on, the recorded sound plays.

3.4.2

Media I/O

For other applications, the programming is done using the device and Media I/O, an on-screen GUI tool. To make a tilt gesture on a glass trigger a water sound to play, user can take the following steps:

Initiate Programming: The user attaches a tag and grabs the glass. An LED on the hardware device lights up, and a chime sound plays to verify that the tag's been registered. The new tag appears on screen, with the default trigger gesture ("grab") and response ("chime").

Change Gesture: When the user presses the button on the device, a list of gesture triggers are displayed, color-coded by contingent accuracy as described in Section 3.3.2 (Fig 27-2). This serves as a guide to help the user choose appropriate gestures, in addition to the physical affordances of the object itself. When user demonstrates the "tilt" gesture, the gesture trigger is updated.

Change Response: At this point, the user can release the glass and use the mouse. She picks the type of output (sound, image, video, or webpage) and a particular instance ("water") pre-included in Media I/O (Fig 27-4). She picks whether to trigger the response on start or end of a gesture, and whether or not to repeat the response with long-running gestures.

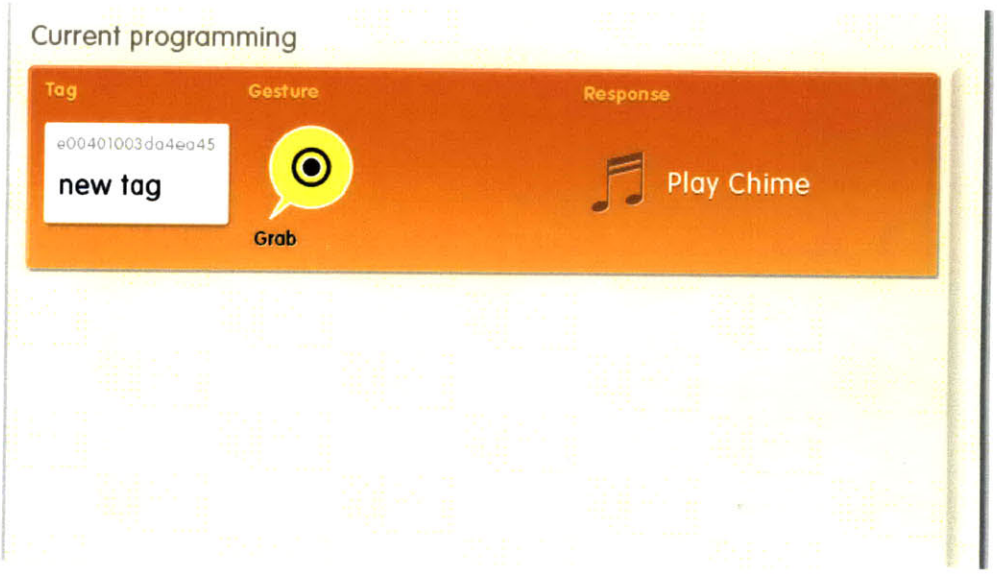


Figure 27-1
Media I/O screen when user first grabs a new tagged object.

Figure 27-2
User changes gesture trigger by demonstration.

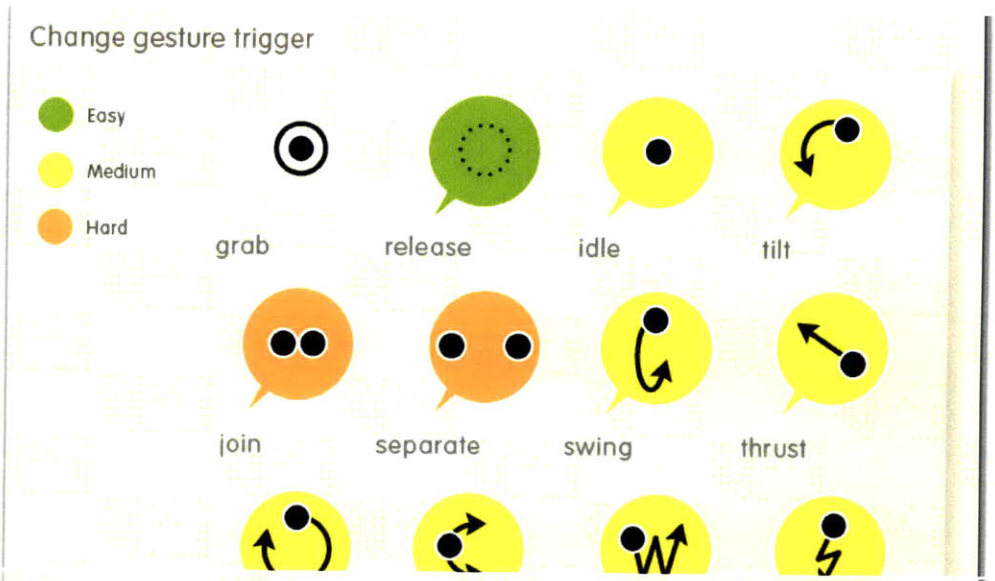
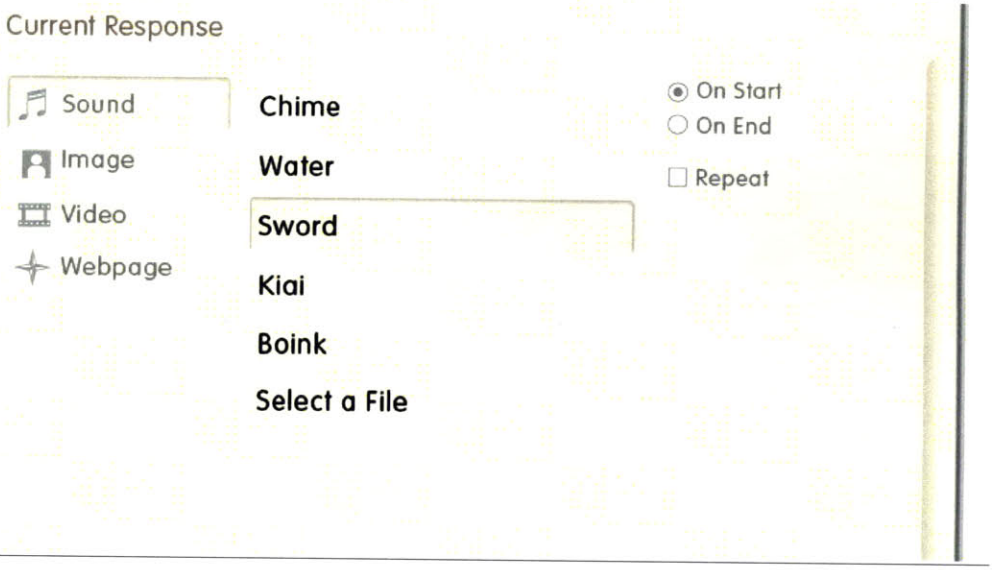


Figure 27-3
Media I/O screen with a new gesture trigger.

Figure 27-4
Response is changed on screen.



3.4.3 KeyMapper

Alternatively, the above programming could be done by editing a configuration text file where each tag, gesture ID is mapped to an operating system-wide keystroke events, so that a large number of software applications can interact with physical objects.

For instance, to play dramatic swordplay sounds wih a folding fan, user would tag the fan, and edit a configuration text file that lists each tag, gestures, and resulting keystroke to be sent system-wide. An Adobe Flash file listens to keystroke events and plays appropriate sound or movie files, while an underlying Python program takes care of serial connection, gesture recognition, and send keystroke events.

The configuration text file may look like the following:

```
# fan tail tag
[e00401003da52d99]
grab:g
thrust:t
swing:s

# no tag: empty-handed gestures
[0000000000000000]
thrust:t
swing:s
special:p
```

The Flash Actionscript then takes each keystroke event and plays appropriate audiovisual elements. Using KeyMapper, existing TUIs like Amagatana + Fula can be quickly prototyped. See Section 5.1.2. for a partial reconstruction of the Amagatana application.

3.4.4 Advanced API

Specialty applications like video game titles can be developed to interface with OnObject and take gesture events on everyday objects. For example, swinging a Sharpie pen can be taken as a sword swing, and shaking it can be interpreted by the game as charging the weapon; when user grabs her other arm by a tag, the game could be frozen. In case of these specialized applications or solutions, developers can use the following information to develop applications and games that incorporate situated objects and body gestures:

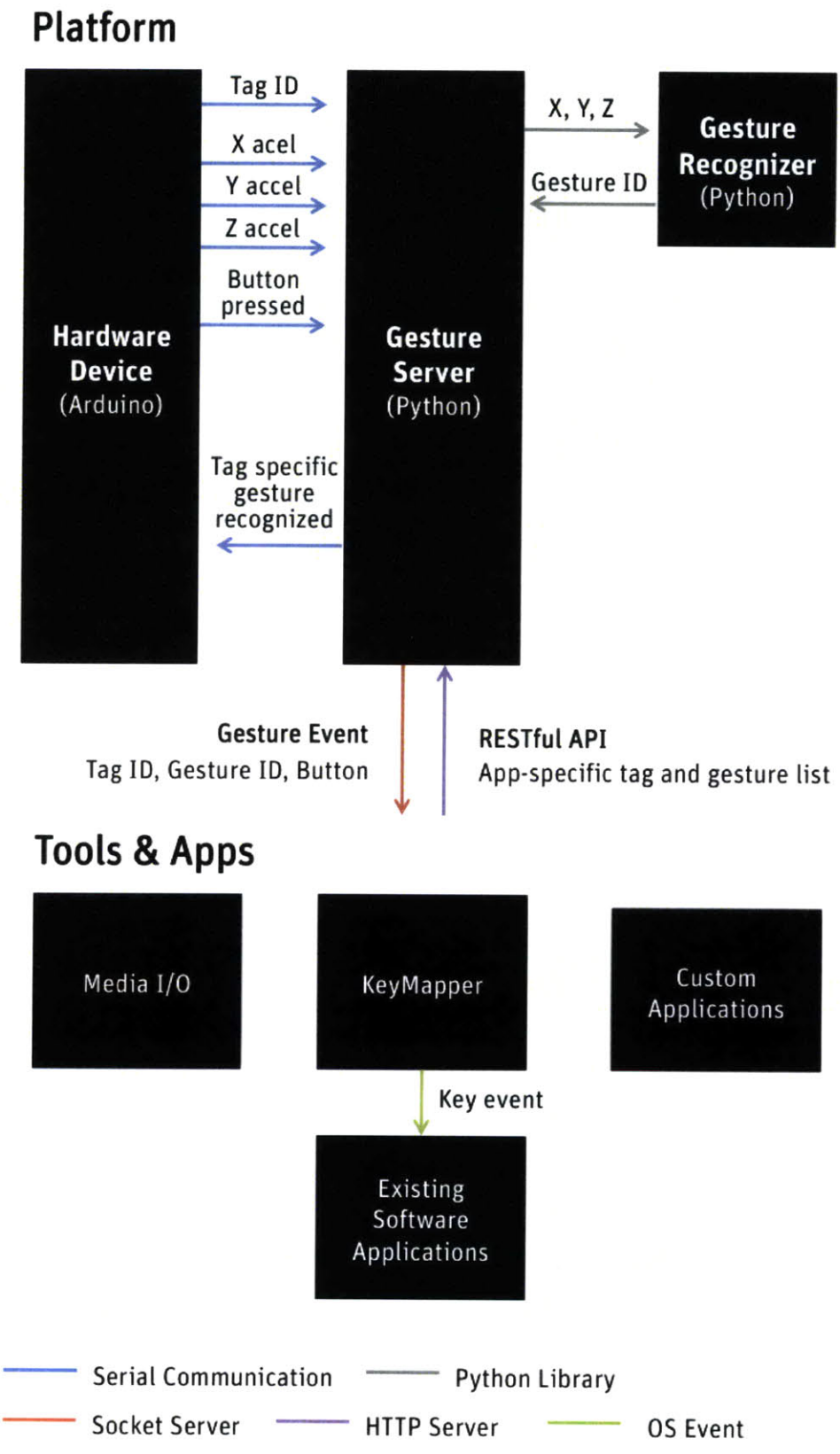
- Current and previous tag IDs
- Current and gesture IDs, local (segmented) and global (statistically adjusted)
- Number of consecutive tag IDs
- Number of consecutive gesture IDs
- Button state
- Tag presence during a gesture
- Elapsed time during a tag presence
- Elapsed time between gestures

Chapter 4

Implementation

OnObject's implementation is as a loosely-coupled service which provides streaming gesture events to applications, and a RESTful API to query and set configuration state. The platform consists of the hardware device and gesture recognizer, which are described in more detail below, as well as an embedded lightweight HTTP server for adjusting configuration state (Figure 28). Each element of the architecture is explained in more detail in the following sections.

Figure 28
OnObject system architecture.



4.1 Implementation Strategy

Throughout this research, the goal has been to demonstrate the possibility of simple yet structured "programming" of the physical world by end users. Therefore, the implementation effort is focused on the architecture of the OnObject platform for application creators, a broad definition that includes preschool children, their parents, hobbyists, interaction designer and game developers.

For a TUI developer, the OnObject device itself is relatively easy to produce. Instead of investing much time in the engineering of the device, more substantial work was spent in three areas:

Fixed Gesture Set

Getting a small yet rich enough set of gestures to work reliably as a part of the programming language.

End User Programming Tools

Supporting three high-priority user groups. 1) Completely screen-free and device-only programming for improvisation by children and laypersons; 2) The ability for more end users to incorporate existing media content using on-screen tool; and 3) Propagating keyboard events for very broad applicability for hobbyists and prototypers to control existing and custom software programs.

System Architecture

Design and development of one API to enable all three tools.

4.2 Hardware

The OnObject device consists of a 3-axis accelerometer, a high-frequency RFID reader, 3 LEDs, a button, and a microprocessor which interfaces with each of these and with the USB serial port on the host computer. See Section 3.1.2 for pictures of the device.

4.2.2 Components

Accelerometer

OnObject uses the Freescale MMA7260 tri-axis accelerometer, sampled at approximately 20Hz, and configured at the "4g" setting to reduce saturation for high-energy motions like shake and thrust.

RFID Tags

OnObject currently employs high-frequency (13.56MHz) RFID tags with ISO15693 protocols, which are as compact as 9mm diameter x 2mm thick (disk) or 13 x 13 x 0.8mm (sticker). As these passive tags are inexpensive and do not require electrical power, repairs, or upgrades, the tags are easily applicable to existing household objects and surfaces, and even to garments that can be washed.

Tag Reader

We used Tagsense Micro-1356 and Skyetek M1 Mini readers for current prototypes. The RFID antenna (23-38mm diameter) is significantly smaller than ReachMedia (wrist-sized) or Berlin's toolkit (110x75mm) [4, 2] and is located on the user's palm by default. Powered with 3.3-5V, the maximum reading distance is 12-33mm respectively when reading a 15mm-diameter tag. The small sensing area, distance, and compact form factor makes it conducive to detecting objects held firmly in hand or near grasp.

Indicator LEDs and Programming Button

Each device includes three LEDs for feedback when an object is first grabbed, the "program" button is pressed, and when it detects a certain level of overall motion. It is also available for application-specific feedback. The button is an unambiguous UI element for initiating the TGR flow described earlier.

Microprocessor

The current hardware is built on the popular Arduino platform using a ATMEGA328 processor. The Arduino program communicates with the recognizer via the serial port.

4.2.3 Antenna Design

An external antenna can be employed to increase sensing area or to specifically sense objects the user's finger touches. We developed custom antennae for the Tagsense RFID reader. Most antennae measure 38-46mm in diameter, contain two loops of multi-strand 22 AWG wires to form the inductor, and are connected to 270-285pF capacitors in series to record approximiate 13.56MHz resonance frequency. When directly connected to the reader and the computer, these antennae recorded a maximum reading area of 50mm W x 60mm D x 33mm H. (Figure 29).

Antennae can be in different sizes and form factors. For instance, a fingertip-size antenna with 593pF capacitors, made with three 18mm diameter loops of wire, measured 17mm reading distance at the resonance frequency of 13.7MHz (Figure 30). See Appendix A for the hardware connection diagram, including how an antenna is connected to the Tagsense reader.

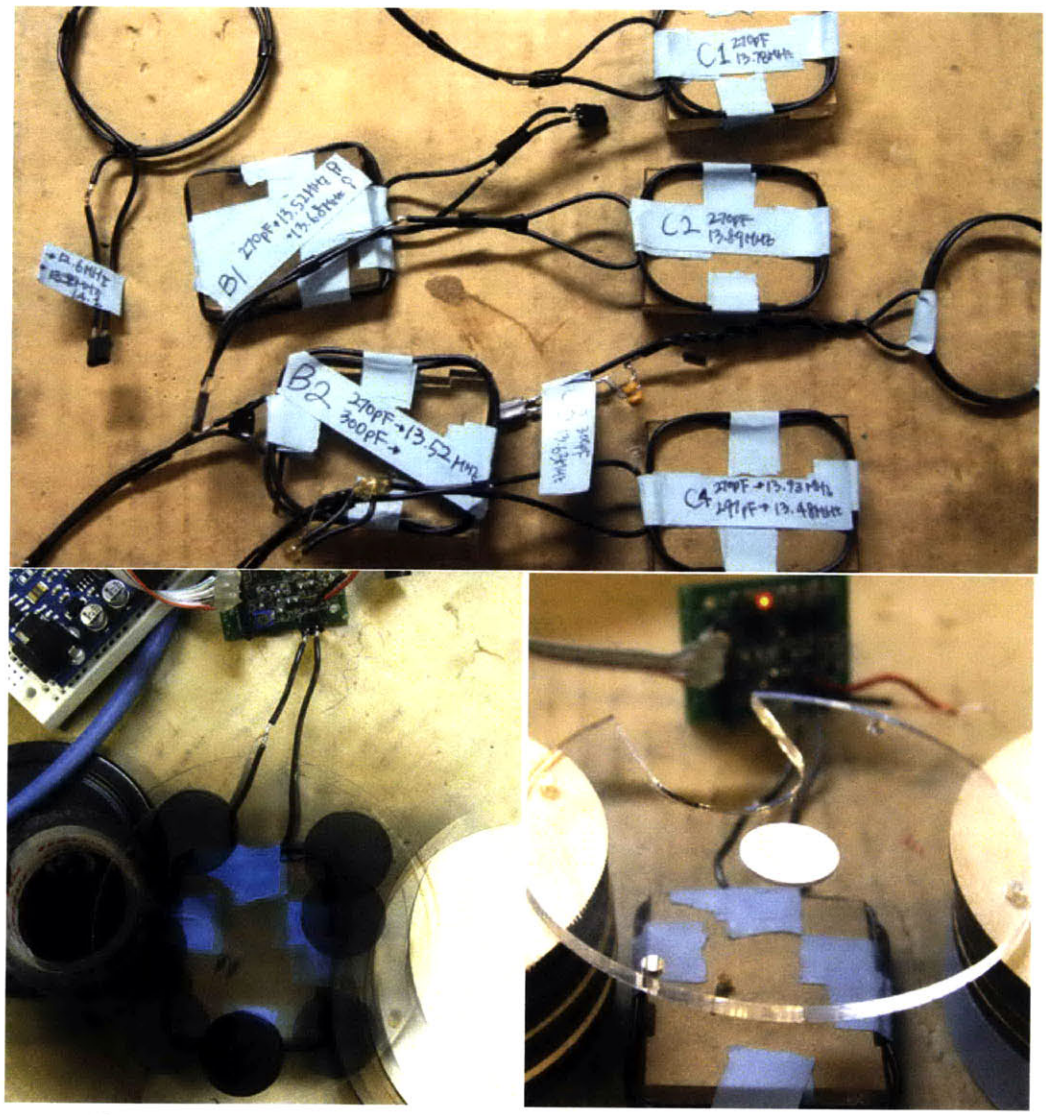


Figure 29 Handmade antennae for Tagsense reader (top), testing the reading range (bottom).

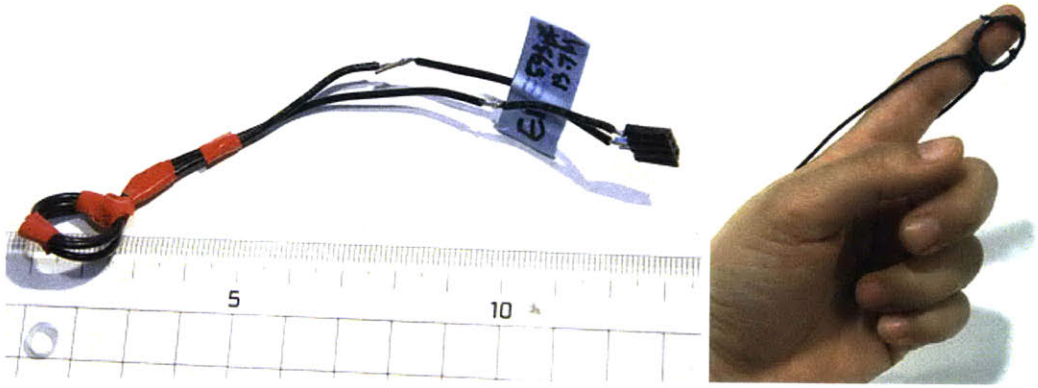


Figure 30 Fingertip antenna.

4.3 Software

4.3.1 Device Firmware

Written for the Arduino platform, the code performs the following tasks:

- Continuously send current tag ID (16-byte Os during absence) and x, y, z acceleration values to the gesture server via serial communicaiton.
- Control indicator LEDs based on tag presence, button press, and gesture activity received from the gesture server.

See Appendix B for the firmware code.

4.3.2 Gesture Server

Gesture Events

Gesture events are emitted from the server over a TCP socket interface. Applications can connect directly to the socket to receive a stream of events. Alternatively, the KeyMapper can sit between the socket and an application and flexibly encode gesture events into system keyboard events. This creates a loose coupling between the tool/application and the recognizer, so the application will only receive events when it has system focus. It is simple, makes OnObject compatible with most existing applications, makes it easy to interface new applications with the system, and is performant.

Configuration

OnObject's configuration is a mapping from tag IDs to valid gesture events. This information is stored in a configuration file that can be hand-edited, but to create the TGR flow we use in our tools, we also provide full application access to the configuration using a RESTful HTTP interface. The interface allows clients to view the

current configuration, view the entire set of known gestures, retrieve the contingent accuracy of any subset of gestures, update the tag to gesture to event mapping, and persist the mapping to file.

4.3.3 Gesture Recognizer

OnObject's recognizer is a layered adaptation of two existing approaches. A local recognizer estimates the current gesture on a rolling, fixed-size window of the input signal by applying a learned Decision Tree to a set of 29 features. At the global level, it feeds the output of the decision tree into a learned Hidden Markov Model (HMM), which considers the current window in context. By dynamically adapting the HMM based on the current object in the user's grasp, OnObject is able to constrain the recognition problem and considerably boost accuracy.

Recognizer Design

The recognizer is designed with several key design criteria in mind. It is extensible to support a variety of gestures through learned models. It is accurate, with high precision and recall, despite noisy accelerometer input data. Decision Tree and HMM learning are simple, well understood by practitioners, nearly ubiquitously available, and fast. Therefore the OnObject recognizer is simple, easy to understand, and easy to reimplement on any platform, including low-cost embedded hardware, and in any programming language. Finally, it can easily adapt to the current object in the user's grasp, implementing the constrained recognition aspect of the TGR flow, without having to retrain the system.

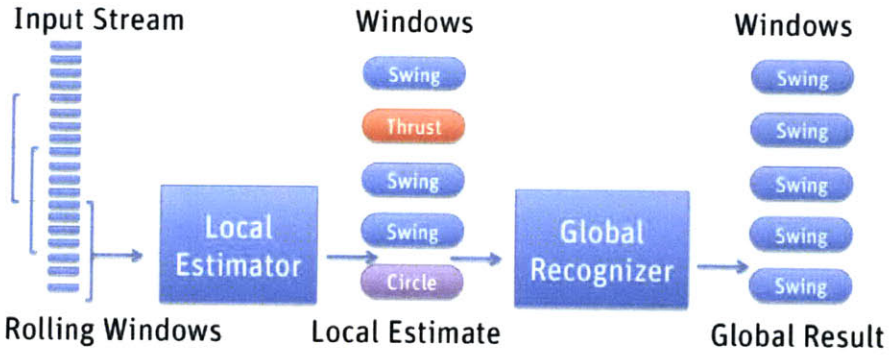


Figure 31 Layered recognizer design.

Local Estimator

OnObect's local estimator is a learned Decision Tree that is applied to a fixed-size window of input samples. For each window in the input data, we extract a collection of 29 signal features as shown in Table 3. These features are borrowed from the Wockets system, a superset of Kim's feature set [31, 13]. We train the local estimator using Weka's J48 decision tree learner [5].

Table 3
Features used in local estimator

INDEX	DESCRIPTION
1 – 4	Sum of axis means, pairwise difference in axis means
5 – 7	Variance of each axis
8 – 10	Range of each axis
11 – 22	FFT top K frequency/magnitude pair for each axis (K=2)
23 – 25	Power of each axis
26 – 29	Pairwise signal cross correlation between axes

Global Recognizer

The global recognizer in the system is a Hidden Markov Model [10], with a simple modification to support constrained recognition based on the modality of the application. The outputs of the local estimator are fed to the HMM. The HMM converts these output into a class probability, and then smoothes these outputs, by encoding the typical transition probabilities (a gesture typically lasts for 6-10 windows before it transitions to idle or another gesture).

We encode constraints in the system by dynamically updating the emission and transition matrices in the HMM so that inactive gestures map to the "idle" state. This is a crude way to achieve a global constraint on the HMM, similar in spirit to the more fine-grained constraints described in CueTIP [22]. Because an HMM is implemented as a pair of emission and transition probability tables based on frequencies, updating the HMM to implement a constraint is as simple as moving counts between the appropriate entries in the table. This can be done efficiently without introducing overhead to the system as the user grabs and releases tags.

Figure 32
Global recognizer (last column) corrects erroneous local estimator result (red).

X accel data	Y accel data	Z accel data	Original label	Local result	Global result
300.000	296.000	384.000	none	----	----
300.000	296.000	388.000	none	----	----
304.000	296.000	384.000	none	----	----
300.000	296.000	388.000	thrust	none	none
296.000	296.000	384.000	thrust	----	----
304.000	296.000	388.000	thrust	----	----
304.000	304.000	396.000	thrust	----	----
304.000	288.000	372.000	thrust	none	none
300.000	296.000	388.000	thrust	----	----
304.000	284.000	384.000	thrust	----	----
288.000	264.000	380.000	thrust	----	----
292.000	220.000	376.000	thrust	none	none
324.000	164.000	356.000	thrust	----	----
204.000	176.000	356.000	thrust	----	----
260.000	204.000	364.000	thrust	----	----
460.000	180.000	336.000	thrust	circle	none
376.000	268.000	340.000	thrust	----	----
672.000	448.000	216.000	thrust	----	----
36.000	688.000	692.000	thrust	----	----
0.000	192.000	144.000	thrust	thrust	thrust
668.000	400.000	348.000	thrust	----	----
384.000	324.000	360.000	thrust	----	----
308.000	288.000	372.000	thrust	----	----
320.000	296.000	388.000	thrust	thrust	thrust
276.000	284.000	348.000	thrust	----	----
292.000	296.000	420.000	thrust	----	----
320.000	284.000	392.000	thrust	----	----
292.000	292.000	380.000	thrust	thrust	thrust
308.000	284.000	376.000	thrust	----	----
292.000	292.000	392.000	thrust	----	----
296.000	284.000	380.000	thrust	----	----
308.000	288.000	384.000	thrust	thrust	thrust
304.000	292.000	388.000	thrust	----	----
300.000	284.000	376.000	thrust	----	----
300.000	284.000	376.000	thrust	----	----
296.000	296.000	396.000	thrust	thrust	thrust
304.000	284.000	376.000	thrust	----	----
300.000	288.000	380.000	thrust	----	----
300.000	296.000	392.000	thrust	----	----
296.000	284.000	372.000	thrust	thrust	thrust
296.000	288.000	376.000	thrust	----	----
308.000	292.000	388.000	none	----	----
304.000	292.000	376.000	none	----	----
288.000	296.000	384.000	none	thrust	thrust
300.000	300.000	396.000	none	----	----
300.000	292.000	388.000	none	----	----
296.000	300.000	396.000	none	----	----
300.000	304.000	404.000	none	thrust	thrust
308.000	284.000	380.000	none	----	----
300.000	292.000	392.000	none	----	----
300.000	280.000	376.000	none	----	----

Rolling windows of 20 data points are pattern-matched every 4 points.

Evaluation

We trained and evaluated the OnObject recognizer on a data set containing 400 of each gesture, and 2400 idle gestures [Table 4]. We found that the recognizer was sensitive to both the user who generated the set of gestures, as one would expect, but that with hundreds of examples from each user the learner was able to generalize. We wanted to understand the performance of the local estimator versus the global recognizer. Given an accurate local recognizer, it is possible to build more complex global recognizers, such as Conditional Random Fields (CRFs) or Probabilistic Context Free Grammars (PCFGs) [10].

Because nearly 30% of the errors are between active gestures, as opposed to between active gestures and the idle state background model, constraining the recognition allows us to not only eliminate those errors completely, but also turn local improvements into more pronounced global improvements.

Table 4
Within-subject evaluation results.

Gesture	N	**Local Estimator**			**Global Recognizer**		
		Precision	Recall	Accuracy	Precision	Recall	Accuracy
Circle	400	0.93	0.90	**0.99**	0.98	0.98	**0.99**
Swing	400	0.82	0.85	**0.97**	0.93	0.94	**0.99**
Thrust	400	0.81	0.97	**0.98**	0.89	0.99	**0.99**
Shake	400	0.80	0.96	**0.98**	0.90	0.99	**0.99**
Fan	400	0.77	0.74	**0.96**	0.86	0.85	**0.98**
Tilt	400	0.80	0.82	**0.97**	0.90	0.90	**0.98**
Idle	2400	0.93	0.88	**0.91**	0.96	0.88	**0.92**
Total	**4800**			**0.87**			**0.94**

Adaptive Recognition Strategy

The gesture recognizer is a work in progress. Although the constrained HMM performance has been tested with within-subject data, it still warrants more data and experiments to validate its performance with a larger number of participants. However, it is important to note that in the context of OnObject applications, the recognizer should be primarily tested for single-user, within-session use.

The user studies and the live demonstrations described in Chapter 5 used an interim solution: local results from decision tree were run through simple heuristics and returned global results based on consecutive number of identical local results. The recognizer returns both local and global results, and some applications make use of both. The single shake recognizer seen in the videos was based on a heuristic that measured cumulative power across a number of short windows.

4.3.4 Development Tools

In the course of the implementation, several software tools were created or borrowed to develop the gesture recognizer.

Plot Real-time Tag and Acceleration

Based on Tom Igoe's work [68], a Processing sketch was created to view tag ID and acclerometer data real time.

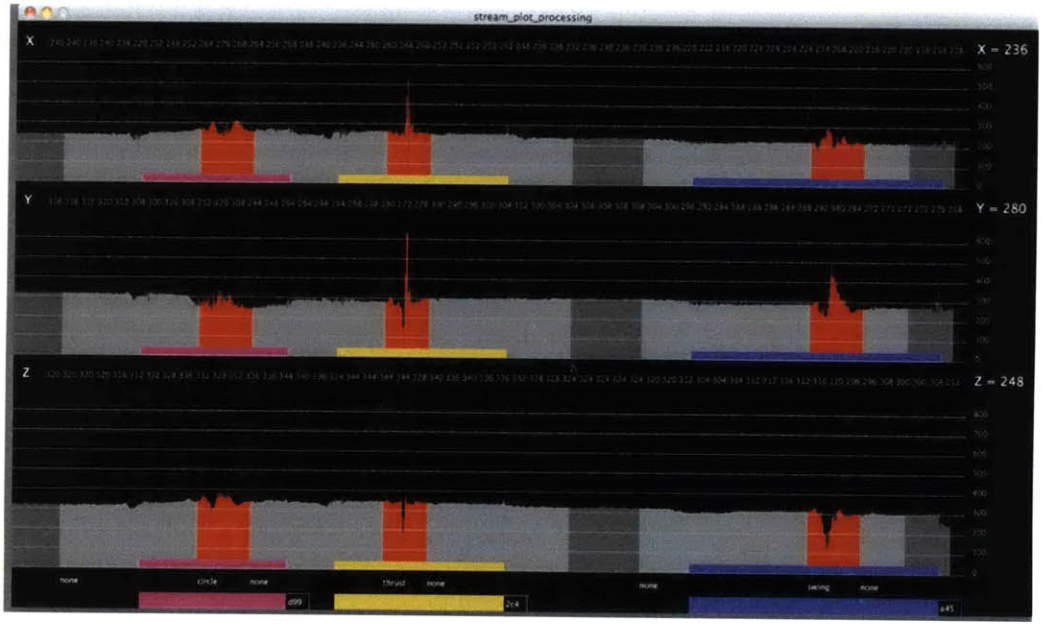


Figure 33 Tag and accelerometer plotter Processing sketch.

Export Acceleration Data to CSV

To sample labeled gesture examples, the sketch was further modified to record and export CSV data real-time based on keystroke commands. Single gesture CSV was organized in order of label ("shake"), number of axis (3), number of data points in each axis (36), $X_0 - X_N$, $Y_0 - Y_N$, and $Z_0 - Z_N$.

```
shake, 3, 36, 268,264,268,264,264,264,264,264,268,268,268,272,2
76,288,504,680,0,108,484,444,240,224,268,288,268,240,252,252,25
6,268,264,252,260,264,260,260,324,328,320,324,328,320,324,324,3
20,324,324,320,324,320,372,40,64,412,348,236,328,312,316,312,32
0,324,316,320,328,324,324,324,320,320,324,324,356,352,356,356,3
56,356,356,360,360,356,360,360,360,364,344,96,0,328,420,420,308
,328,360,372,364,352,352,356,356,356,352,352,352,356,356,356,3,
36,268,264,268,264,264,264,264,264,268,268,268,272,276,288,504,
680,0,108,484,444,240,224,268,288,268,240,252,252,256,268,264,2
52,260,264,260,260,324,328,320,324,328,320,324,324,320,324,324,
320,324,320,372,40,64,412,348,236,328,312,316,312,320,324,316,3
20,328,324,324,324,320,320,324,324,356,352,356,356,356,356,356,
360,360,356,360,360,360,364,344,96,0,328,420,420,308,328,360,37
2,364,352,352,356,356,356,352,352,352,356,356,356
```

Plot and Crop CSV Data

Once saved as a CSV file, the data can be loaded to another Processing sketch for viewing, and can be cropped to remove extraneous parts with a mouse drag.

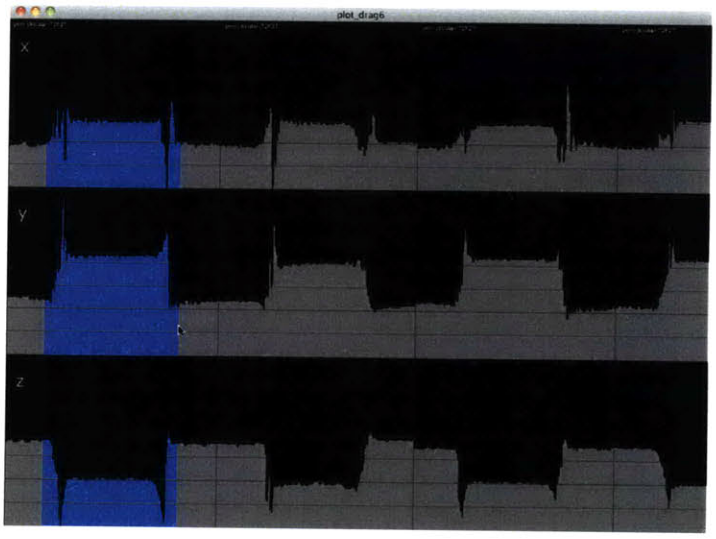


Figure 34 Cropping a CSV file using a Processing sketch.

Weka Decision Tree

Using the feature set in Table 3, a Weka ARFF file is generated for each training example set.

```
@attribute feature19 NUMERIC
@attribute feature20 NUMERIC
@attribute feature21 NUMERIC
@attribute feature22 NUMERIC
@attribute feature23 NUMERIC
@attribute feature24 NUMERIC
@attribute feature25 NUMERIC
@attribute feature26 NUMERIC
@attribute feature27 NUMERIC
@attribute class {thrust,none,circle,swing}

@data
0.331456108689,0.525953471561,0.194497362873,0.0949021693
5103067606,23.6592620254,11.1746031746,342.021753359,10.1
9196722136,10.1587301587,244.738314766,9.14285714286,139.
-0.45436536443,-0.544980175604,-0.0906148111745,-0.835238
5463877948,15.8131046819,11.1746031746,431.711087923,10.1
```

Figure 35 ARFF file of sample gestures (partial view).

The ARFF file is loaded to Weka, an open source machine learning software. Using J48 learner, Weka generates a decision tree. The decision tree is loaded in the Python recognizer library for local estimator.

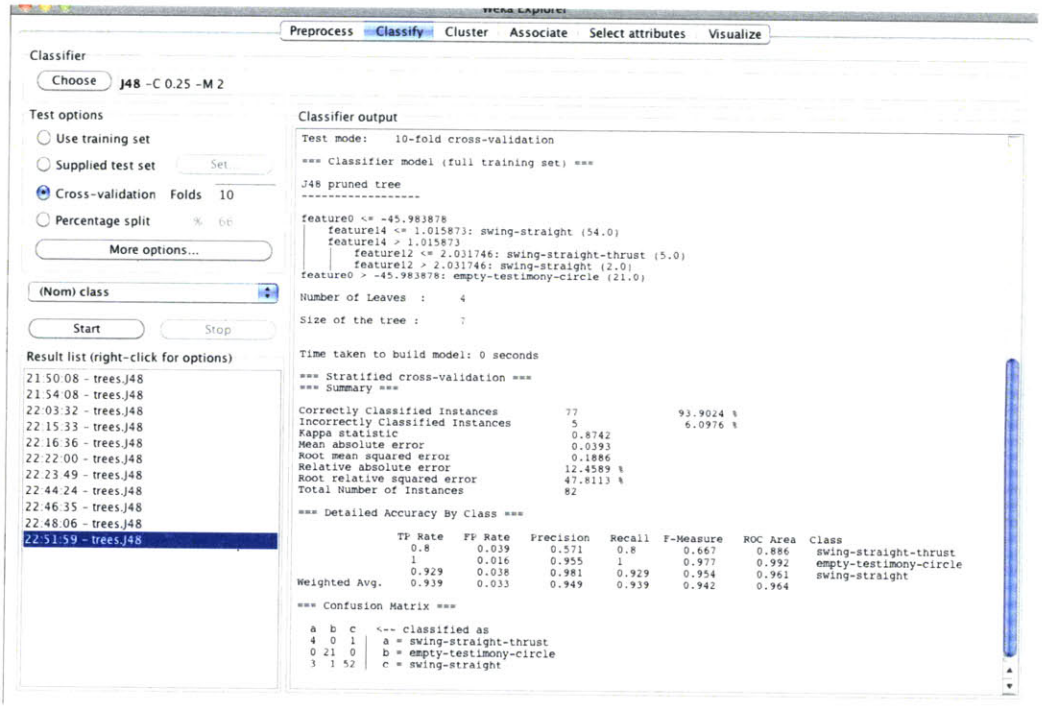


Figure 36 Generating J48 decision tree using Weka.

Chapter 5

Applications & Evaluation

We developed several applications to communicate and assess the appeal and utility of OnObject to end users, including educational and play activities applications, video gaming applications, and prototyping applications for product and TUI designers.

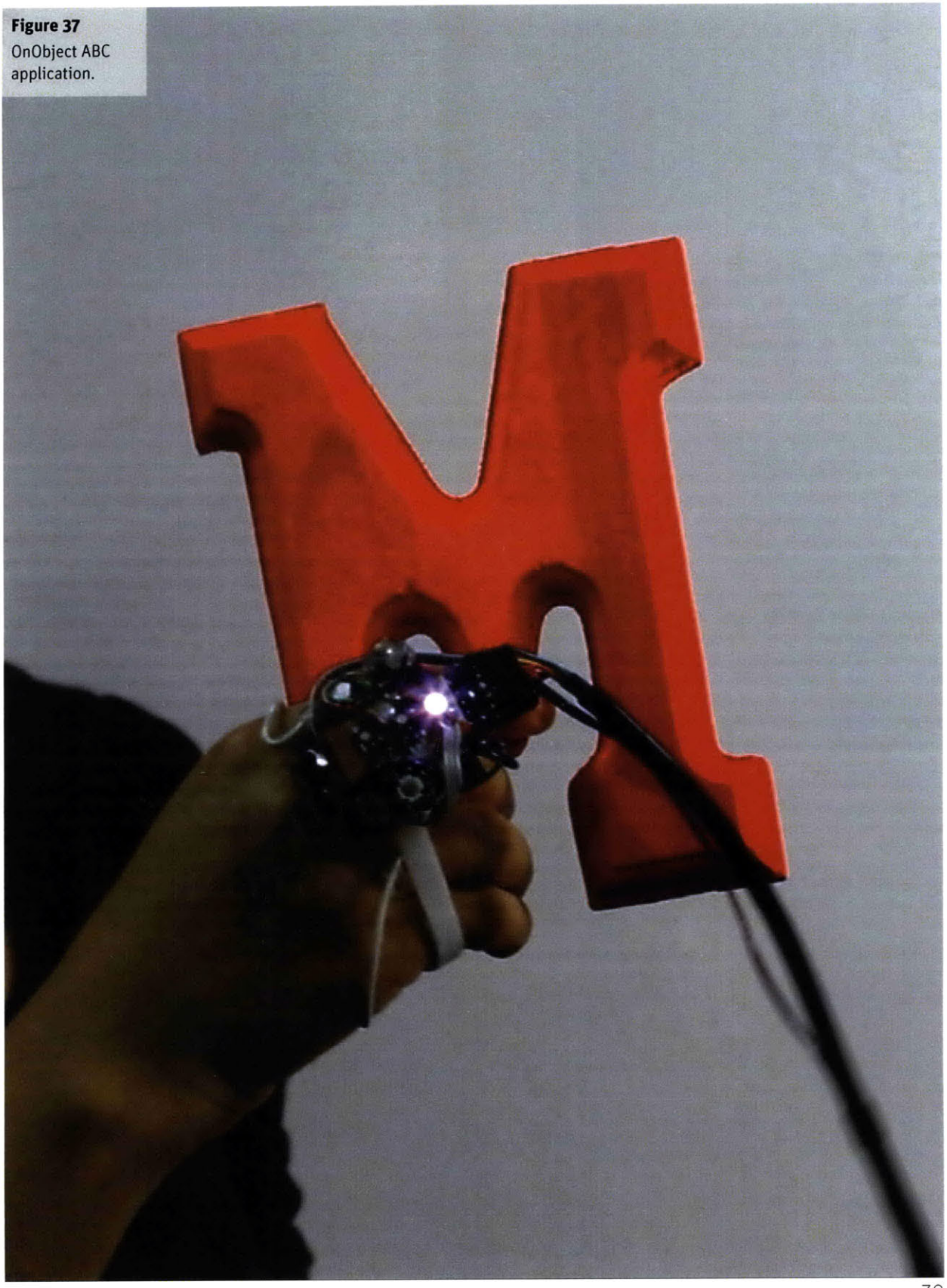


Figure 37
OnObject ABC application.

5.1 Reconstructing Existing TUIs

OnObject can be used to rapidly prototype an interesting and useful subset of TUI interactions using any physical objects and environment, without requiring custom-made artifacts with embedded circuitry and a power source.

5.1.1 Marble Answering Machine and musicBottles

Through the simple Tag-Gesture-Response flow, a casual end user can use OnObject to recreate or approximate many classic and contemporary TUIs in a matter of minutes. Table 5 shows how TUIs can be programmed using Media I/O, such as Marble Answering Machine (1992), musicBottles (1999), and Amagatana + Fula (2008) [9, 12, 17].

Figure 38 Marble Answering Machine (left), musicBottles (center), Amagatana + Fula (right).

Table 5
OnObject configurations for example TUIs.

TAG	GESTURE	RESPONSE	OPTIONS
Marble Answering Machine			
Ball 1	Grab	Play voicemail	On start, no repeat
	Shake	Archive voicemail	On start, no repeat
musicBottles			
Bottle 1 neck, bottle 1 cap	Separate	Play cello track	On start, no repeat
Bottle 2 body	Tilt	Play piano track	On start, no repeat
Bottle 3 body	Shake	Play cymbal	On start, repeat
Amagatana + Fula			
Umbrella handle	Swing	Play sword sound	On start, no repeat
	Thrust	Play kiai sound	On start, no repeat
	Swing, swing, thrust	Play explosive sound	On start, no repeat

Join and Separation of Tags

Subject to the firmware of the RFID reader, an OnObject application can be modified to detect separation and joining of two RFID tags within a 40mm area.

5.1.2 OnObject Amagatana

We can use a series of multiple gestures to trigger compound effects, similar to the original Amagatana + Fula implementation where an ordinary umbrella gave an illusion of dramatic swordplay with sound effects. For example, a thrust followed by two swings produces special effects such as an explosive sound, allowing richer and more dramatic experience.

On-Screen Interface

This application was implemented using KeyMapper configuration file and a Flash program that receives the keystrokes and produces appropriate sound in response (see Section 3.4.3 for KeyMapper).

Using Media I/O, user would configure two end tags using the on-screen interface like the following screen mockup (see section 3.4.2 for Media I/O):

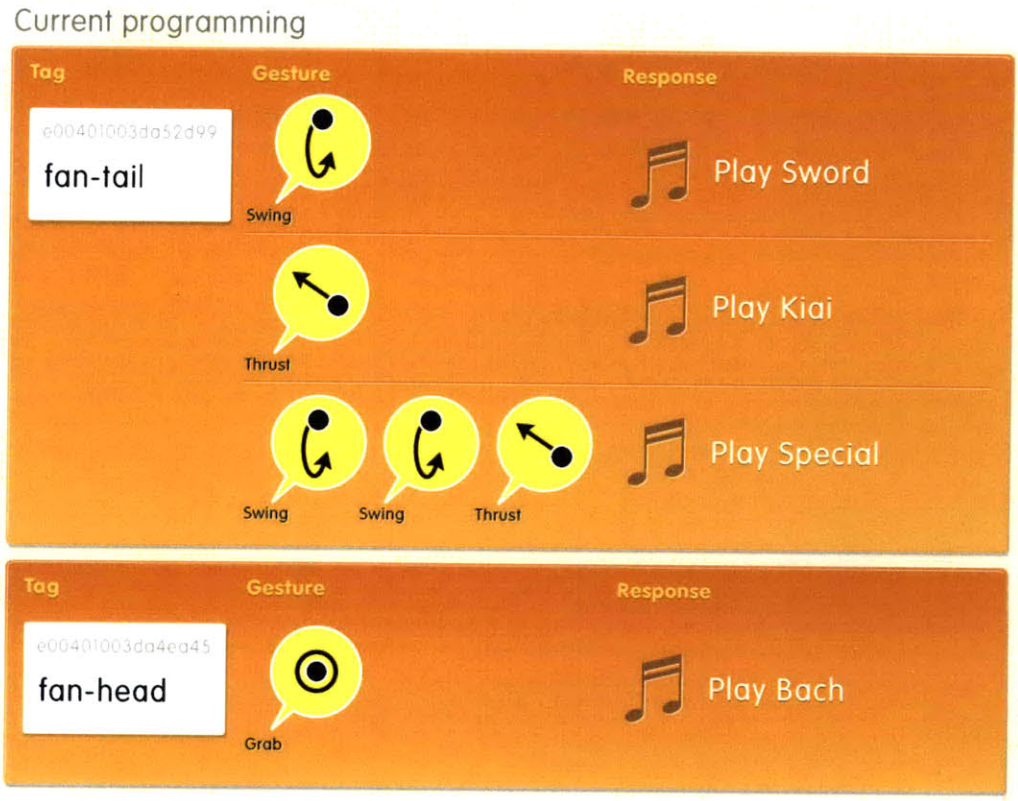


Figure 39 Tag-Gesture-Response mapping for a dramatic swordplay performance.

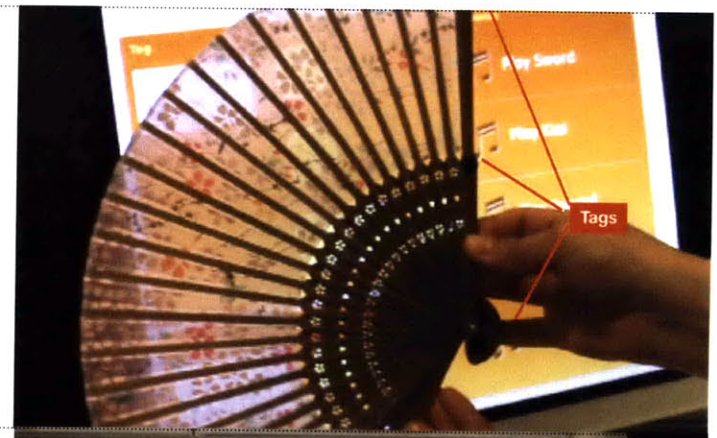
User Interaction Sequence

Table 6 shows how a folding fan was used for a dramatic swordplay performance. Real-time demonstration video is available online [63].

Table 6
Dramatic swordplay performance with a folding fan.

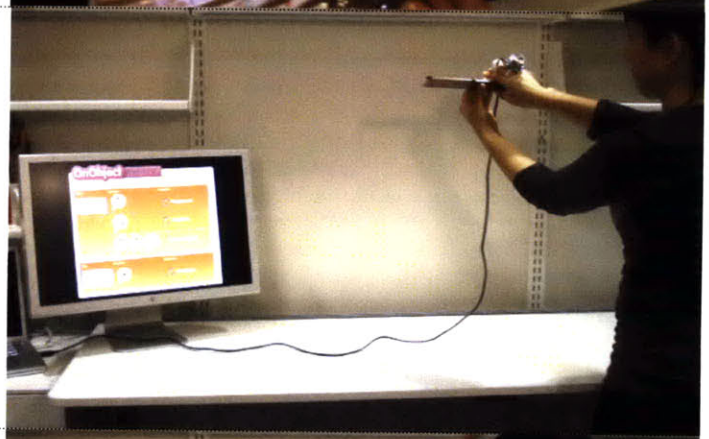
1 Tag

Attach RFID tags to both ends and middle of a folding fan.



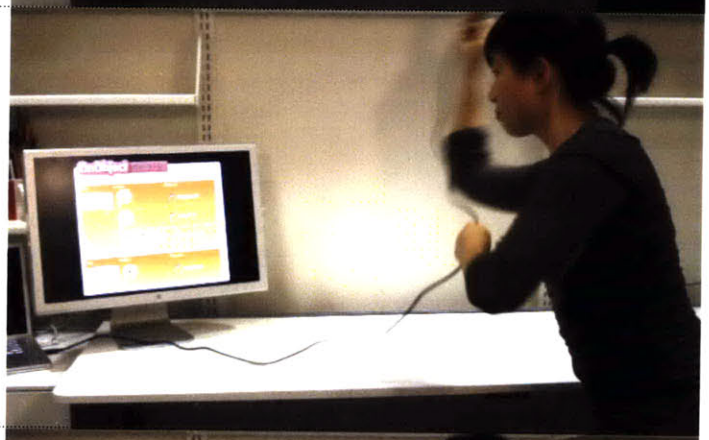
2 Grab tail

Grab the tail end of the fan to hear default sound feedback (chime).



3 Swing

With the fan grabbed by the tail end, swing the fan to hear sharp sword swing sound effect.



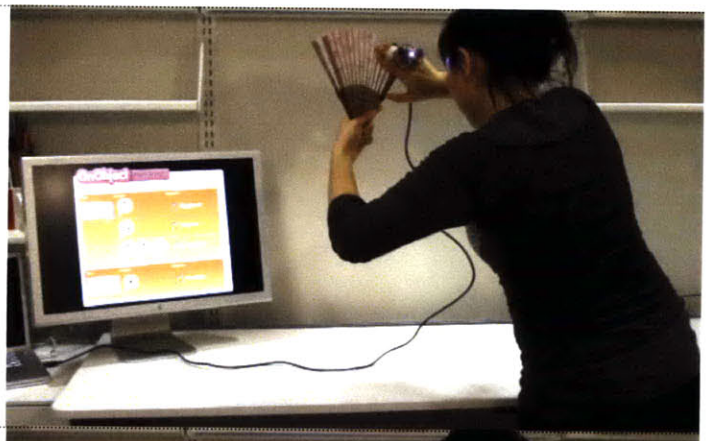
4 Thrust

Thrust the fan to hear mail voice kiai ("hah!") sound effect.

Table 6 continued

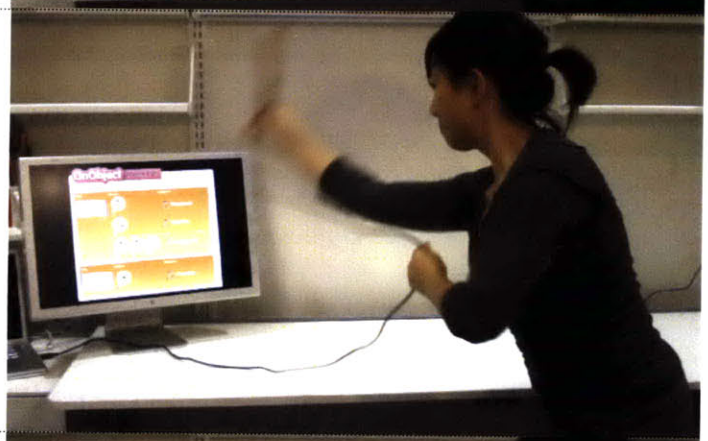
5 Grab head

Dramatic Bach music segment plays.



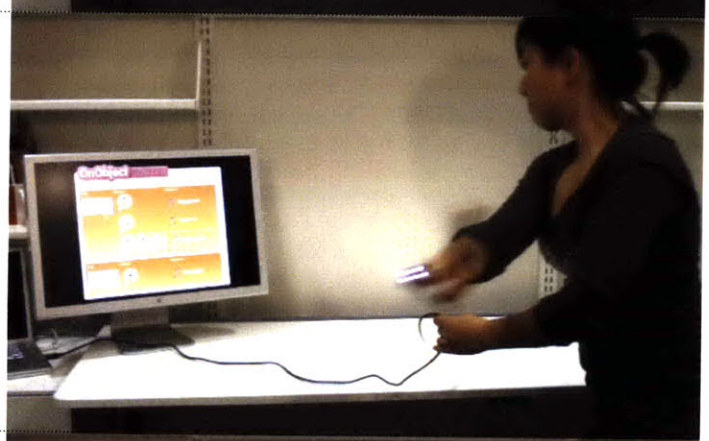
6 Swing

Sword swing sound plays.



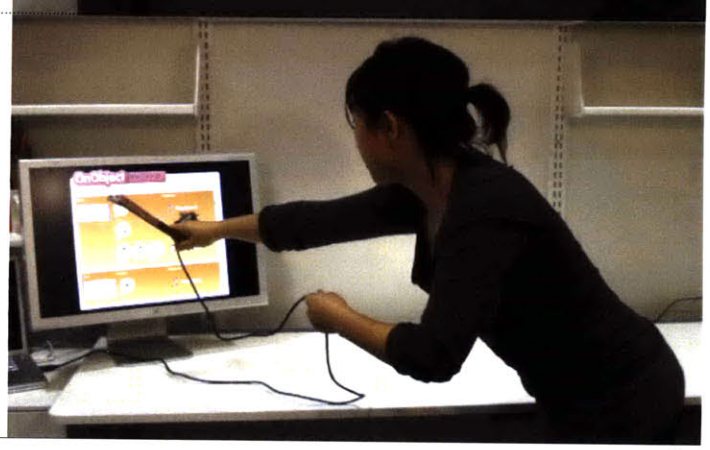
7 Swing

Sword swing sound plays.



8 Thrust

Swing+Swing+Thrust = Special effect: A special thunderous sound effect plays.



Required Tools

- Folding fan.
- ISO 15693 13.56MHz RFID tags.
- Adhesive tape or glue gun to attach tags to the fan.
- OnObject device with microphone, communicating with nearby computer with speakers.
- KeyMapper configuration text file.
- Flash application with sound effect files, supporting Python software libraries.

OnObject as a Scalable TUI Prototyping Platform

This application accentuates the merits of OnObject as a gestural TUI prototyping platform compared to the one-off construction of hardware-embedded objects. Figure 40 compares original Amagatana umbrellas (top) with OnObject fan (bottom), approximately in scale.

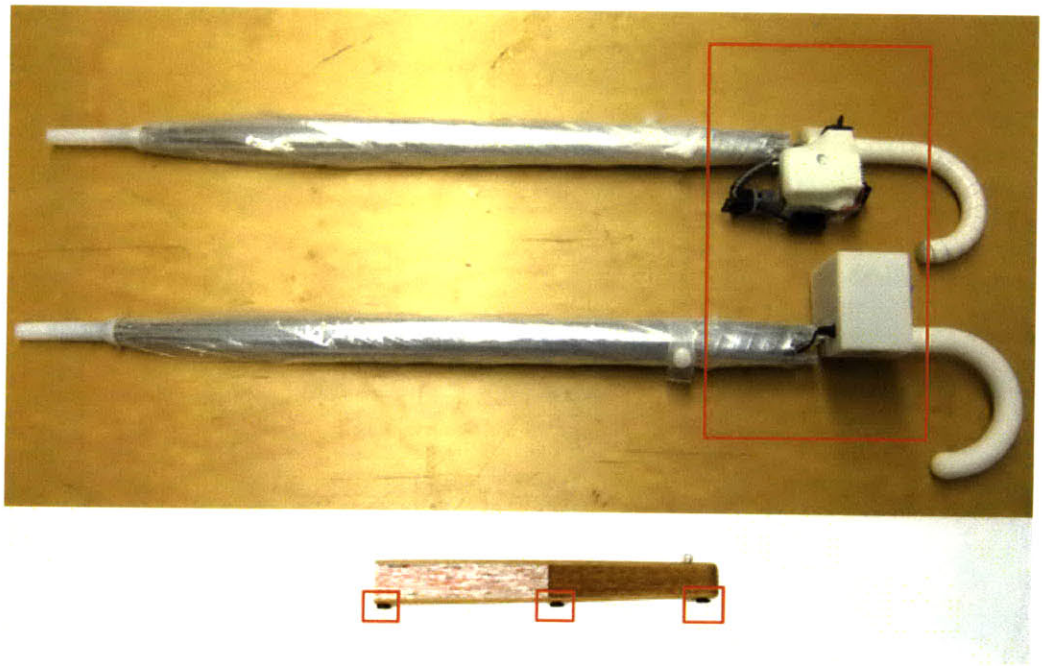


Figure 40 Amagatana umbrellas (top) compared to OnObject fan (bottom).

Modification of Host Objects: As highlighted in red boxes, Amagatana's umbrellas have been significantly modified to included embedded hardware, to the point that these objects can not serve their original purpose. In contrast, OnObject fan has 9mm diameter tags attached to it, largely retaining the object's form factor and function. Also the modification is almost always reversible, where the tags glued or taped can be removed without damaging the original object.

Expertise Required: It requires considerable knowledge to create the necessary hardware, program the firmware, mount it onto the host object properly and fabricate the housing. Even those who possess the skills have to invest weeks of time to design and execute applications like Amagatana. End users simply do not possess such skills or incentive to invest nearly as much time to see a simple interaction happen. With OnObject, the hardware setup is already complete. Users only have to complete simplified software tasks of clicking through GUI screens (with Media I/O) or modifying a KeyMapper configuration text file and a Flash file.

Flexibility and Scalability: Even with necessary skills, creators of one-off TUIs like Amagatana have to invest another chunk of considerable time, materials and efforts to make changes to the interaction. For instance, what if a user decides that she wants two more umbrellas with same effect? She would have to create new circuit boards, assemble and program them, house and mount them again. What if she then wants to hold the umbrella by the top instead of the handles? She would have to dismantle the embedded hardware and carefully re-mount it at the other end, which can take hours to weeks if it requires repairs. In contrast, using OnObject she could simply attach more tags to more umbrellas, or move existing tags to the other end of the umbrella and she would be finished in minutes.

5.2 Tangible Thinking in Product Development

One of main virtues of TUIs is tangible thinking, where physical and direct manipulation and bodily engagement aids in problem solving and creativity [51]. Similarly, design and creative professionals have emphasized the value of rapid physical prototyping of product, toy, or scientific ideas beyond two-dimensional sketching. In his book *The Art of Innovation*, IDEO's Tom Kelly encourages brainstomers "to have materials on hand to build crude models of a concept: blocks, foam core, tubing, duct tape, whatever might be useful. [54]"

In addition to creating gestural interfaces from any situated objects, OnObject users can create, sculpt and prototype the objects themselves on the spot. If you want to create an interactive musical instrument unlike anything you can find in the house, you can instantly prototype one using blocks, clay, or even junk and apply RFID tags to them.

Therefore, using OnObject, professional toy and product designers or casual users can create prototypes that are not only visual, physical, but also interactive in minutes: a tubing snake that hisses when you grab it, a dusting mitt that laughs when you shake it hard, a bathrobe that plays music when you massage on the neck, and so on.

5.2.1 OnObject Clay

OnObject Clay is a prototyping application where user sculpts an ad-hoc toy using clay, LEGO blocks, foam chunks and glue, attach tags to the prototype, perform gesture trigger, and then record sound response into the microphone. Current version of the Clay application can be used using only the device.

Figure 41 shows toy prototypes made with a previous version of OnObject Clay, which let users specify a tag (labeled A-Z) and pick an audio response for a gesture using an on-screen menu. For a detailed description of a user test session, see Section 5.4.3.

Figure 41 An assortment of makeshift interfaces constructed with OnObject Clay.

5.3 Storytelling and Entertainment

To emphasize the core aspects of user-defined object programming and novel interactions created with OnObject, three applications were developed and demonstrated to the Media Lab visitors and preschool children.

5.3.1 OnObject ABC

An obvious application of the ability to record sound feedback into physical objects is to teach children words and concepts associated with each object.

Scenario

A parent, babysitter, or sibling teaches words starting with alphabet letters M and P by programming the letters to say their names when grabbed, and name words that begin with the letter when shaken or swung via a programming sequence outlined below.

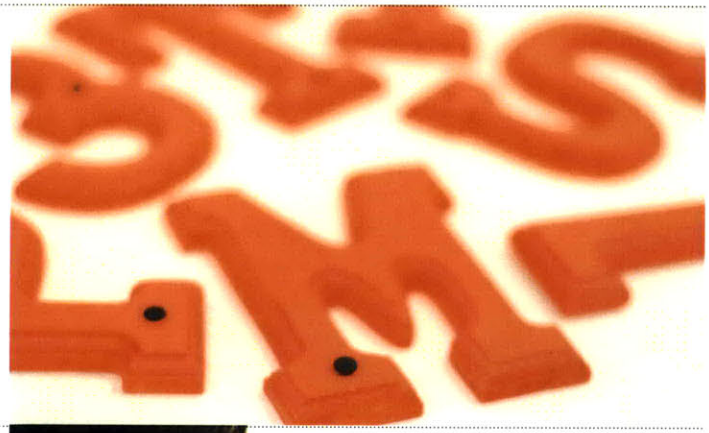
User Interaction Sequence

Table 7 describes how to program an M to produce "M is for" sound when grabbed, "monkey" when shaken. A real-time demonstration video is available online [59].

Table 7
Program an M to produce "M is for" sound when grabbed, "monkey" when shaken.

1 Tag

Attach RFID tag to an M letter.



2 Gesture

Grab the letter by the tag wearing the device; default sound response (chime) plays.

3 Response

Hold button down and say "M is for" toward the letter; release the button when finished.

4 Gesture

Shake the letter while grabbing it by the tag; default sound response (boink) plays.

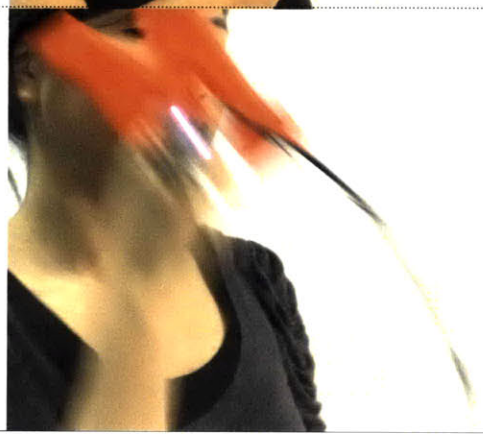
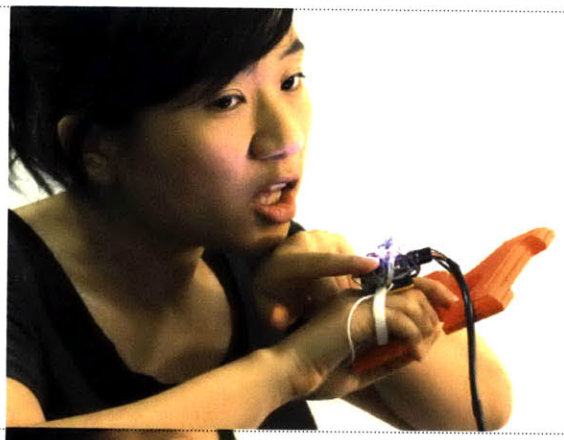
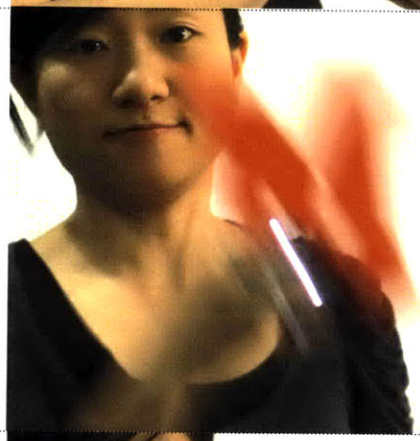
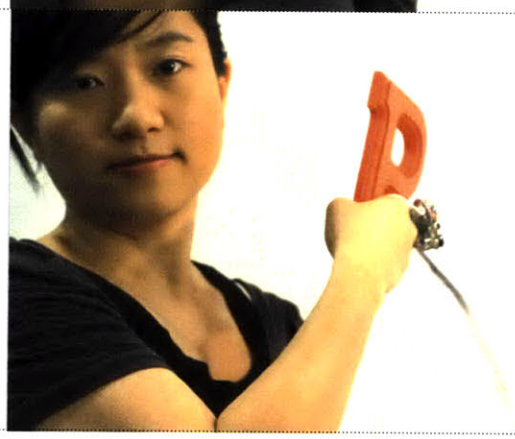


Table 7 continued

5 Response Hold button down and say "Monkey"; release the button when finished.	
6 Play Grab the letter to hear "M is for"; shake the letter to hear "Monkey!"	
7 Play Similarly, user can make the letter P say "P is for" when grabbed, and "Piggy!" when swung .	

Required Tools

- Wood or cardboard alphabet letters.
- ISO 15693 13.56MHz RFID tags.
- Adhesive tape or glue gun to attach tag to the letters.
- OnObject device with microphone, communicating with nearby computer with speakers
- Application software and supporting Python library running on the same computer.

Fast, Simple and Flexible

In the real-time demonstration video, first set of T-G-R sequence (steps 1-3) takes 4 seconds, and entire programming sequence described above (steps 1-6) takes 9 seconds.

In a matter of seconds, user has transformed a mere piece of wood to an educational audio device — without any need for programming, wiring, or watching raw signals. Each letter's responses to gestural triggers can be reprogrammed as user desires. For example, a parent may start with "M is for" "Monkey" for beginner student, and progress to more abstract words such as "Money" or "Mind."

This ABC application is a simple yet powerful example that demonstrates the simplicity and flexibility of the OnObject system.

5.3.2 OnObject Storytelling

Taking the ABC application one step further, more conversational narrative structures can be created based on user's gestural actions.

Scenario

Based on their routine play activities, parents create a simple narrative with child's stuffed animals where the animals ask a child's name and incorporate her name into the consequent parts of the story.

User Interaction Sequence

Table 8 shows how to introduce the user's name to a frog and then get introduced to frog's friend kangaroo. Similarly, the user can introduce a tagged lion that scares the animals when grabbed to growl "what is happening?" and invites everyone to dance when shaken. A real-time demonstration video is available online [60].

Required Tools

- Stuffed animals or toys.
- ISO 15693 13.56MHz RFID tags.
- Adhesive tape or glue gun to attach tag to the letters.
- OnObject device with microphone, communicating with a nearby computer with speakers.
- Application software and supporting software library running on the same computer.

Table 8

Introduce my name to a frog, get introduced to frog's friend kangaroo.

1 Tag

Attach RFID tags to stuffed frog, kangaroo and lion.

2 Gesture

Grab the frog by the tag wearing the device; frog says "hey I'm froggy, what is your name?"

3 Response

Hold button down and say your name (e.g. "Keywon"); release the button when finished.

4 Gesture

Shake the frog to hear "Hello *[Keywon]*, let's go play with the kangaroo" — *your name* in your voice, and the rest in frog's voice.

Table 8 continued

5 Play Grab kangaroo by the tag to hear "hi there froggy... "	
6 Play "Hello *[Keywon]*, jump with me!"	
7 Play Shake the kangaroo to hear default response (boink) sound.	

Varying Degrees of Structure

Constructing interactive narratives with OnObject can be done in a varying degree of premeditated structure, from hard-coded to templatized, to freeform:

Table 9
Varying degrees of structure in recording-based storytelling applications.

Structure	Example	Description
Hard-Coded	**Scripted story** Frog grab: "What's your name?" User records: "Keywon" Frog shake: "This is my new friend *[Keywon]*" Kangaroo grab: "Hello *[Keywon]*, jump with me!" Video is available online [60].	Requires frog and kangaroo are configured beforehand to incorporate the user-recorded name with their pre-recorded lines. Hard-coded applications like this can be provided for common play scenarios.
Template-Based	**Madlibs** Frog grab: "Tell me an animal." User records: "Crocodile." Frog shake: "Froggy meets a *[crocodile]*. Tell me a verb." User records: "Twist." Frog shake: "They *[twist]* happily ever after."	Pairs up pre-recorded prompts with user-recorded words randomly for improvised and comical effects. Requires the application is aware of the superset of tagged animal characters and a set of pre-recorded prompts.
Freeform	**Simple Q&A 1** Frog grab: "What's your name?" Child records: "Keywon." Parent: "Who loves spinach?" Frog grab: "*Keywon*." **Simple Q&A 2** Frog grab: Default chime Parent records: "What does a frog say?" Frog shake: Default boink Child records: "Ribbit." Video is available online [61].	Without any modifications to the ABC application, parents and children can enjoy simple but engaging interactions.

This application showcases the range of interpretation and liberty users can take to create engaging activities using simple audio-only OnObject applications. Using only OnObject's basic features, a parent and child can have instructive and entertaining Q&A and simple story. With additional applications hardcoded and modified to introduce varying degrees of structure to the narrative, users can enjoy scripted or improvise stories.

5.3.3 OnObject Swordplay

Given further development to make OnObject compatible with video game platforms, gesture triggers can be mapped to other responses including in-game actions instead of simple audio responses.

Scenario

Video game players can now use any graspable everyday objects as in-game tools, creating personalized game controllers instead of relying on one remote controllers.

User Interaction Sequence

Copy-Paste (Table 10): Play videogame with a toy sword and a Sharpie pen.

Add a Modifier (Table 11): Make the Sharpie pen into a glowing sword in-game.

Real-time demonstration video is available online [62].

Required Tools

- Everyday objects.
- ISO 15693 13.56MHz RFID tags.
- Adhesive tape or glue gun to attach tag to the letters.
- OnObject device with microphone, communicating with nearby computer with speakers.
- Videogame platform with OnObject API (see section 3.4.4).

Table 10

Copy-Paste: Play videogame with a toy sword and a Sharpie pen.

1 Tag

Attach RFID tags to a toy sword and a Sharpie pen. Toy sword has been registered to be used as an in-game weapon (see section 3.4.4 on application programming interface).

2 Play

Swing and thrust gestures on the toy sword is mapped to in-game swordfighting actions.

3 Copy

Press button while holding the sword by the tag to hear a "sword copied" message.

4 Paste

Grab the Sharpie pen (or any tagged object) by the tag and press the button to hear "sword pasted."

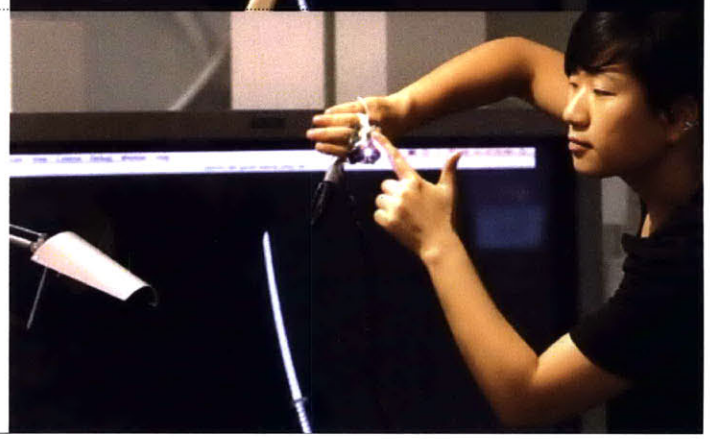


Table 10 continued

5 Play

Swing and thrust gestures on the pen is mapped to the same in-game swordfighting actions as with the toy sword.

Table 11

Add a Modifier: Make the Sharpie pen into a glowing sword in-game.

1 Copy

Press button while holding a lamp by the tag to hear a "lamp copied" message.

2 Paste

Grab the Sharpie pen (or any tagged object) by the tag and press the button to hear "lamp pasted."

3 Play

Swing and thrust gestures on the pen now result in a glowing sword moves in-game.

Copy-Paste of Object Programming

This application showcases a unique and novel interaction OnObject provides: An instant and physical duplication and propagation of object programming. After one object has been programmed to work within the gaming application, the particular Gesture-Response mapping can be duplicated to another object with two button presses (steps 1-5). Six or seven objects in the room may be programmed to act as an in-game sword in one minute using this method.

These capabilities invite users to appropriate existing objects for new virtual purposes in the game. In some cases, users may try to match the form factors of the physical object to the virtual weapon: trays in the kitchen may be used for a shield in the game. In other cases, physical affordances may be mismatched for creative outcome: a lightweight pencil may be used for faster maneuvering; non-stick-shaped objects like a round tube or a banana can be used to control in-game sword more ergonomically.

Taking this notion of copy-paste one step further, there can be modifier objects whose properties can be added to the virtual representation of the programmed objects. As seen in the steps 6-8 above, a lamp is a modifier object with a glow property. When the definition of lamp is added to the Sharpie pen, the pen acts as a glowing sword in the videogame.

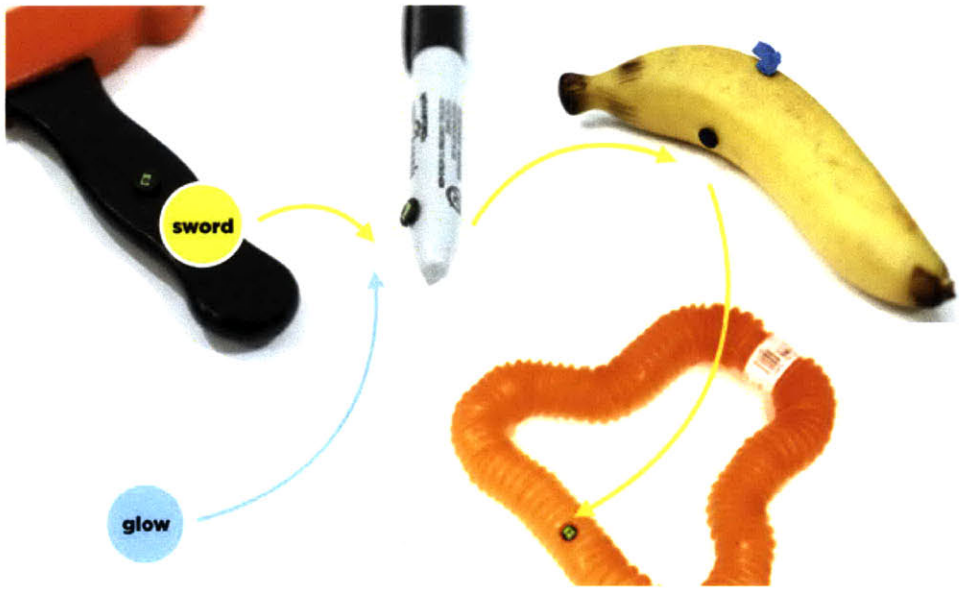


Figure 42 Propagation of programming via copy-paste.

Personalizing Play Tools

Based on this notion of copy-paste, users can duplicate the programming of an existing object to a new object and then modify or repurpose it to create a unique weapon or tool for themselves. We speculate game players can highly personalize their physical tools using OnObject: a Sharpie pen that is just the right size and weight for me that acts as a glowing sword in Zelda; a scarf that works as a cloak in-game that I can wave masterfully for dramatic effects; a water bottle that I've used for the last four months that works as all kinds of weapons in the game.

By appropriating and mastering situated physical objects, game players would be able to tout their personalized gaming tools not unlike master chefs or painters who often possess personalized and beloved tools, rather than using one-size-fits-all controllers provided by the manufacturer.

In fact, ABC, Storytelling and Swordplay applications demonstrate OnObject allows users to appropriate situated everyday objects as personalized education, storytelling, and videogaming.

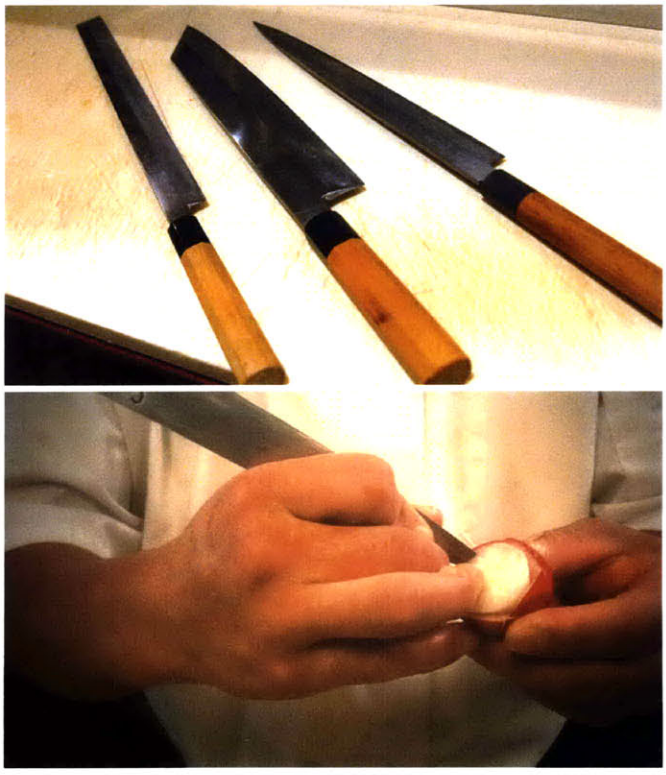


Figure 43 Custom made chef knives (top), chefs master their tools for better performance (bottom).

5.4 Evaluation Sessions

OnObject applications introduced above have been demonstrated to audience members and tried by participants on the following occasions.

5.4.1 Play Session with Preschool Children

In May 2010, two children were invited to play with OnObject with the facilitation by a fellow research assistant for 60 minutes. The children were 3- and 5-year-old sisters. Over a dozen tagged objects including stuffed animals and plastic imitation fruits were present, and more RFID tags were provided along with adhesive tape and a glue gun. Other play materials such as color pencils, blank and used papers and clay were provided (Figure 44).

Figure 44 Tagged toys were provided.

Soft Device

A more child-friendly OnObject device was provided, neoprene-covered to protect the children's hands from hard and sharp circuit boards, also simplifying the overall shape and only exposing necessary lights and button for better usability (Figure 45).

Figure 45 Soft device with elastic antenna band.

Recording Sound Into Objects

A 3-year-old child successfully recorded her own voice into a tagged lump of clay she sculpted, and then to a tagged banana following the instructions from the facilitator. The initial recording, from encountering the device (child asks "what is this?") to playing her recorded voice with grab gestures, took 27 seconds.; the second time with a tagged banana the Gesture-Response process took 11 seconds:

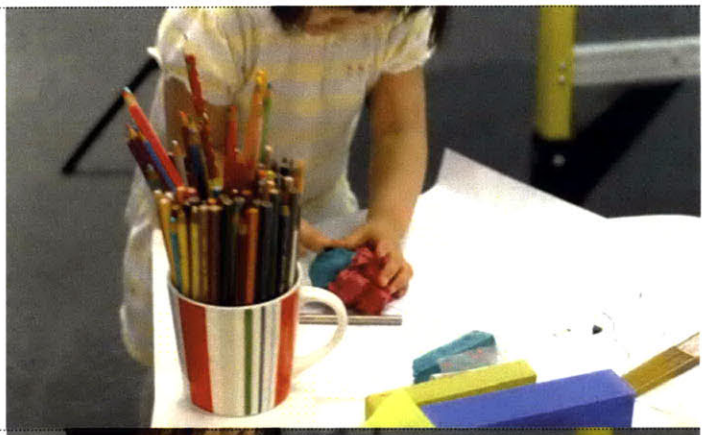
Table 12 shows the 3-year-old recording sound into the lump of clay, and Table 13 shows her recording sound into a tagged banana. Video documentation of these activities is available online [61].

Table 12

3-year-old recording sound into a lump of clay.

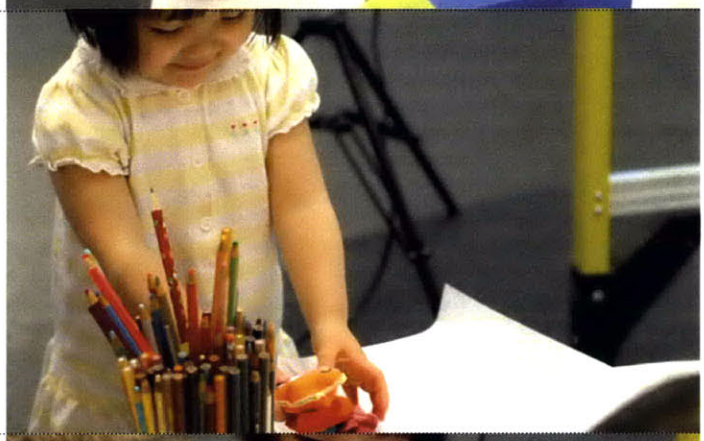
1 Tag

The child inserts a tag into a lump of clay. She sees the OnObject device facilitator is holding and asks "what is this?"



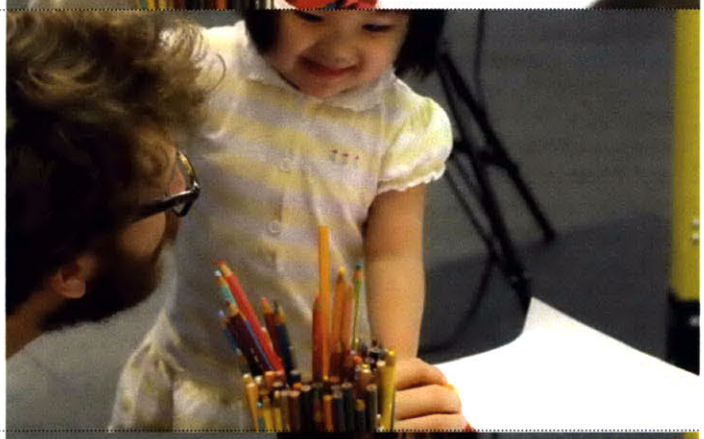
2 Gesture

Guided by the facilitator, the child grabs the device in her hand, touches the clay by the tag to hear a chime sound. She says "yeah."



3 Response

Facilitator presses the button and asks her to say something. The child hesitates, then says "this is not a snowman."



4 Play

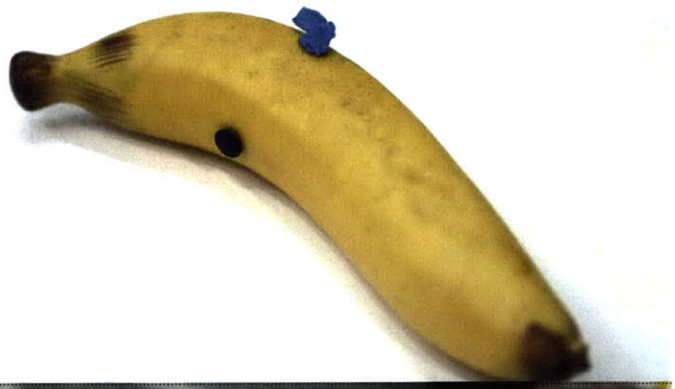
The child touches the clay to hear her voice, and giggles.

Table 13

3-year-old recording sound into a tagged banana.

1 Tag

Already tagged. Child says "this is a banana!"



2 Gesture

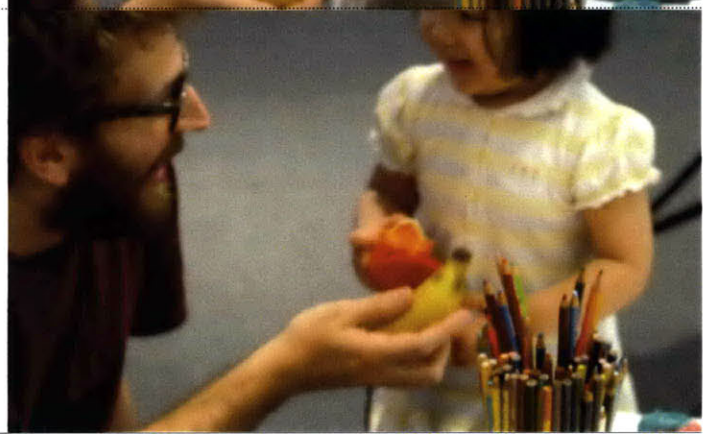
The child grabs the device and touches the banana by the black dot tag.

3 Response

The child presses the button herself and says "Banana--!" toward the banana this time.

4 Play

The child touches the banana again to hear her voice and giggles.



Incorporating Tags to Objects

Motivated by the presence of small RFID tags, easy to use masking tape, and a demonstration by the facilitator, children applied RFID tags to a variety of objects available around them.

Clay: 3-year-old child created ad-hoc toys by sculpting provided clay and inserting tags on them. While the lump of multicolor clay sculpture in Figure 46 is not recognizable by others, the child had particular ideas about its identity, and declared "this is not a snowman." By recording those words into the clay, she has specified its identity and has mentally changed their meaning to herself and other people around her.

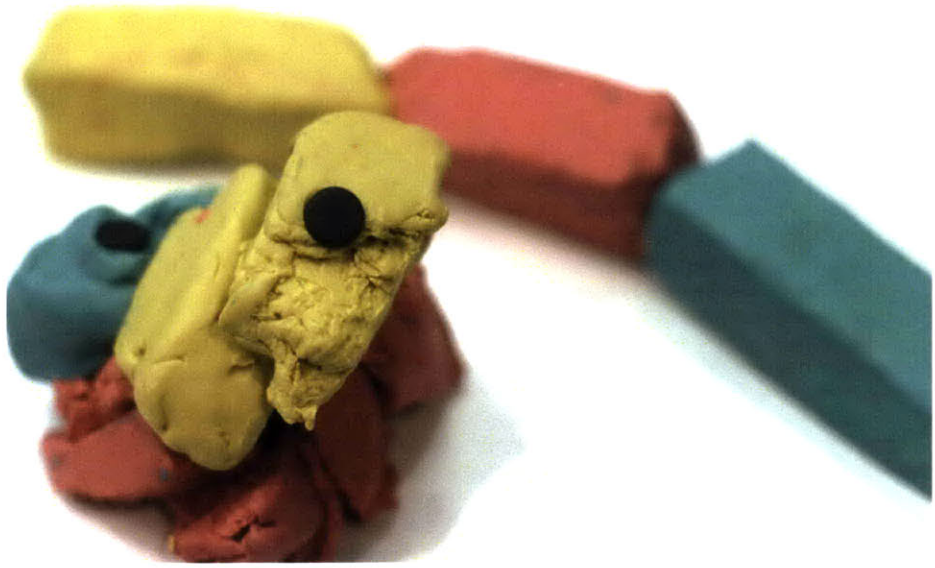


Figure 46 Lump of clay sculpted and tagged by children.

Umbrella and Pencil: Once they saw a demonstration of OnObject, both children applied tags to an umbrella and a pen and requested to put sound in them (Figure 47).

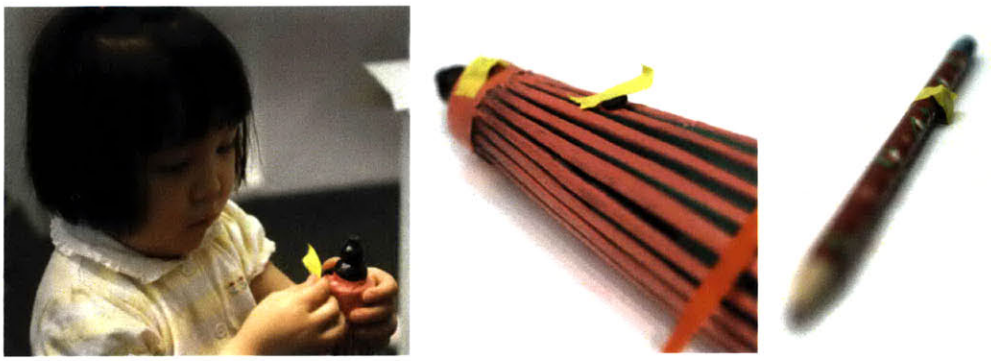


Figure 47 Child tapes two RFID tags to an umbrella and a pencil.

Drawings: The 5-year-old child made drawings on a blank sheets of paper, applied tags using yellow masking tape, and attempted to record sound to the tags (Figure 48). This suggests a novel application of OnObject: user-created interactive books where user can make their own drawings, record their voice into specific scenes or characters of the drawing, and play them by touching the spots on the drawing. This function has been previously implemented in Jabberstamp (2003) by Raffle et al., which requires a tablet input device to be placed beneath each drawing, a microphone-integrated tablet pen, plus a tablet eraser pen [55].

In contrast, interactive stories created by OnObject can be used in its more natural form of loose pieces of paper in any size or shape without an electronic device underneath. Also, the attached tags makes it clearer where sounds may be embedded, and children can simply remove or relocate each sound by re-taping the tags.

Figure 48 Drawing created and tagged by a 5-year-old child.

Roleplaying and Storytelling

Simple Q&A: Using the ability to program different sound for different gestures, the facilitator was able to have an engaging and rhythmic exchange with the children about animal sounds. Tags were glued to stuffed frog and pig, and the facilitator recorded "what does a frog say" into a frog when it's grabbed, and "ribbit" when it's shaken prior to the following sequence:

Table 14
Children have rhythmic exchange with a facilitator about animal sounds.

1 Play The facilitator grabs the frog to play his voice saying "What does a frog say?" The children respond: "Ribbit!"	
2 Play As the facilitator shakes the frog and it says "ribbit" in his voice too, the children start singing along: "Ribbit, Ribbit, Ribbit..."	
3 Gesture The facilitator says "What does a pig say?" then shakes the pig to hear default boink sound response.	
4 Response The facilitator presses button down and extends the pig to a child who responds "oink."	

Children were overall comfortable with the sequence of the interaction during the session and were able to incorporate it into their play activities. Real-time video documentation is available online [61].

Lion Play: With a scripted story application, the children repeatedly played a lion character story which includes dancing and fanfare (Figure 49). The 5-year-old was able to prompt the stuffed lion to say pre-recorded lines when grabbed and shaken.

Figure 49 The child grabs and shakes a tagged lion with OnObject device for roleplay.

5.4.2 Media Lab Sponsor Week Demonstration

OnObject ABC, Storytelling, and Swordplay were demonstrated to over 60 visitors of diverse background at the research open house in May 2010. More than 10 visitors participated in the demonstration to varying degrees by providing their own voice for recording, pressing the recording button, or trying grab gestures on a variety of objects. Many expressed surprise at the ability to incorporate regular objects for interactive play. Some found the prospect of user-created applications like interactive drawings interesting, and suggested other possibilities including a treasure hunt. Many visitors found the Swordplay application particularly amusing. The fact that one can enjoy Nintendo Wii-like interaction with their own belongings and copy-paste the setting from a sword to a pen surprised the audience. A visitor from consumer electronics industry commented "I think of the many possible applications."

Visitors familiar with the RFID technology noted that it was an unusual way of utilizing the particular technology. However, for those not familiar with RFID, the details were not considered important. Without necessarily knowing how RFID works, visitors easily accepted that the device detects the tags when touched. Instead of seen as an radio antenna technology with an effective range, visitors simply understood it as a near-touch detection.

Figure 50 Author giving swordplay demonstrations (left), tagged props in display (right).

5.4.3

Toy Prototyping Test

In order to test the OnObject Clay application, a 34-year-old female participant spent 60 minutes brainstorming and prototyping interactive toys in April 2010. Participant was a design professional who had not previously seen OnObject in action. She was asked to brainstorm as many toys as possible that will encourage children to become more physically active and immediately create functioning prototypes of those toys. This test was conducted partially in Wizard-of-Oz style, as the facilitator configured the mapping of tags, gestures, and output sound using KeyMapper text file according to the participant's decisions. However, once the mapping was complete, the resulting gestural interaction was functional and tested by the participant herself.

Procedure

The participant was given a set of prototyping "ingredients"—foam blocks, LEGO blocks, tubings, tapes, clay, cardboard, glue gun, and labeled RFID tags. She was then shown and provided with an OnObject device and 26 tags. The facilitator explained four gestures (grab, circle, swing, thrust) that can be attached to each tag, and four sound effects (water, chime, boink, kiai) that can be triggered by those gestures.

The participant described her ideas aloud as she assembled the ingredients (Figure 51-1), attached a tag to the assembled prototype (Figure 51-2), and stated to the facilitator the tag she used (labeled A through Z) and which gesture on the tag will trigger which sound effect. The facilitator then changed the configuration text file and executed the application. The participant tried the gestures and explained the toy idea to the facilitator (Figure 51-3), before moving onto the next idea (Figure 51-4).

Results

During the 60-minute section, the participant was able to brainstorm and demonstrate 9 ideas that respond to gesture triggers. The ideas included: X-shaped weapon, a cleaning sponge that makes noise when used in circular motion to clean windows, an arrow that is spun by one player and grabbed by another player pointed by the arrow, a cleaning mop thrust to make sound, and an aerobic and yoga device.

As the session progressed, the participant became comfortable with the features of the system and started to suggest ideas that were outside of original criteria yet she

felt were well-suited to OnObject. Three of the ideas involved using only the OnObject device without an actual toy in hand, designed to provide feedback for dancing and exercise activities or theatrical presentations. Two notable feedback from the participant are as follows:

Audio response enables diverse and engaging interaction: Even though the response was limited to short audio playback, the participant commented that it still afforded many scenarios where the interaction was fun and new, and that by simply adding audio feedback, mundane activities like exercising or cleaning could be made significantly more engaging.

Inaccurate gestures can be a useful part of the interaction: Toward the end of the session, the participant suggested ideas that trigger sound response to any known gesture or general level of activities. She suggested that when the user is allowed to make imprecise and freeform gestures that still trigger responses, the unexpectedness in fact encourages more activities.

Figure 51 Participant brainstorms toys, sculpt and tag them, and trigger audio response to gestures.

5.5 Analysis and Reflections

5.5.1 Novelty and Appeal

How interesting and exciting is OnObject?

Overall, those who used OnObject, saw live demonstrations, or saw the videos have expressed that it was new and different. The verbal comments received from the audience and peer-reviewed conference submission indicates that the notion of "programming" real objects, incorporating routine objects one by one as you go and making them interactive, copy-paste propagation, and the easiness that suits a complete novice user or a child were perceived novel to HCI experts and end users.

The most common excitement was regarding the possibility of creating interactive story plays with the audience's children and playing video games using everyday objects. "I can imagine how excited my girls would be to play with this," said an audience member who attended the OnObject presentation at the SEGD Conference in June 2010 [56].

Those who have prior experience using or developing for the RFID technology recognized that OnObject applied an existing technology to new purposes other than shipping, supply chain tracking, transportation or activity tracking. A visitor from a consumer electronics company wrote OnObject showed an "interesting use of RFID technology."

However, for end users, children, designers, or product development experts who were not so technology-centric, what seems to have impressed them the most was the ability to give a virtual meaning to the mundane everyday objects and perceive them differently, similar to the notion of a magician turning an inanimate object into an active character.

For instance, the 3-year-old child from the play session learned to record sound into a tagged banana so that when she touches it she hears "banana" in her voice. This appeared to give her the idea that she can make everything tell her their names. She immediately turned to a bundle of pencils and went, "let's try this..." Then she said "pencils!" holding the device toward the pencils, as if she is using the device as a magic wand (Figure 52). While this is not the correct way to use the device as the device needs to be in a much shorter distance to the object, in her mind, the inanimate objects had become capable of speaking, or rather, she had become capable of making them speak.

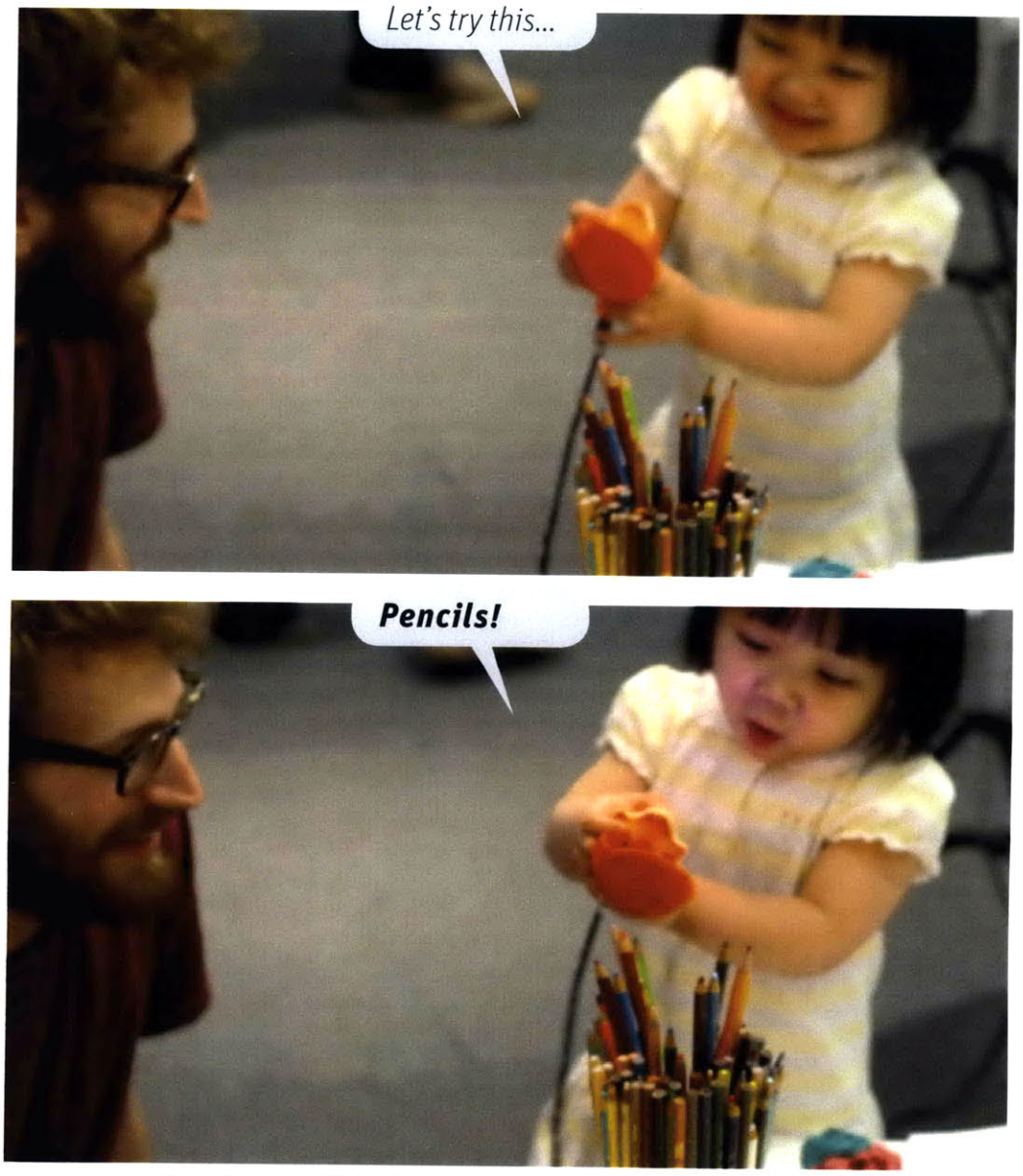


Figure 52 3-year-old child wants to record sound into pencils from a distance.

Echoing this observation, a consumer electronics industry insider also noted in response to the Sponsor Week live demo, "using technology connected with the human imagination can be a powerful tool and communicator" for children and adults alike, nothing that part of her interest in OnObject was "my 'inner kid' bragging about the newest toy."

5.5.2 Clarity and Usability

Is the interaction clear for the user and the audience?

Tags

The 9mm diameter plastic RFID tags proved easy to apply to a variety of objects including paper, clay, stuffed toys, plastic and wooden toys, pencils, and umbrellas, even in the hands of a 3-year-old. While a more durable adhesive like the glue gun was useful for many applications, it was also important to have easier and quicker tools like masking tape to encourage experiment without committing to a particular design of the new gesture object interface.

Going forward, a potential problem with these tags when used with children is the choking hazard: the Child Safety Protection Act (CSPA) prohibits small parts to be used for children under 3 years of age [57]. However, this problem is largely avoidable by targeting audience 3 years or older and adding labels as mandated by the government regulations.

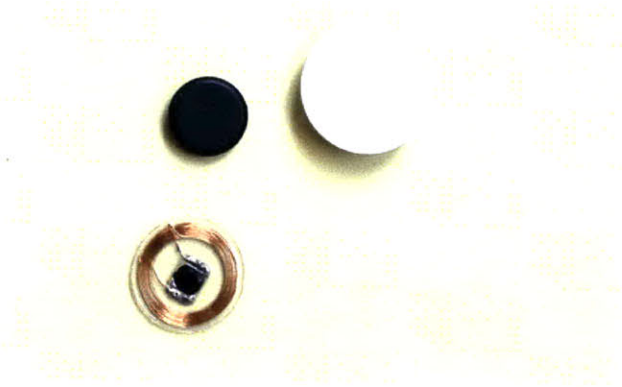


Figure 53 The 9mm diameter laundry tags mainly used for evaluations (top left).

Ergonomics of the Device

Some of the performance errors with gesture recognition is closely related to how the device is worn and constructed. While the toy test participant, play session facilitator and children were able to wear the device and eventually make the grab, shake, circle, swing, thrust work for them, there are obvious points of improvement:

Wires: The current version of the device is connected to the computer via 6-wire serial-to-USB cable and two wires for the microphone. Making the device completely wireless and easy to recharge will take considerable amount of effort. As an interim solution, however, the USB cable can be replaced with a Bluetooth module for wireless communication and two power wires. By removing the heavy USB cable connected to the device, the accelerometer data is expected to have less noise.

Orientation: Because gesture recognition relies on the accelerometer data, for many applications it is important the device is worn in the right orientation. Future designs and version of device will have to make the orientation clearer.

Tag-Gesture-Response Flow

Interaction Sequence: All three participants and a facilitator from the play session and toy prototyping test were able to follow the programming procedure without much difficulties. Most who watched the video demonstration of OnObject ABC were able to conclude that the recorded sound is associated with the last performed gesture. When the 3-year-old child from the play session was led by the facilitator the first time she recorded her voice into clay, she spoke toward the facilitator. However, after seeing the result of the first programming she made her next recording explicitly toward the tagged banana, indicating that she understood the logic of the interaction sequence.

Feedback: With all participants and audience, the default sound feedback upon grab played an important role. Because there is no display in the device and the scenarios incorporate situated objects, users often do not expect digital or computational experience when they first encounter OnObject. When they hear the chime sound when you touch or grab an object, it is the first indication that an interactive experience is beginning. As this is unexpected, some participant expressed excitement, such as the 3-year-old child saying "yeah" when she first heard the chime. Although there is an indicator light on the device for tag detection, future design of the device can utilize the light feedback and make it more visible and meaningful by incorporating color change or placing the light differently.

5.5.3

Feature Requests

What more do people want to see?

Multi-User Interaction

In three separate presentations, audience members asked for an application where more than one user are wearing the device.

The Ceiling

Despite the participants' satisfaction regarding audio response, the technical simplicity of the response has been named a point of improvement by HCI experts. What is the ceiling of the OnObject solution, i.e., can we built more complex applications than simply TUI-stories where the only interactive part is audio playback? In order to address this issue, we are considering three strategies.

- **Multimodal Output:** Employ other modalities of output, such as visual output as in the Swordplay application or Internet-connected data output such as posting an entry to a website.
- **Multi-User Interaction:** Consider applications that involve two or more users wearing the device.
- **Programmable Narrative:** Make audio-only narrative construction more complex and grammatical, by providing abilities to add conditionals, sequential contingencies, or timeout functions.

See Section 6.2 for a list of proposed future improvement to OnObject.

5.5.4

Performance Dimension

What interesting aspect has come to attention?

Prototyping as a Performance

In The Art of Innovation, Kelly noted [54] "Years of experience have taught us that prototyping is also part performance, that if the act isn't well orchestrated and substantial, the audience gets antsy." During the toy prototyping test, the participant exhibited similar tendency where she was not only explaining the concept, but giving an energetic and entertaining rendition of the toy usage. As OnObject enables much faster turnaround of simple interactive prototyping, it can make the prototyping session into a more enjoyable group activity and increase the energy level that Kelly notes as crucial to successful brainstorms.

Usage and Demonstration as a Performance

As demonstrations for live audience or video shooting progressed, it was noted the body gestures had more and more "flare" and theatrical quality. Instead of a normal grabbing of an object, the demo grab gesture often resembled a slow-motion gesture, and the shake gesture was accompanied by a low-pitch voiceover. Similarly, the 3-year-old child expressed a level of authority when she commanded the pencils to speak.

These tendencies not only applied to the trigger gestures, but also to the transition between them. A remarkable example is the hand transition between copying and pasting: After pressing the button on the sword, the hand released the sword and traveled to grab the Sharpie pen in a theatrical manner, with a verbal explanation "the definition of the sword is now in my hand, and I am transferring it to the pen..." Although these transitional movements are not a part of OnObject's functional requirements, these are still part of what Kendon and Goffman dubbed "foreground action" and "main-line" attentional track [58].

As the users of OnObject become familiar with the use, they seem to establish a flow of foreground actions and embellish them to convey specific sentiments or meaning, whether it is command (3-year-old child), transfer (copy-paste), or excitement (toy test), with an external audience in mind – adding a performance dimension to the user experience.

Power of User-Defined Mapping

A potential explanation of this transition from a "user" to a "performer" can be found in the Bowl platform study by Martinussen et al. [40] Their research found that "homemade" tokens mapped to user-created content proved to be more engaging than pre-mapped tokens to a 2-year-old child. While the child successfully used and explored pre-mapped tokens with clear association, when it came to homemade tokens that she found and mapped to content created with her parent, the child turned into a performer of a sort, enthusiastically presenting the tokens and associated memories to visitors.

The anecdotal evidences from this study and OnObject experience support a hypothesis that when the user defines the mapping between the object, their actions and the system's responses, users "buy into the interface" – they find the interface believable because they defined it. As they are already persuaded, users can skip the usual steps necessary to validate the interface and instead focus on persuading others. A more ecologically generalizable study would be necessary to test this hypothesis.

Chapter 6

Conclusion

Proposing a scalable, simple, yet engaging method to incorporating situated objects in interactive experiences, OnObject claims programming of physical objects as a new area of Tangible User Interfaces.

Figure 54
Micky Mouse from Disney's Fantasia casts spell to make inanimate objects come alive: An inspiration for OnObject.

6.1 Defining Contributions

The design of OnObject has introduced a novel class of Human-Computer Interaction: gestural programming of situated physical objects. The novelty and merits of OnObject approach to programming includes the following:

Embodied

Instead of writing code, the user's hand is the primary medium for associating between object and subsequent programming, demonstrating gestures, and initiating programming commands like recording and copy-paste.

Body-Object-Centric

A small sensing platform has been developed for user's hand, which leads users to focus on the point of interation and enables mobile and pedestrian use.

Scalable

Rapid and low-commitment tagging is applicable to a broad set of real situated objects, and encourages open exploration.

Grammatical

OnObject introduced a simple grammar of programming, Tag-Gesture-Reponse flow. Copy-pasting programming between objects in the Swordplay application also demonstrated that the grammar may be further expanded for more rich and complex interactions.

User-Defined Mapping

Instead of being exposed to low-level development tasks, the user is led to focus on creating an enjoyable mapping between gestures and audio responses. As the user defines the mapping between a particular object, gesture, and response, the interaction becomes more believable to the user.

Expressive

As the user is the designer of the gestural applications, she instantly becomes an active presenter of the concept to others rather than a mere passive "user" of the system. The user tends to bring theatrical quality and embelishment to her gestures as she performs the interaction for others.

6.2 Future Work

As OnObject addresses a novel area of tangible and embodied programming, only a small set of applications and possibilites have been explored in this thesis.

Multimodal Output

Employ other modalities of output, such as visual output as in the Swordplay application or Internet-connected data output such as posting an entry to a website.

Multi-User Interaction

Applications that involve two or more users wearing the device.

Programmable Narrative

Make audio-only narrative construction more complex and grammatical, by providing the ability to add conditionals, sequential contingencies, or timeout functions.

Refine Tag-Gesture-Response Flow

For example, after user demonstrates the trigger gesture, if the gesture is not properly recognized, this step can invoke a training flow until the user learns the gesture.

Design Exploration of the Device

Smaller, user friendly form factor, and better integration of control components with the programming flow. Implement wireless devices to free users from the bound of the nearby computer.

Tag Dispenser and Inspector

Rapid application of tags to physical objects, to accentuate the convenience; A method to inspect the tag and view programming associated with a given tag.

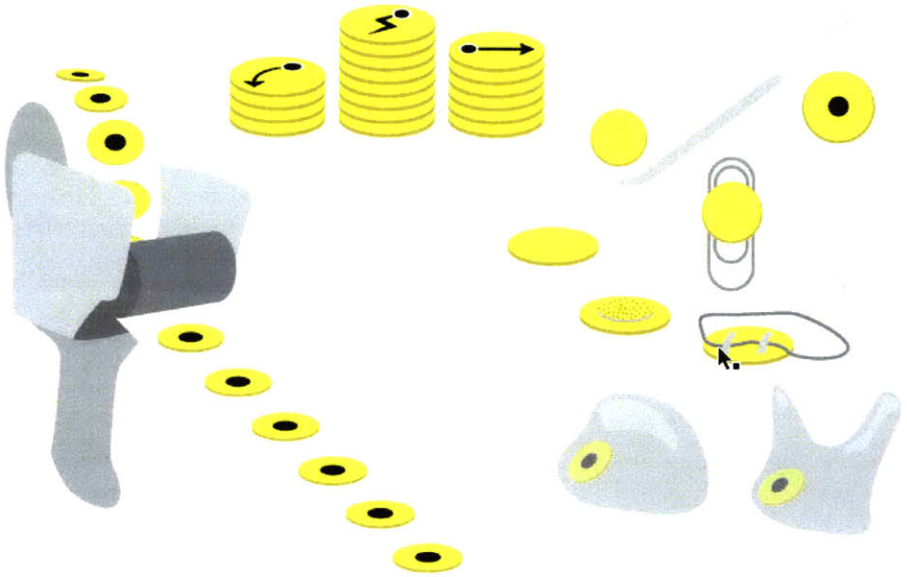
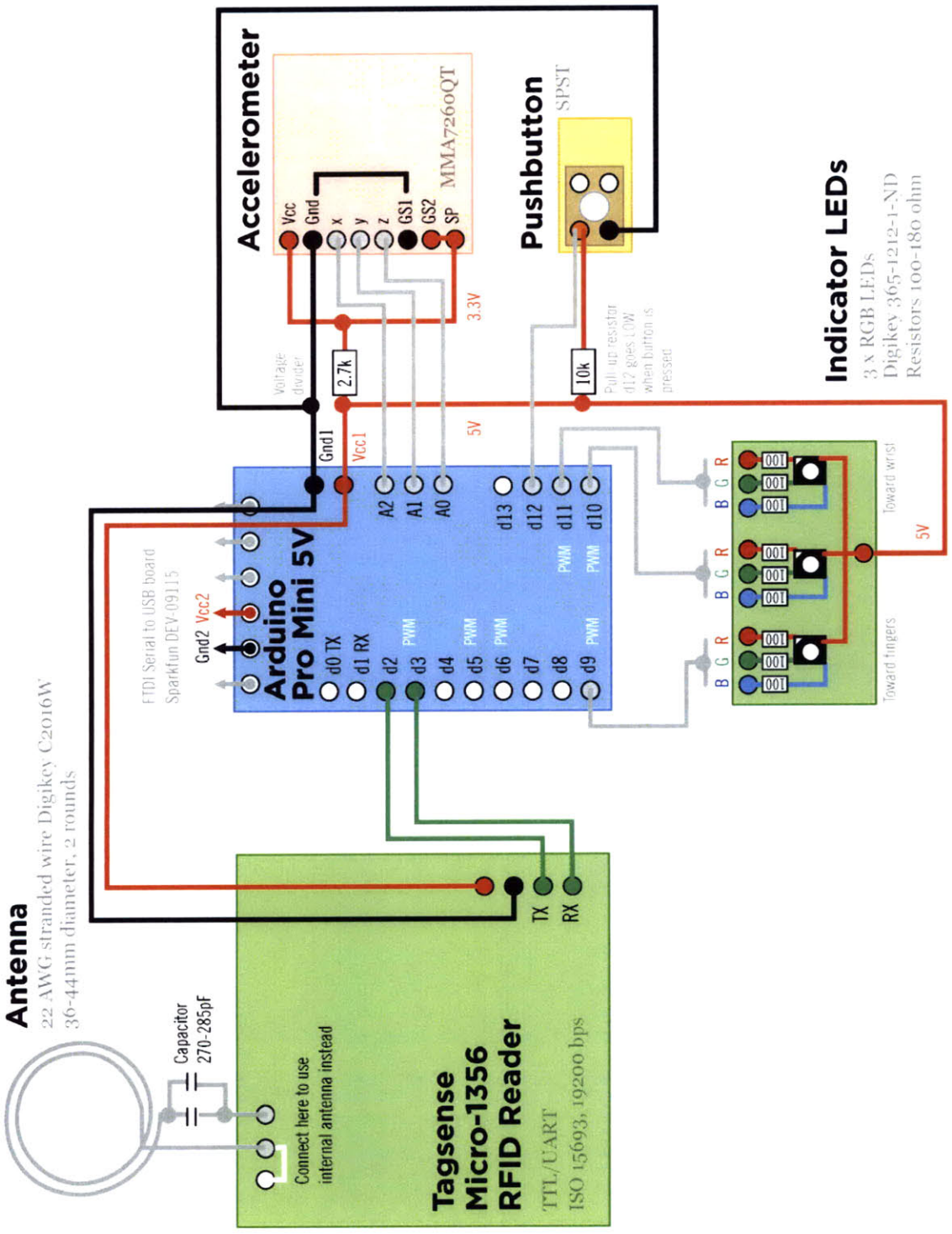


Figure 5 Tag dispenser concept sketch.

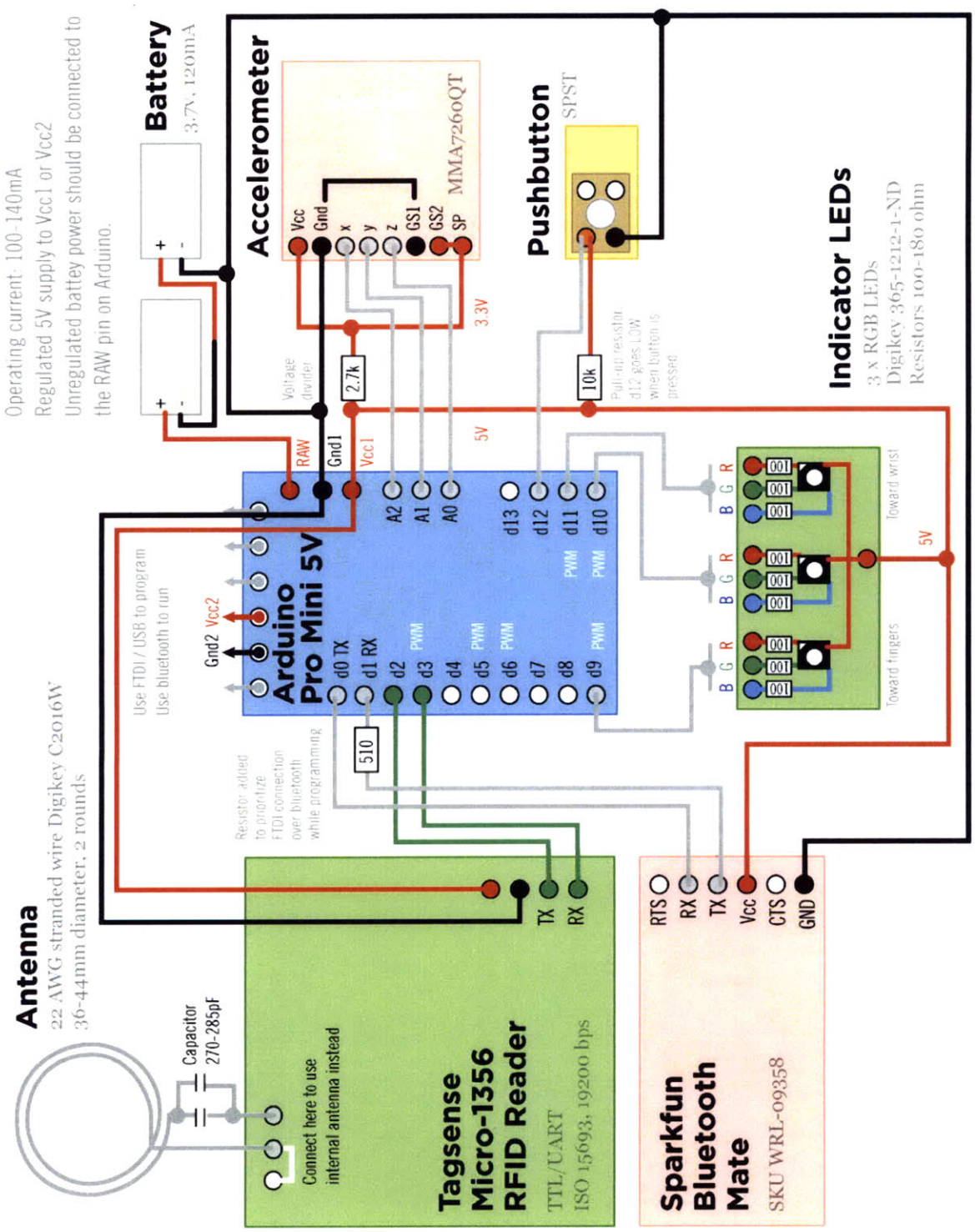
Although Curlybot [64] and Topobo [18] pioneered kinetic programming by demonstration and tabletop systems like Lumino [65] and Sensetable [66] leverage physical manipulation of specially made puck or cube objects, the ability to program real-world situated objects has not been explored. OnObject highlights tangible and embodied programming of situated objects as a potential future area of Human-Computer Interaction.

Appendices

A. Hardware Diagram (Wired)



A. Hardware Diagram (Wireless)



B. Device Firmware Arduino Code

```
/*
OnObject Arduino Device code
for Tagsense Micro-1356 RFID reader and MMA7260QT accelerometer
By Keywon Chung, Chris Merrill 2009-2010
Prints out 345 xxx yyy zzz 678 0000000000000000 999 0

 -Arduino gets 3-axis accelerometer data from analog pins.
 -Arduino gets 16-byte RFID tag data via NewSoftSerial.
  See http://arduiniana.org/libraries/NewSoftSerial/
 -Arduino sends the accel and RFID data to Python for gesture recognitio
  via hardware serial.
 -When a tag is detected, one LED is turned on.
 -When user presses a button on the device, another LED is turned on.
 -When a known gesture is detected, the last LED is turned on.
 -Applications can also send a string to the Python gesture server
  via socket server/client connection
  http://mostrey.be/python_socket_server_for_flash
 -Python then sends the string to Arduino via hardware serial
 (PySerial library for Python) to control the LEDs on the hardware.
*/

#include <NewSoftSerial.h>

int accelmode = 2;
// accelmode 0: print in serial monitor
// accelmode 1: plot graph in processing
// accelmode 2: send to python recognizer

#define RX_PIN 2 // digital pin: software serial rx pin
#define TX_PIN 3  // digital pin: software serial tx pin
#define X_PIN 2 // analog pins A2 - A0
#define Y_PIN 1
#define Z_PIN 0
#define LED_PIN1 9 // digital pins with PWM, towards finger
#define LED_PIN2 10
#define LED_PIN3 11 // toward wrist
#define BTN_PIN 12

#define RFID_DATA_LEN 16 // tag ID is "*" + 16 bytes long
int rfid_bytecount = 0; // increment within DATA_LEN
int tag_present = 0;
#define APP_DATA_LEN 2
int app_bytecount = 0; // increment within FLASH_LEN
int btn_status = 0;

// Baud rates:
// RFID reader to Arduino via software serial: 19200
// Accelerometer + RFID reader data on Arduino to the computer
// via hardware serial: 9600
// Pins:
// digital 2/3: soft serial rx/tx
NewSoftSerial RFID = NewSoftSerial(RX_PIN,TX_PIN);
unsigned long time;

// data storage
```

```
char rfid_data[RFID_DATA_LEN]; // string, to store tag ID
char app_data[APP_DATA_LEN]; // string, to store messages coming from applications like Flash
char old_tag[RFID_DATA_LEN]; // previous tag ID

// led values 0-255, time delay (reverse for white LED later: this is for RGB led)
int led_offvalue = 255; // approximation of forward voltage drop, where the LED actually
appears off (80)
int led_fullvalue = 80;
int led_increment = 1; // how much to dim each time
// don't turn off LED until you see tag absent N consecutive times: ignore little glitches
int tag_offcount = 0;
int tag_absent_threshold = 2;

// map led functions: tag present, gesture found, etc.
int LED_TAG = LED_PIN1;

void setup() {
  RFID.begin(19200);
  Serial.begin(9600);
  pinMode(RX_PIN, INPUT);
  pinMode(TX_PIN, OUTPUT);
  Serial.println("Start");
  // configure RFID reader
  RFID.print("p3\r\n"); // check for ISO15693: reader should print "p" back
  Serial.print("p3\r\n");
  delay(100);
  RFID.print("d\r\n"); // enable continuous data streaming: reader should print "d" back
  Serial.print("d\r\n");
  delay(100);
  RFID.print("k\r\n"); // enable continuous autoscan: reader should print "k" back
  Serial.print("k\r\n");
  delay(100);
  RFID.print("S\r\n"); // enable continuous autoscan: reader should print "k" back
  Serial.print("S\r\n");
  delay(100);
  // to print reader's response:
  // char readerByte = RFID.read();
  // Serial.write(readerByte);

  // LED pins to be turns on based on Flash commands: this pinmode is not necessary.
  pinMode(LED_PIN1, OUTPUT);
  pinMode(LED_PIN2, OUTPUT);
  pinMode(LED_PIN3, OUTPUT);
  // turn off LEDs
  analogWrite(LED_PIN1, led_offvalue);
  analogWrite(LED_PIN2, led_offvalue);
  analogWrite(LED_PIN3, led_offvalue);

}
```

```
void loop() {

  // Send data continuously to the computer/python via serial; flush is crucial
  // Empty initial RFID ID storage and App message storage
  clear_rfid_data();
  clear_app_data();
  RFID.flush();

  // ----------------------------------
  // Frame incoming bytes, get accel > rfid data > look for app messages

  if (accelmode == 0) { // print in serial monitor
    Serial.print(3);
    Serial.print(4);
    Serial.print(5);
    Serial.print("  ");
    Serial.print(analogRead(X_PIN));
    Serial.print("  ");
    Serial.print(analogRead(Y_PIN));
    Serial.print("  ");
    Serial.print(analogRead(Z_PIN));
    Serial.print("  ");
    Serial.print(6);
    Serial.print(7);
    Serial.print(8);
    Serial.print("  ");
    print_tag_id();
    Serial.print("  ");
    Serial.print(9);
    Serial.print(9);
    Serial.print(9);
    Serial.print("  ");
    print_button_status();
    Serial.println();
    /* // approximate sampling rate: 19.2-20 Hz
     time = millis();
     Serial.print(" millis- ");
     Serial.println(time);
     */
  }
  else if (accelmode == 1 || 2) {
    // send to processing to plot graphs
    // send to python to run applications
    Serial.write(3);
    Serial.write(4);
    Serial.write(5);
    Serial.write(analogRead(X_PIN)/4 - 10); // because byte only goes up to 256
    Serial.write(analogRead(Y_PIN)/4 - 10); // value will be multiplied by 4
    Serial.write(analogRead(Z_PIN)/4 - 10); // again in python.
    Serial.write(6);
    Serial.write(7);
    Serial.write(8);
    delay(10);
    print_tag_id();
    Serial.write(9);
    Serial.write(9);
    Serial.write(9);
    print_button_status();
  }
```

```
  if (accelmode == 2) {
    app_to_arduino();
  }

} // end of loop


void print_button_status() {
  int i = 0;
  // programming button pressed or not
  if (digitalRead(BTN_PIN) == LOW) {
    analogWrite(LED_PIN2, led_fullvalue);
      i = 1;
  }
  else {
    analogWrite(LED_PIN2, led_offvalue);
    i = 0;
  }

  if (accelmode == 0) {
    Serial.print(i);
  }
  else {
    Serial.write(i);
  }
}


void print_tag_id() {
  // ---------------------------------- Read RFID Data
  // wait for "*"
  char in = (char)RFID.read();

  // is it garbage, or is there a tag ID starting with a "*"?
  if (in == '*') {
    // beginning of * + 16 bytes tag ID
    // once you detect a "*", store the following 16 bytes of tag ID
    while (rfid_bytecount < RFID_DATA_LEN) {
      in = (char)RFID.read();
      // bytes after the "*"
      rfid_data[rfid_bytecount] = in;
      rfid_bytecount++;
    }
    // indicator LED on
    analogWrite(LED_TAG, led_fullvalue);
    tag_offcount = 0;
  }

  else { // no "*" found: make tag id 16 "0"s
    clear_rfid_data();
    // ignore little glitches, don't turn LED off
    // until you see tag absent for N consecutive times
    if (tag_offcount > tag_absent_threshold) {
      analogWrite(LED_TAG, led_offvalue);
    }
    tag_offcount++;
  }
```

```
  // Check to make sure we didn't recieve junk data
  if (!check_string(rfid_data,RFID_DATA_LEN)) {
    copy_array(old_tag,rfid_data,RFID_DATA_LEN);
  }
  // finished storing all 16 bytes
  if (rfid_bytecount >= RFID_DATA_LEN && rfid_data[1] != '*') {
    // send 16 byte string tag ID to Python
    for (char i = 0; i < RFID_DATA_LEN; i++) {
      Serial.write(rfid_data[i]);
    }
    copy_array(rfid_data,old_tag,RFID_DATA_LEN);
    // re-initialize the variables
    rfid_bytecount = 0;
  }
  else { // no complete tag ID found: still send 16 "0"s

    for (char i = 0; i < RFID_DATA_LEN; i++) {
      Serial.write(rfid_data[i]);
    }
    copy_array(rfid_data,old_tag,RFID_DATA_LEN);
    // re-initialize the variables
    rfid_bytecount = 0;
  }

}

// for debugging
int is_empty() {
  for(int i = 0; i < RFID_DATA_LEN; i++) {
    if(rfid_data[i] != '0') {
      return 0;
    }
  }
  return 1;
}
// Copy the contents of one array into another
void copy_array(char * orig, char * into, int size) {
  for (int i = 0; i < size; i++) {
    into[i] = orig[i];
  }
}

// Check to see if there are non-hex characters in the string
boolean check_string(char * str, int size) {
  char c = 0;
  for (int i =0; i < size; i++) {
    c = str[i];
    if (!((c < 58 && c > 47) || (c < 123 && c > 96))) return false;
  }
  return true;
}

void clear_rfid_data() {
  for (char i = 0; i < RFID_DATA_LEN; i++) {
    rfid_data[i] = '0';
  }
}
```

```
void clear_app_data() {
  for (char i = 0; i < APP_DATA_LEN; i++) {
    app_data[i] = ' ';
  }
}

////////////////////////////////// Serial.print can be watched from serproxy
void app_to_arduino () {
  // ---------------------------------- FLASH/PYTHON to LED
  if (Serial.available() > 0) {
    char in;
    app_bytecount = 0;
    in = (char)Serial.read();
    // is it garbage, or is there a flash string starting with a "_"?
    if (in == '_') {
      // beginning of _ + 1 byte string from app
      // once you detect a "_", store the following 1 byte of string
      while (app_bytecount < APP_DATA_LEN) {
        app_bytecount++;
        in = (char)Serial.read();
        // app_data[app_bytecount] = in;
        if (app_bytecount == 1) {
          int msgNum = int(in) - 48; // ascii to decimal
          clear_app_data();
          if (msgNum == 5) {
            // app specific LED feedback
            // ... resume processing gestures and tags
          }
          else if (msgNum == 4) {
            // known gesture started
            analogWrite(LED_PIN3, led_fullvalue);
          }
          else if (msgNum == 3) {
            // known gesture ended
            analogWrite(LED_PIN3, led_offvalue);
          }
        }  // end of if bytecount
      } // end of while
    } // end of if in == _
  } // end of if serial available
}
```

C. Basic ABC/Storytelling Application Python Code

```
############################################################
# demo settings: froggy, kangaroo, bunny, piggy, aardvark, bird on table, lion is away

# froggy, kangaroo, and lion tags have hardcoded response, the other tags behave same as
audio06
    # froggy grab: what's your name?
    # user: press button "___" release button
    # froggy shake: hello ___, let's go jump with the kangaroo!
    # kangaroo grab: jump with me!
    # kangaroo shake: boink [kangaroo and froggy jump together]
    # lion grab: roar, what is happening?
    # lion thrust: let's play some music!
    # cup grab: fanfare [put it on the aardvark's mouth]

# comp_recognizer_st.py with shake_recognizer_weak.py
    # shake, swing, thrust are interchangeable. shake is weak

# to run: python animal.py comp tag
    # reco mode shake = only detects grab, release, shake
    # reco mode dt = detects grab, release, circles, swing, thrust
    # reco mode comp = detects grab, release, shake, swing, thrust
    #    (circle = disabled from comp_recognizer)
    # reco mode all = not implemented yet
    # app mode tag = tag gestures only
    # app mode ghost = empty handed gesture triggers response for last real tag
    # app mode empty = empty handed gesture gets its own trigger and response
# press Control Z to exit
############################################################
import sys

from optparse import OptionParser

import logging
import threading
import datetime

import simplejson
from recognizer import *
from shake_recognizer_weak import *
from dt_recognizer import DT_GESTURE
from dt_segmented import *
from comp_recognizer_st import *
import pyaudio
import wave
import sys

BASELINE_SAMPLESIZE = 20

#SERIAL_PORT = '/dev/tty.usbserial-A600aips'
#SERIAL_PORT = '/dev/tty.usbserial-A600aipH'
#SERIAL_PORT = '/dev/tty.usbserial-A600aims'
#SERIAL_PORT = '/dev/tty.usbserial-A600aipu'
#SERIAL_PORT = '/dev/tty.usbserial-A6008i1I'
#SERIAL_PORT = '/dev/tty.usbserial-FTESJFZW'
#SERIAL_PORT = '/dev/tty.usbserial-FTESJDZ6'
SERIAL_PORT = '/dev/tty.FireFly-2796-SPP' # Bluetooth Mate
```

```
SERIAL_BAUD_RATE = 9600
tag = None
prev_realtag = None # except for 0000000000s
tag_gesture_response_dict = defaultdict(dict)
chunk = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
FROG_TAG = 'e00401003da512c4' # H
KANGAROO_TAG = 'e00401003da51421' # U
LION_TAG = 'e00401003da52d74' # N

LOG_FILE = 'gesture_sendkeys.log'

logging.basicConfig(filename=LOG_FILE, level=logging.DEBUG)

class AsyncRecorder(threading.Thread):
    IDLE = 0
    START_RECORDING = 1
    RECORDING = 2
    STOP_RECORDING = 3

    def __init__(self):
        threading.Thread.__init__(self)
        self.read_sensor = True
        self.sound_files = ["chime.wav"]
        self.state = self.IDLE
        self.record_obj = None
        self.record_data = []
        self.record_stream = None
        self.wait_event = threading.Event()
        self.start()

    def run(self):
        while True:
            self.wait_event.wait() # puts run() on hold, not taking up cpu
            print '.'
            if self.state == self.START_RECORDING:
                self.record_obj = pyaudio.PyAudio() # repeated
                self.record_stream = self.record_obj.open(format=FORMAT,
                                                          channels=CHANNELS,
                                                          rate=RATE,
                                                          input=True,
                                                          frames_per_buffer=chunk)
                self.record_data = []
                self.state = self.RECORDING
            elif self.state == self.RECORDING:
                data = self.record_stream.read(chunk)
                self.record_data.append(data)
            elif self.state == self.STOP_RECORDING:
                self.record_stream.close()
                self.record_obj.terminate()

                # write data to WAVE file
                data = ''.join(self.record_data)
                t = datetime.datetime.now()
                filename = str(t) + ".wav"
```

```
                self.sound_files.append(filename)
                filepath = "./sound_files/" + filename
                wf = wave.open(filepath, 'wb')
                wf.setnchannels(CHANNELS)
                wf.setsampwidth(self.record_obj.get_sample_size(FORMAT))
                wf.setframerate(RATE)
                wf.writeframes(data)
                wf.close()
                self.state = self.IDLE
                self.wait_event.clear()

    def start_recording(self):
        assert self.state == self.IDLE, 'Expected IDLE, got %s' % self.state
        self.state = self.START_RECORDING
        self.wait_event.set() # resumes run()
        print "* recording"

    def stop_recording(self):
        print "* done recording"
        assert self.state == self.RECORDING, 'Expected RECORDING, got %s' % self.state
        self.state = self.STOP_RECORDING
        while True:
            if self.state == self.IDLE:
                return self.sound_files[-1]

    def exit_program(self):
        self.join()

    def _play_sound(self, filename):
        self.read_sensor = False
        filepath = './sound_files/'+filename
        wf_play = wave.open(filepath, 'rb')

        play_obj = pyaudio.PyAudio()

        # open stream
        play_stream = play_obj.open(format =
                        play_obj.get_format_from_width(wf_play.getsampwidth()),
                        channels = wf_play.getnchannels(),
                        rate = wf_play.getframerate(),
                        output = True)

        # read data
        data = wf_play.readframes(chunk)
        #import pdb; pdb.set_trace()

        # play stream
        while data != '':
            play_stream.write(data)
            data = wf_play.readframes(chunk)

        play_stream.close()
        play_obj.terminate()
        self.read_sensor = True

class APP_MODE(Enumeration):
    TAG = 'tag'
```

```
    GHOST = 'ghost'
    EMPTY = 'empty'

class RECO_MODE(Enumeration):
    COMP = 'comp'
    SHAKE = 'shake'
    DT = 'dt'
    ALL = 'all'

class AudioApp(object):
    def __init__(self, reco_mode, app_mode):
        self.recorder = AsyncRecorder()
        self.last_sound = 'boink.wav'
        self.app_mode = app_mode
        self.reco_mode = reco_mode
        self.last_gesture = GESTURE.NONE
        self.last_nonzero_tag = NO_TAG

        if self.reco_mode == RECO_MODE.COMP:
            print "reco mode comp = grab, swing, shake"
        elif self.reco_mode == RECO_MODE.SHAKE:
            print "reco mode shake = grab and shake"
        elif self.reco_mode == RECO_MODE.DT:
            print "reco mode dt = grab, swing, circle, thrust"

        if self.app_mode == APP_MODE.TAG:
            print "app mode tag = tag gestures only"
        elif self.app_mode == APP_MODE.GHOST:
            print "app mode ghost = empty handed gesture triggers response for last real tag"
        elif self.app_mode == APP_MODE.EMPTY:
            print "app mode empty = empty handed gesture gets its own trigger and response"


    def on_button_down(self, tag):
        print "button down"
        self.recorder.read_sensor = False
        self.recorder.start_recording()

    def on_button_up(self, tag):
        print "button up"
        self.last_sound = self.recorder.stop_recording()
        print 'sound attached to', self.last_nonzero_tag
        tag_gesture_response_dict[self.last_nonzero_tag][self.last_gesture] = self.last_sound
        print "attached to tag", self.last_nonzero_tag, "on", self.last_gesture
        self.recorder.read_sensor = True

    def on_tag_grab(self, tag):
        print 'g'
        self.last_nonzero_tag = tag
        self.last_gesture = GESTURE.GRAB
        if tag == FROG_TAG:
                self.recorder._play_sound('froggy_iam.wav')
                self.recorder._play_sound('froggy_whatsyourname.wav')
        elif tag == KANGAROO_TAG:
                self.recorder._play_sound('kangaroo_hello.wav')
                self.recorder._play_sound(self.last_sound)
                self.recorder._play_sound('kangaroo_jumpwithme.wav')
        elif tag == LION_TAG:
```

```
                self.recorder._play_sound('lion_happening.wav')
        elif tag == 'e00401003da51c97':
            self.recorder._play_sound('2010-05-16 21:09:31.774802.wav') # p is for
        elif tag in tag_gesture_response_dict:
            self.on_gesture(tag, GESTURE.GRAB)
        else:
            tag_gesture_response_dict[tag][GESTURE.GRAB] = 'chime.wav'
            self.recorder._play_sound('chime.wav')

    def on_tag_release(self, tag):
        pass
        # call on_gesture with "release"

    def on_gesture(self, tag, gesture):
        if tag != NO_TAG:
            print gesture
        self.last_gesture = gesture
        if tag == FROG_TAG:
            if gesture != GESTURE.NONE:
                self.recorder._play_sound('froggy_hello.wav')
                self.recorder._play_sound(self.last_sound)
                self.recorder._play_sound('froggy_lets.wav')
        elif tag == KANGAROO_TAG:
            if gesture != GESTURE.NONE:
                self.recorder._play_sound('boink.wav')
        elif tag == LION_TAG:
            if gesture != GESTURE.NONE:
                self.recorder._play_sound('lion_music.wav')
                self.recorder._play_sound('fanfare.wav')
        elif tag == 'e00401003da51c97':
            self.recorder._play_sound('2010-05-16 21:09:41.578919.wav') # piggy
        elif tag in tag_gesture_response_dict:
            if gesture in tag_gesture_response_dict[tag]:
                print 'playing sound for', gesture, 'on', tag
                self.recorder._play_sound(tag_gesture_response_dict[tag][gesture])
            else:
                print gesture, "trigger added for", tag
                tag_gesture_response_dict[tag][gesture] = "boink.wav" # default
                self.recorder._play_sound("boink.wav")
        if tag == NO_TAG:
            # mode 1 = empty handed gesture triggers response for last real tag
            # mode 2 = empty handed gesture gets its own trigger and response
            if self.app_mode == APP_MODE.GHOST:
                if gesture in tag_gesture_response_dict[self.last_nonzero_tag]:
                    self.recorder._play_sound(tag_gesture_response_dict[self.last_nonzero_tag]
[gesture])
            elif self.app_mode == APP_MODE.EMPTY:
                print gesture, "empty handed : now press button and record sound"
                tag_gesture_response_dict[tag][gesture] = "boink.wav" # default
                self.recorder._play_sound("boink.wav")
        else:
            self.last_nonzero_tag = tag

class TagFilter(object):
    def __init__(self):
        self.min_gap = 15
        self.none_count = 0
```

```
        self.prev_tag = NO_TAG

    def process_tag(self, tag):
        if tag == self.prev_tag:
            self.prev_tag = tag
            self.none_count = 0
            return (tag, GESTURE.NONE)

        if tag == NO_TAG:
            self.none_count += 1
            if self.none_count > self.min_gap: # RELEASE
                print 'r'
                self.prev_tag = NO_TAG
                self.none_count = 0
                return (NO_TAG, GESTURE.RELEASE)
            else:
                return (self.prev_tag, GESTURE.NONE)

        else:
            self.prev_tag = tag
            self.none_count = 0
            return (tag, GESTURE.GRAB)

def test_tag_filter():
    TAG1 = 'tag1'
    TAG2 = 'tag2'
    tag_streams = [([TAG1]*20 + [NO_TAG]*10 + [TAG2]*10 + [NO_TAG]*20,
                    [TAG1]*30 + [TAG2]*25 + [NO_TAG]*5),
                   ([TAG1]*20 + [NO_TAG]*20 + [TAG2]*10 + [NO_TAG]*20,
                    [TAG1]*35 + [NO_TAG]*5 + [TAG2]*25 + [NO_TAG]*5), ]

    for input_stream, expected_output_stream in tag_streams:
        tag_filter = TagFilter()
        actual_output_stream = []
        for tag in input_stream:
            (output_tag, gesture) = tag_filter.process_tag(tag)
            actual_output_stream.append(output_tag)
        eq_(expected_output_stream, actual_output_stream)

def main(options, args):
    reader = SerialInput(SERIAL_PORT, SERIAL_BAUD_RATE, INPUT_DIMENSIONS)

    if args[0] == RECO_MODE.SHAKE:
        recognizer = ShakeRecognizer(INPUT_DIMENSIONS)
    elif args[0] == RECO_MODE.DT:
        recognizer = DtRecognizer(INPUT_DIMENSIONS)
    elif args[0] == RECO_MODE.COMP:
        recognizer = CompRecognizer(INPUT_DIMENSIONS)
    else:
        assert False, 'Mode ALL not implemented'

    app = AudioApp(args[0], args[1])
    tag_filter = TagFilter()

    obs_0 = reader.read()
    eq_(len(obs_0), INPUT_DIMENSIONS + 2)
    prev_button_status = BUTTON_STATE.OFF
```

```
    print "grab object, perform gesture, and record sound by pressing button. \nunattached
object makes chime sound."

    ACTIVE_GESTURES = set(DT_GESTURE.values() + SHAKE_GESTURE.values())

    try:
        while True:

            obs = reader.read()
            #print obs #(244, 276, 344, '0000000000000000', 'off')

            tag = obs[-2]
            #print tag
            button_status = obs[-1] # comes from recognizer.py _read() function
            filtered_tag, grab_release = tag_filter.process_tag(tag)

            if app.recorder.read_sensor:
                if grab_release == GESTURE.RELEASE:
                    app.on_tag_release(tag)
                elif grab_release == GESTURE.GRAB:
                    app.on_tag_grab(tag)

                gesture = recognizer.append(obs[:INPUT_DIMENSIONS], tag) # (local, segemented)
string
                if gesture[0]:
                    if gesture[0] in ACTIVE_GESTURES:
                        reader.write("_4") # turn on LED
                    if gesture[0] == GESTURE.NONE:
                        reader.write("_3") #turn off LED
                if gesture[1] != GESTURE.NONE and gesture[1] != GESTURE.IDLE:
                    #print gesture[1]
                    app.on_gesture(tag, gesture[1])

            if button_status != prev_button_status:
                if button_status == BUTTON_STATE.ON:
                    app.on_button_down(tag)
                else:
                    app.on_button_up(tag)
            prev_button_status = button_status

    except KeyboardInterrupt:
        print 'Exiting'
        recorder.exit_program()
        sys.exit()

if __name__ == '__main__':
    usage = 'usage: %prog [options] [reco_mode] [app_mode]'
    opt_parser = OptionParser(usage=usage)

    options, args = opt_parser.parse_args()
    assert (len(args) == 2 and args[0] in RECO_MODE.values()
            and args[1] in APP_MODE.values()), opt_parser.usage
    main(options, args)
```

Bibliography

1. Arduino, http://www.arduino.cc/
2. Berlin, E., Liu, J., van Laerhoven, K., Schiele, B. 2010. Coming to grips with the objects we grasp: detecting interactions with efficient wrist-worn sensors. TEI '10. ACM, New York, NY, 57-64.
3. Dey, A. K., Hamid, R., Beckmann, C., Li, I., and Hsu, D. (2004). a CAPpella: programming by demonstration of context-aware applications. CHI '04. ACM, New York, NY, 33-40.
4. Feldman, A., Tapia, E. M., Sadi, S., Maes, P., and Schmandt, C. (2005). ReachMedia: On-the-move interaction with everyday objects. ISWC 2005, 52-59.
5. Hall, M., Frank, E., et al. (2009) The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1
6. Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S. R. (2007). Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. CHI '07. ACM, New York, NY, 145-154
7. Hartmann, B., Klemmer, S. R., et. al, J. 2006. Reflective physical prototyping through integrated design, test, and analysis. UIST '06. ACM, New York, NY, 299-308.
8. Buettner, M., Prasad, R., Philipose, M., and Wetherall, D. 2009. Recognizing daily activities with RFID-based sensors. Ubicomp '09. ACM, New York, NY, 51-60
9. Ishii, H., Mazalek, A., and Lee, J. 2001. Bottles as a minimal interface to access digital information. CHI '01. ACM, New York, NY, 187-188
10. Jurafsky, D., Martin, J. (2009) Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 2nd edition. Prentice-Hall.
11. Jordà, S., Geiger, G., Alonso, M., and Kaltenbrunner, M. 2007. The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. TEI '07. ACM, New York, NY, 139-146.
12. Katsumoto, Y., Inakage, M. (2007). Amagatana. ACM MULTIMEDIA '07. ACM, New York, NY, 361-362.
13. Kim, I., Im, S., E. Hong, E., Ahn, S., Kim, H., Adl clas- sification using triaxial accelerometers and rfid, in In- ternational Conference on Ubiquitous Computing Con- vergence Technology. Beijing, China, 2007
14. LeClerc, V., Parkes, A., and Ishii, H. 2007. Senspectra: a computationally augmented physical modeling toolkit for sensing and visualization of structural strain. CHI '07. ACM, New York, NY, 801-804.
15. Liu, J., Zhong, L., Wickramasuriya, J., and Vasudevan, V. (2009). uWave: Accelerometer-based personalized gesture recognition and its applications. Pervasive and Mobile Computing. 5, 6 (Dec. 2009), 657-675.
16. Nielsen, J. Usability Engineering, Morgan Kauffmann, San Francisco, CA, USA, 1994.
17. Polynor, R. 1995. The Hand That Rocks the Cradle. I.D., May/June 1995, pp. 60-65.

18. Processing, http://processing.org

19. Raffle, H. S., Parkes, A. J., and Ishii, H. 2004. Topobo: a constructive assembly system with kinetic memory. CHI '04. ACM, New York, NY, 647-654

20. Ryokai, K., Marti, S., and Ishii, H. 2004. I/O brush: drawing with everyday objects as ink. CHI '04. ACM, New York, NY, 303-310.

21. Scratch Project, http://scratch.mit.edu/

22. Shilman, M., Tan, D.S., Simard, P. (2006). CueTIP: A Mixed-Initiative Interface for Correcting Handwriting Errors. UIST 2006.

23. Vaucelle, C. and Ishii, H. 2008. Picture this!: film assembly using toy gestures. UbiComp '08, vol. 344. ACM, New York, NY, 350-359

24. Wilson, A. (2007) Depth-sensing video cameras for 3d tangible tabletop interaction, TABLETOP '07. pp. 201-204.

25. Wilson, A. and Shafer, S. (2003). XWand: UI for intelligent spaces. CHI '03. ACM, New York, NY, 545-552.

26. Zhang, H. and Hartmann, B. 2007. Building upon everyday play. CHI '07. ACM, New York, NY, 2019-2024.

27. Control Freaks http://failedrobot.com/thesis

28. Johansson, S. 2009. Sniff: designing characterful interaction in a tangible toy. IDC '09. ACM, New York, NY, 186-189

29. Verplaetse. C., Inertial Proprioceptive Devices: Self motion-sensing toys and tools. IBM Systems Journal Vol 35. No 3-4, 1996, 639-650

30. Hudson, S., Mankoff, J. Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. UIST '06. 289-298.

31. MITes and Wockets http://web.mit.edu/wockets/

32. Revelle, G., Zuckerman, O., Druin, A., and Bolas, M. 2005. Tangible user interfaces for children. In CHI '05 Extended Abstracts on Human Factors in Computing Systems (Portland, OR, USA, April 02 - 07, 2005). CHI '05. ACM Press, New York, NY, 2051-2052. DOI= http://doi.acm.org/10.1145/1056808.1057095

33. Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. 2004. Papier-Mâché: toolkit support for tangible input. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Vienna, Austria, April 24 - 29, 2004). CHI '04. ACM, New York, NY, 399-406. DOI= http://doi.acm.org/10.1145/985692.985743

34. B. Ullmer and H. Ishii. The metaDESK: models and prototypes for tangible user interfaces. In Proceedings of 1997 ACM Symposium on User Interface Software and Technology, pages 223–232, 1997.

35. Carvey, A., Gouldstone, J., Vedurumudi, P., Whiton, A., and Ishii, H. 2006. Rubber shark as user interface. In CHI '06 Extended Abstracts on Human Factors in Computing Systems (Montréal, Québec, Canada, April 22 - 27, 2006). CHI '06. ACM, New York, NY, 634-639. DOI= http://doi.acm.org/10.1145/1125451.1125582

36. Adam Kumpf. Trackmate: Large-Scale Accessibility of Tangible User Interfaces. Thesis (M.S.)--Massachusetts Institute of Technology, School of Architecture and Planning, Program in Media Arts and Sciences, 2009.

37. Zigelbaum, J., Browning, A., Leithinger, D., Bau, O., and Ishii, H. 2010. g-stalt: a chirocentric, spatiotemporal, and telekinetic gestural interface. In Proceedings of the Fourth international Conference on Tangible, Embedded, and Embodied interaction (Cambridge, Massachusetts, USA, January 24 - 27, 2010). TEI '10. ACM, New York, NY, 261-264. DOI= http://doi.acm.org/10.1145/1709886.1709939

38. Ishii, H. and Ullmer, B. 1997. Tangible bits: towards seamless interfaces between people, bits and atoms. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Atlanta, Georgia, United States, March 22 - 27, 1997). S. Pemberton, Ed. CHI '97. ACM, New York, NY, 234-241. DOI= http://doi.acm.org/10.1145/258549.258715

39. Hudson, S., Mankoff, J. Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. UIST '06. 289-298.

40. Martinussen, E. S., Knutsen, J., and Arnall, T. 2007. Bowl: token-based media for children. In Proceedings of the 2007 Conference on Designing For User Experiences (Chicago, Illinois, November 05 - 07, 2007). DUX '07. ACM, New York, NY, 3-16. DOI= http://doi.acm.org/10.1145/1389908.1389930

41. Adobe Flash http://www.adobe.com/products/flash/

42. Collapsible: The Genius of Space- saving Design. By Per Mollerup. Chronicle Books, 2001 ISBN 0811832368, 9780811832366

43. Bijan Aryana, Seyed Javad Zafarmand, Sardar Hajati Modaraie, Caro Lucas, and Somayeh Naghibi, APPLICATION OF OBJECT ORIENTED THINKING IN PRODUCT DESIGN: DESIGN PROCESS OF PERSONAL DIGITAL PARTNER. IASDR (International Association of Societies of Design Research). 2007

44. ULLMER, B. 2002. Tangible Interfaces for Manipulating Aggregates of Digital Information. Ph.D. dissertation, MIT Media Laboratory, 2002.

45. Choudhury, T., Borriello, G., Consolvo, S., Haehnel, D., Harrison, B., Hemingway, B., Hightower, J., Klasnja, P., Koscher, K., LaMarca, A., Landay, J. A., LeGrand, L., Lester, J., Rahimi, A., Rea, A., and Wyatt, D. 2008. The Mobile Sensing Platform: An Embedded Activity Recognition System. IEEE Pervasive Computing 7, 2 (Apr. 2008), 32-41. DOI= http://dx.doi.org/10.1109/MPRV.2008.39

46. Nintendo Wii Remote http://en.wikipedia.org/wiki/Wii_Remote

47. Touch-a-tag http://www.touchatag.com/

48. Violet Mir:ror http://www.violet.net/_mirror-give-powers-to-your-objects.html

49. Brandon T. Taylor and V. Michael Bove, Jr.. 2009. Graspables: grasp-recognition as a user interface. In Proceedings of the 27th international conference on Human factors in computing systems (CHI '09). ACM, New York, NY, USA, 917-926. DOI=10.1145/1518701.1518842 http://doi.acm.org/10.1145/1518701.1518842

50. Apple iPhone tech specs http://www.apple.com/iphone/specs.html

51. Jacob, R. J., Ishii, H., Pangaro, G., and Patten, J. 2002. A tangible interface for organizing information using a grid. InProceedings of the SIGCHI Conference on Human Factors in Computing Systems: Changing Our World, Changing Ourselves (Minneapolis, Minnesota, USA, April 20 - 25, 2002). CHI '02. ACM, New York, NY, 339-346. DOI= http://doi.acm.org/10.1145/503376.503437

52. Parkes, A., Poupyrev, I., and Ishii, H. 2008. Designing kinetic interactions for organic user interfaces. Commun. ACM 51, 6 (Jun. 2008), 58-65. DOI= http://doi.acm.

org/10.1145/1349026.1349039

53. Parkes, A. 2008. Phrases of the Kinetic: Dynamic Physicality as a Construct of Interaction Design. Thesis Proposal for the degree of Doctor of Philosophy at the Massachusetts Institute of Technology

54. Kelley, T., Littman, J. The art of innovation: lessons in creativity from IDEO, America's leading design firm, Random House, Inc., 2001. ISBN: 0385499841, 9780385499842. Pages 62-112

55. Raffle, H., Vaucelle, C., Wang, R., and Ishii, H. 2007. Jabberstamp: embedding sound and voice in traditional drawings. InProceedings of the 6th international Conference on interaction Design and Children (Aalborg, Denmark, June 06 - 08, 2007). IDC '07. ACM, New York, NY, 137-144. DOI= http://doi.acm.org/10.1145/1297277.1297306

56. SEGD Annual Conference + Expo 2010, http://segd-dc2010.com/

57. Choking hazard regulations http://www.nypirg.org/consumer/toysafety03/choking.html

58. Kendon., A. Gesture: visible action as utterance. Cambridge University Press, 2004. ISBN: 0521542936, 9780521542937. Pages 12-13

59. OnObject ABC demonstration video http://vimeo.com/12562654

60. OnObject Storytelling demonstration video http://vimeo.com/12562823

61. OnObject play session video documentation http://vimeo.com/12218305

62. OnObject Swordplay demonstration video http://vimeo.com/12562881

63. OnObject Amagatana demonstration video http://vimeo.com/10659648 starting at 1:00

64. Frei, P., Su, V., Mikhak, B., and Ishii, H. 2000. curlybot: designing a new class of computational toys. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 129-136. DOI= http://doi.acm.org/10.1145/332040.332416

65. Baudisch, P., Becker, T., and Rudeck, F. 2010. Lumino: tangible blocks for tabletop computers based on glass fiber bundles. In Proceedings of the 28th international Conference on Human Factors in Computing Systems (Atlanta, Georgia, USA, April 10 - 15, 2010). CHI '10. ACM, New York, NY, 1165-1174. DOI= http://doi.acm.org/10.1145/1753326.1753500

66. Patten, J., Ishii, H., Hines, J., and Pangaro, G. 2001. Sensetable: a wireless object tracking platform for tangible user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Seattle, Washington, United States). CHI '01. ACM, New York, NY, 253-260. DOI= http://doi.acm.org/10.1145/365024.365112

67. E. M. Tapia, N. Marmasse, S. S. Intille, and K. Larson, "MITes: Wireless portable sensors for studying behavior," in Proceedings of Extended Abstracts Ubicomp 2004: Ubiquitous Computing, 2004.

68. Tom Igoe's accelerometer datalogging Processing sketch http://itp.nyu.edu/physcomp/sensors/Code/DataloggerMulti

All URLs were last accessed on 7/29/2010.

Image Credits

The Idea

Figure 3 I/O brush http://web.media.mit.edu/~kimiko/iobrush/
Figure 3 Amagatana + Fula version 2 http://www.flickr.com/photos/yuichirock/3446581519
Figure 3 Reactable http://www.flickr.com/photos/dav83/440110298/
Figure 3 G-Stalt demo http://zig.media.mit.edu/Work/G-stalt

Related Work

Figure 7 Control Freaks http://failedrobot.com/thesis/
Figure 10 Sniff http://www.nearfield.org/sniff/concept.html
Figure 10 musicBottles http://www.flickr.com/photos/danagordon/663460695
Figure 11 Picture This! http://www.flickr.com/photos/cati/4118659556
Figure 12 Picture This! http://web.media.mit.edu/~cati/portfolio/PictureThis.html
Figure 14 G-Speak screen capture from http://vimeo.com/2229299

System Design and User Experience

Figure 15 Notes image by Jean-Baptiste Labrune

Applications

Figure 38 Marble Answering Machine screen capture from Tangible Bits [18]
Figure 38 musicBottles http://www.flickr.com/photos/danagordon/663460695
Figure 38 Amagatana + Fula player http://www.flickr.com/photos/yuichirock/2337584358/
Figure 40 Amagatana version 1 umbrellas http://www.flickr.com/photos/yuichirock/3033763205/
Figure 43 Knives by Herman Au http://www.flickr.com/photos/hermanau/366678492/
Figure 43 Knife skills by jeromebot http://www.flickr.com/photos/pixelmassive/4413235758/

Conclusions

Figure 54 Fantasia http://wwww.fanpop.com/spots/classic-disney/images/5776599/title/fantasia-wallpaper-wallpaper and http://www.graphicshunt.com/wallpapers/images/mickey_mouse_in_fantasia_-492.htm

All other images were created by the author.