ProMadgets

Prototyping of tangible magnetic widgets for interactive tabletops

Diploma Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Christopher Gretzki

Thesis advisor:
Prof. Dr. Borchers

Second examiner:
Prof. Dr. Schroeder

Registration date:	Feb 15th, 2011
Submission date:	Aug 15th, 2011

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, August 2011
Christopher Gretzki

Contents

List of Figures

List of Tables

Abstract

As interactive tabletops are becoming increasingly relevant for the use in everyday applications, there is a need for general-purpose controls to manipulate arbitrary virtual data. When designing these controls, the time it takes to build early prototypes is critical, since it affects the number of design iterations possible in the available time, which eventually influences the quality of the final product (as stated in Nielson [1993].)

Actuated tabletops offer a rich user experience by supporting a bilateral haptic communication between the user and the system. These systems actuate tangible controls to keep them consistent with their virtual representation. The implementation of the actuation technology inside the tabletop (e.g., *the Actuated Workbench* by Gian Pangaro [2003]) is particular helpful for prototyping. This enables the usage of passive tangibles, which are low-cost and easy to build since they do not contain any electronics. While recent tabletop actuation systems facilitate prototyping of new tangibles theoretically, toolkits to do so practically have received only little attention.

In my thesis I present a toolkit to leverage the prototyping capabilities of *Madgets* (see *Actuated translucent controls for dynamic tangible applications on interactive tabletops* by Weiss et al. [2009]). A user study has been conducted that gives empirical evidence of the prototyping capabilities of the Madgets technology using this toolkit. Although the software prototype is limited to a small set of Madgets it already shows that it is possible to prototype a fully functional Madget in under two hours. The results show that the visual programming paradigm of the toolkit offers a low threshold for the integration of a new Madget from scratch. Furthermore, teething problems of the Madgets tabletop hardware could be identified as causes of difficulties in the prototyping process.

Überblick

Der Einsatz von interaktiven Tischen wird immer relevanter für alltägliche Anwendungen. Dadurch steigt auch die Notwendigkeit von Eingabegeräten für diese Technologie, die Anwendungs übergreifend einsetzbar sind. Die Zeit, die zum Entwerfen dieser Eingabegeräte gebraucht wird, bestimmt wieviele Iterationen in der zur Verfügung stehenden Zeit möglich sind. Nach Jakob Nielson (Nielson [1993]) ist dies ein kritischer Faktor, da die Anzahl der Iterationen in der frühen Prototyping-Phase einen starken Einfluss auf die Qualität des Endprodukts habe.

Aktuierende interaktive Tische sind Systeme, die die physikalischen Eingabegeräte aktuieren, d.h. steuern können. Dadurch ist eine bilaterale, haptische Kommunikation zwischen dem Benutzer und dem System möglich, so dass die Eingabegeräte mit ihren digitalen Repräsentationen synchronisiert werden können. Wird die Aktuierung durch Technologie im interaktiven Tisch ermöglicht (z.B: *the Actuated Workbench* von Gian Pangaro [2003]), kommen die Eingabegeräte selbst ohne Elektronik aus. Dieser Ansatz ist für das Prototyping besonders nützlich, da die Eingabegeräte einfacher und günstiger zu konstruieren sind. Es gibt bereits einige Technologien interaktiver Tische, die Prototypen aktuierter Eingabegeräte ermöglichen. Allerdings fanden Software Toolkits zur Unterstützung dieses Prototyping Prozesses eher wenig Beachtung.

In meiner Diplomarbeit stelle ich ein Software Toolkit vor, dass das Prototyping von physikalischen Eingabegeräten für aktuierende, interaktive Tische ermöglicht. Die Software unterstützt das Madgets System (siehe *Actuated translucent controls for dynamic tangible applications on interactive tabletops* von Weiss et al. [2009]). Um die Fähigkeiten von Madgets als Prototyping Werkzeug zu testen, wurde eine empirische Studie durchgeführt. Die Ergebnisse zeigen, dass bereits der Software Prototyp, der auf wenige Madgets limitiert ist, das Prototyping eines voll funktionsfähigen Madgets in unter zwei Stunden ermöglicht. Desweiteren kann gezeigt werden, dass das Toolkit durch seine visuelle Programmierumgebung eine geringe Einstiegshürde aufweist. Außerdem können Schwierigkeiten der Probanden beim Prototyping auf technisch nicht ausgereifte Details der Hardware des Madget's Tisches zurückgeführt werden.

Danksagung

Diese Diplomarbeit möchte ich meinen Eltern widmen,
die mich während des gesamten Studiums unterstützt haben,
immer an mich glauben und für mich da sind!

Diese Arbeit ist auch für Johanna,
die mir auf ihre eigene, einzigartige Weise Kraft gibt,
mir immer hohe Ziele zu setzen.
Aber auch immer daran erinnert,
niemals die wichtigen Dinge aus den Augen zu verlieren!

Für meinen Bruder…
um es ihm unter die Nase zu reiben –
als Ansporn auch einmal fertig zu werden ;)

Für Tina & Alex
ohne die ich gar nicht gewusst hätte,
wie ich das letzte Unijahr hinter mich bringen sollte.
Mit denen ich eine super Zeit in Aachen hatte.

Danke!

Conventions

Throughout this thesis I use the following conventions.

UML class diagrams

The notation of UML class diagrams is slightly adapted to fit to the Objective-c language, namely the notation of methods: Only the important properties and methods according to the context referring the diagram are listed; unless stated otherwise. Method signatures starting with the plus sign (+) are class methods, whereas the minus sign (-) denotes an instance method.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

Excursus:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition: *Excursus*

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English.

Chapter 1

Introduction

Rapid prototyping is "a specific strategy for performing requirements definitions wherein user needs are extracted, presented, and successfully refined by building a working model of the ultimate system quickly" (Boar [1984]). It is an iterative and user-centered methodology, where the user interface (UI) is designed, implemented, evaluated and refined again. The test purpose is a quick identification of functional requirements, the investigation of the needs of the user and to find usability flaws. The time these iterations take is critical, since it affects the number of iterations possible in the available time, which eventually influences the quality of the final product. (As stated in Nielson [1993].)

When prototyping tangible UIs, the physicality of the prototype has a big impact on the user's experience of the UI. The physical part oftentimes requires profound knowledge in electronics, the integration into a software system is hindered by the required skills in programming. Some toolkits provide help with the electronics part by providing all kinds of sensor parts that can be attached to the physical prototype to extend its functionality, but still require a low-level programming language to use those. Other toolkits concentrate on the programming part only and e.g., provide mechanisms to program UIs by simply demonstrating gestures one want to get recognized (Exemplar). But it is important to decouple form and functionality, so that both aspects – the functionality and the physicality of the UI – can be dealt with independently and iterated at the same time. (see Avrahami and Hudson [2002]).

This work is about prototyping of tangible controls for the tabletop domain. The systems in that domain can be grouped into supporting one-directional and bi-directional communication, whereas the latter features richer feedback capabilities. Bidirectional communication is guaranteed either via electronics inside the tangibles (Rosenfeld et al. [2004]) or actuation technology incorporated into the tabletop (Gian Pangaro [2003]). Untethered tangibles are more appropriate for rapid prototyping for the following reason: They are faster and easier to construct since they do not contain any electronics, which also enables more freedom in their visual gestalt.

While there are several tabletop technologies that theoretically allow prototyping of tangible UIs, software toolkits to rapidly create functional prototypes have received only little attention. In this work, a toolkit is developed and evaluated to provide empirical evidence about the appropriateness of Madgets (Weiss et al. [2010]) as prototyping tool.

A toolkit for rapid prototyping should have a low threshold and high ceiling. A low threshold enables many iterations in the early prototype stage and eventually enhances the quality of the final product. A high ceiling allows the prototype to extend after the initial phase. This toolkit should support both goals with one consistent way of operation, i.e., more advanced prototypes should not require to switch to a different programming paradigm. Furthermore, the toolkit should support the work flow of the user. In order to achieve this goal, the design starts with a low fidelity prototype and a pilot study, to get a system image close to the user's mental model of how the system works.

After that, a medium fidelity prototype is implemented that runs on the targeted tabletop hardware. The feature set will be limited to a scenario that enables the construction of Madgets that can dynamically change their physical properties as described in *Rendering physical effects in tabletop controls* (Weiss et al. [2011]). This set of Madgets best represents the degree of freedom of functionality and physicality the technology offers.

Overview

In Chapter 2—“Related work”, I show several systems that have been developed to support the prototyping process in related fields, of tangible user interfaces and of controls for tabletops.

In Chapter 3—“Fundamentals”, I will show how the actuated tabletop and its tangible widgets work. I also show two frameworks that were developed for this hardware to enable a bidirectional communication between the user and this system.

I then present my own toolkit to prototype new tangibles for the actuated tabletop which incorporates ideas of Hartmann et al. [2007] to ease the programming of the tangible. The storyboard and paper prototype implementing the UI are discussed in chapter 4—“Low fidelity prototype”. A software prototype is developed that runs on the Madgets tabletop. This prototype is discussed in detail in chapter 5—“Medium fidelity prototype”. A final evaluation is carried out to test the appropriateness of the developed prototyping system. Chapter 6—“Final evaluation” describes the setup of the user study and discusses its results.

Chapter 7 summarizes the results and contribution of this work and gives an outlook on future research ideas.

Chapter 2

Related work

"Every tool carries with it the spirit by which it has been created."

—Werner Karl Heisenberg

This section discusses previous work in related fields, especially in the field of tangible user interfaces (TUIs). Prototyping TUIs requires technical expertise in several fields to cope with the following problems: The physicality of the prototype must be carefully crafted to have the desired affordances and form factor. Also, the choice of supported input/output technology – to study the user's interaction with the device or e.g., render tactile feedback – influences the form factor and vice versa. The other field is the programming of the device or software to acquire, abstract and use the input.

The following toolkits try to eliminate these obstacles and, as a consequence, reduce the development time. The first toolkits listed here focus primarily on the tangibles and acquisition of the physical data. The latter focus on the abstraction of the input data and support for the programming part. Finally, toolkits about the prototyping of TUIs in the tabletop domain are discussed.

Switcharoo (Avrahami and Hudson [2002]) is a tool to enable designers to prototype physical interactive products without any special technical knowledge. Prototypes are augmented with input components that are both wireless and passive, so that the technology does not interfere with the form factor or the experience of the interaction – the authors stress the importance to explore the form and interactivity of physical prototypes at the same time. The components can communicate to *Macromedia Director* (*Adobe Director* was formerly named *Macromedia Director*. It is a multimedia application authoring platform using a movie metaphor and own scripting language. to originally build animation sequences. The scripting language is aimed for designers rather than programmer. See Adobe's Director Homepage[1] for more information.)

As proof of concept a small-scale button is presented (see Figure 2.1) that uses passive RFID[2] , so that it does not require batteries and works wirelessly. An antenna must be installed nearby to transfer power to the component.
The advantage of this system is that these components can easily be integrated in physical prototypes and make only minimal restrictions on the form factor. Several buttons can be composed to more complex devices like a slider but the complexity is strongly limited.

Madgets have only few form constraints too and allow an exploration of form and functionality at the same time. They support a wider range of tangibles and do not require a nearby power source, which can complicate the setup.

Phidgets (Greenberg and Fitchett [2001]) is a toolkit containing several general-purpose physical user interface components that can be combined into more complex ones. The toolkit's primary aim is to hide the electronics details from the designer. The communication is encapsulated into an API to circumvent low-level hardware programming. Figure 2.2 shows *phidgets* of the general-purpose construction kit like switches, LEDs and sensors. This toolkit features several different sensors allowing to test a variety of different interactions. *Phidgets* are quite small, which is optimal in regard of the form factor of the prototype. But the com-

[1]http://www.adobe.com/de/products/director/

[2]Radio-frequency identification (RFID) is a technology that uses radio waves to transfer data from an electronic tag through a reader for the purpose of identifying and tracking objects.

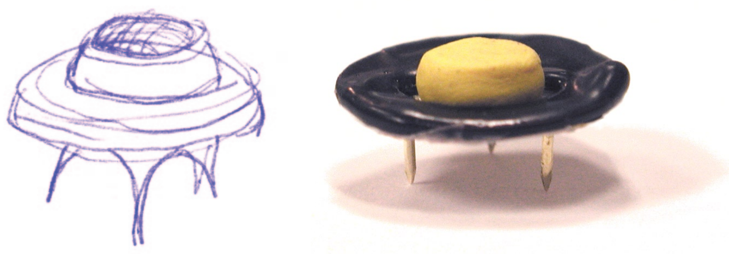


Figure 2.1: Sketch and construction of a small-scale button of the *Switcharoo* toolkit. The three pins on the back are for attachment. This button uses passive RFID technology and requires an antenna nearby for power transfer.

ponents are not wireless, which in turn can interfere with the physicality of and interaction with the prototype.

Madgets do not require any wiring and thus, are easier to construct. Although the *Phidgets* toolkit is obviously targeted at programmers, the tasks of data acquisition and processing is unnecessary complex. ProMadgets offers a visual programming language, which accelerates the programming task significantly.

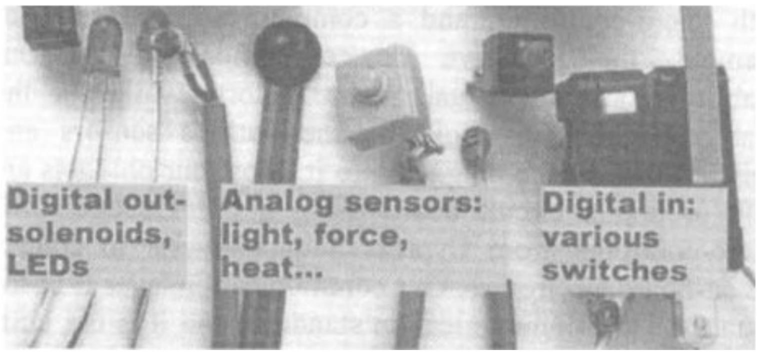

Digital out- solenoids, LEDs
Analog sensors: light, force, heat...
Digital in: various switches

Figure 2.2: The *Phidgets* toolkit provides a general-purpose construction kit consisting of several switches, LEDs and sensors. These can can be connected via USB to a PC and communicate via an API, which eliminates the need for low-level programming.

CookieFlavors (Kimura et al. [2006]) is also a toolkit that tries to reduces the complexity of the prototyping task by hiding the heterogeneous nature of input devices. To do so, it provides a set of general-purpose components, which are called *Cookies*. These *Cookies* are coin-sized, wireless components (see Figure 2.3) with different types of sensors for different input primitives: rotating, tilting and knocking. The components communicate sensory data via bluetooth. The small form factor and wireless communication are advantageous for prototyping the physicality of the tangible (for the same reasons as stated before.) But this work does not provide any support for data acquisition. Also, in order to customize *Cookies* the firmware has to be edited. Although the components cover interesting input dimensions, position tracking is missing. Another obstacle is the short battery life (about two hours), which disturbs the prototyping process and makes extended tests impractical.

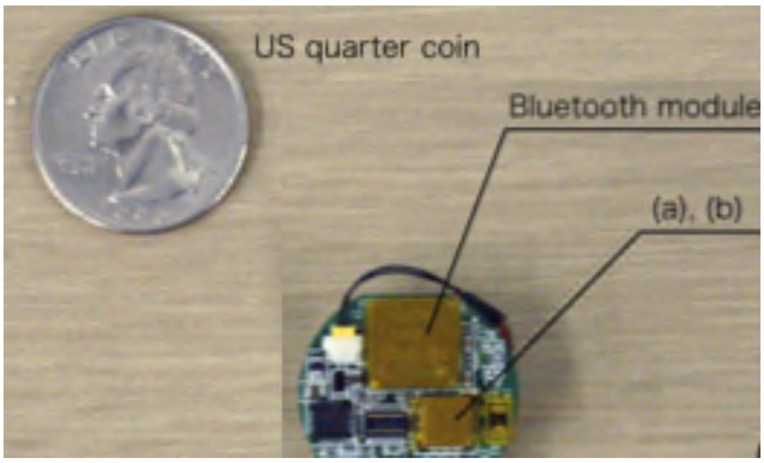


Figure 2.3: Components of the *CookieFlavor* toolkit are coin-sized, communicate wirelessly via bluetooth and support different kinds of sensors; e.g., (a) 2-axis linear acceleration, (b) 3-axis magneto impedance. The onboard battery lasts for about 2 hours.

In Kimura et al. [2007] this work is extended to eliminate the shortcomings stated above. It offers location tracking via visual tags and adopts *Apple's Quartz Composer*.

Quartz Composer is a visual programming language for processing and rendering graphical data. (See Quartz Composer User Guide[3]

for more details.) While its visual programming paradigm lowers the threshold for non-programmers, it also has clear limitations: More complex interaction sequences require some kind of Finite state machine, which is not supported by *Quartz Composer*. ProMadgets also supports a visual programming environment to lower the threshold for non-programmers.

Other toolkits like *Papier-Maché* (Klemmer et al. [2004]) or *DisplayObjects* (Akaoka et al. [2010]) use vision-based approaches to track the tangible controls and achieve a better decoupling of physicality and functionality of the tangibles. *DisplayObjects* uses an accurate 3D model of the physical prototype, texture-mapped with interactive elements like buttons, to project a realistic looking interface onto the physical prototype. Retroreflective markers on the prototype and on the user's finger (as seen in Figure 2.4) are captured by a *Vicon motion capture system*[4] to track the orientation of the prototype and to enable the interaction with the interface.

The vision-based approach best decouples the physicality and functionality of the prototype, since the tangible does not require any electronics to function. The retroreflective markers make only very small limitations to form factor, but the projection of the interface on the prototype will be occluded by the user, which can negatively impact the experience of the interaction.

The Madgets tabletop uses vision-based tracking too, but the back projection does not lead to occlusion problems. Instead of the Vicon motion capture system Diffused Surface Illumination (see 3.1—"Tabletop hardware") is incorporated to track the tangibles. The required markers to track Madgets set comparable limitations on the form factor.

[3]http://developer.apple.com/library/mac/#documentation/GraphicsImaging/Conceptual/QuartzComposerUserGuide

[4]Vicon motion capture system is an infrared marker-tracking system that offers millimeter resolution of 3D spatial displacements.

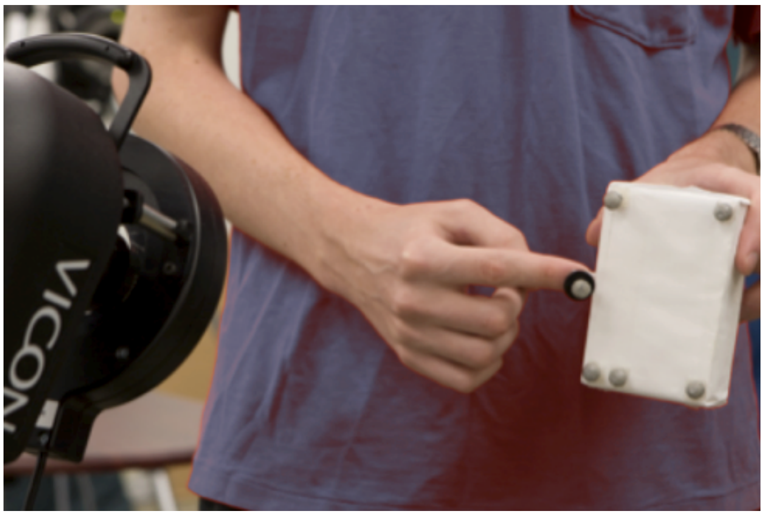


Figure 2.4: A physical prototype made out of Styrofoam with retroreflective markers is used with the DisplayObjects toolkit. The markers are attached to enable a Vicon system to capture its location, orientation and deformation. A 3D model of the prototype is used to project an interface onto the object's surface. Another marker on the user's finger is used to interact with the interface.

Toolkits supporting the abstraction and processing of the input data

A CAPpella by Dey et al. [2004] is a prototyping environment for context-aware recognition-based (desktop) systems, combining data streams from discrete sensors, a vision algorithm and a microphone. It allows programming context recognizers by demonstration and selecting the relevant data inside a GUI. *A CAPpella* uses pattern recognition to substitute algorithm tinkering with generalization from the annotated samples. This *programming by demonstration* (PBD) approach enables to build context-aware recognizers without writing any code, i.e., would otherwise require advanced programming skills and cost much time.

The PBD paradigm is used in other prototyping systems as well (Hartmann et al. [2007], Fails and Olsen [2003], Li and Landay [2005]) since it offers an intuitive input method that lowers the threshold of programming.

ProMadgets implements the PBD paradigm for authoring the possible movements and actuation paths of new controls for the Madgets tabletop. A slider is programmed by demonstrating the linear movement of the carriage to the system. The tabletop registers the shape of the path and adds functionality to, e.g., use the position data of the slider to control the viewport of a text document.

Interactive tabletops

Interactive tabletops are a technology that displays data and detects touch and tangible object input. Its main characteristics is direct interaction with data through tangible objects (or tangibles); the digital bits become tangible (Ishii and Ullmer [1997]). Compared to traditional desktops with keyboard and mouse the interaction is more direct. Another characteristic is that physical objects have become augmented with computational abilities; the body becomes more relevant for the interaction (Dourish and Dourish [2004]).

To track touches tabletop systems use capacitive or resistive sensing. In order to track objects too, optical tracking technologies with infrared cameras are commonly used. The most prominent variants are *Frustrated Total Internal Reflection* and *Diffused Surface Illumination*. Bill Buxton provides a good overview of multi-touch systems on his homepage (www.billbuxton.com[5]).

A bilateral communication between the user and the tabletop system can be achieved in two ways: One approach is to let tangibles actuate themselves like in the *Planar Manipulator Display* (Rosenfeld et al. [2004]). But that makes them harder to construct, besides the disadvantage that more maintenance is required, e.g., for changing the battery. An alternative is to use passive tangibles and incorporate the actuation technology inside the tabletop like the *Actuated Workbench* (Gian Pangaro [2003]). These systems decouple form and functionality of the tangibles, which is advantageous for prototyping.

[5]http://www.billbuxton.com/multitouchOverview.html

Toolkits to prototype tangible controls on tabletops

Reconfigurable ferromagnetic input devices (Hook et al. [2009]) are input devices that have a free physical form and must include ferrous objects. The tabletop hardware can detect the presence, position and deformation of the ferrous objects inside the tangible controls.

This work provides a new degree of freedom in the design of the input devices and the type of input – the deformation – for tabletop controls. A C# library pre-processes the input data to provide clean and reliable data, but this data can be acquired via a shared memory only, which requires profound programming knowledge to query and interpret the data.

Madgets (Weiss et al. [2010]) keep the idea of passive tangibles, and extend it to more complex widgets. These are handled as a collection of rigid bodies that can be actuated independently. This permits the physical shape to remain very basic and at the same time offers a high degree of freedom.

While these toolkits help to design controls for tabletops, they lack support to acquire and process this data. ProMadgets alleviates these issues by providing a visual authoring environment that offers easy registration and integration of physical prototypes into the *Madgets* tabletop.

Chapter 3

Fundamentals

"If anything can go wrong, it will."

—Murphy's law

This chapter discusses the hardware and software this work is based on. The first section explains the assembly of the *Madgets tabletop* and its tangible widgets. The second section discusses two frameworks that were developed for this hardware and are used for the software prototype of the toolkit.

3.1 Tabletop hardware

Figure 3.1 shows the different layers of the hardware setup. The Madgets tabletop combines Diffused Surface Illumination[1] with electromagnetic actuation. An array of electromagnets underneath the TFT panel is used to actuate objects on top of the tabletop. In order to track objects an Endlighten acrylic is employed as top layer, fed with infrared (IR) light by the surrounding LEDs. An IR camera

[1]Diffused Surface Illumination is a multi-touch technique that uses a special acrylic named Endlighten. This acrylic disperses even light if the edges are lighted. This technique is generally inspired by Tim Roth. See http://iad.projects.zhdk.ch/multitouch/ for more details.

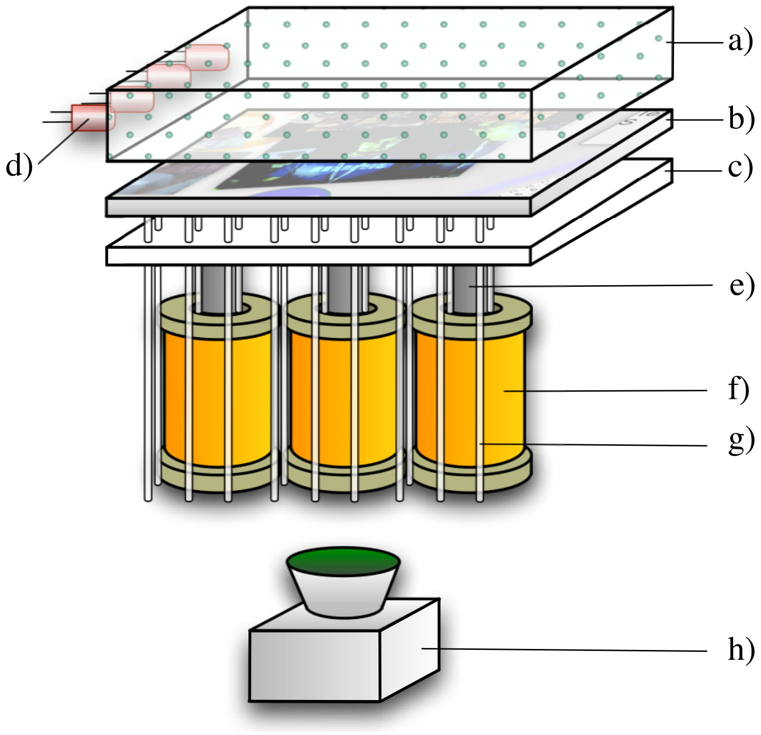


Figure 3.1: Schematic view of the employed tabletop technology. a) Endlighten acrylic. b) TFT Panel. c) EL foil. d) IR LED. e) Iron rod core. f) Electromagnet. g) Fiber optic cable. h) IR camera.

beneath the table captures the light, which is diffusely reflected from objects touching the Endlighten layer. To circumvent the occlusion of the light by intermediate layers, a matrix of fiber optic cables among the electromagnets and around their iron rod cores is used. The cables pierce the EL foil underneath the TFT panel and transport the light to the camera. This technique provides a heavily down-sampled tracking resolution (58×37).

3.1.1 Tangibles of the Madgets tabletop

Madgets are tangible controls for the Madgets tabletop. They are used as interactive widgets, i.e., they are used to control the system but can also be controlled vice versa. The widget has magnets attached to it to enable the tabletop to hold it in place or move it around. This ensures a bidirectional communication between the user and the system. Furthermore, cylindric markers attached to the bottom of a Madget provide reference points for tracking. These markers are made of bright paper to best reflect the IR light inside the Endlighten acrylic.

The radio-button Madget in Figure 3.2 is made of transparent acrylic, so that its appearance can be changed dynamically using the display. The arrangement of the three markers at the edges produce its unique footprint. The markers on the button plates tell the system whether they are currently lifted.

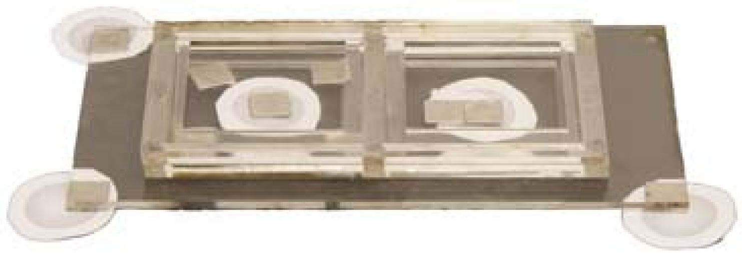


Figure 3.2: Photo of a radio-button Madget with two buttons. To be able to elevate the button plates there are magnets attached to them. The markers on the plates allow to check if the actuation succeeded.

Additionally, there is a permanent magnet for every marker. This allows the system to control the Madget's state, as illustrated in figure 3.3: A positive magnetic force pulls one button plate whilst the other one is repelled using a negative magnetic force to indicate the button's state as off or on respectively. Furthermore, the Madget is hold in place by pulling the magnets at the corners.

3.1.2 Construction constraints of Madgets

In this hardware setup the following constraints are set to the design of Madgets: If two magnets are too close together, they cannot be controlled independently. This is

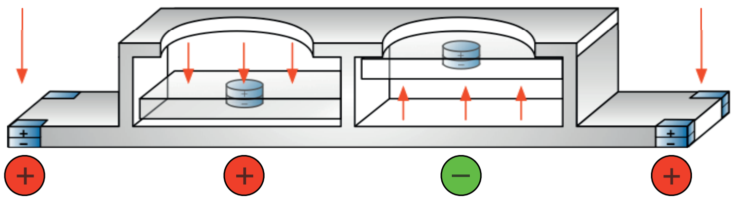


Figure 3.3: Schematic of an actuated radio-button Madget. The plates are pulled or pushed to indicate their state as off or on. The edges are pulled too to keep the Madget in place.

due to the size and alignment of the electromagnets. Therefore, the minimal distance between each pair of magnets is about *3 cm*. Furthermore, the maximal (magnetic) force with which Madgets can be actuated is very limited. An informal experiment showed that the maximal weight that can be lifted is about 16 gram. Thus, the maximal weight and friction of movable parts has to be reduced as much as possible.

Another limitation is the size of and distance between markers due to the input resolution. In a first experiment *3 cm* for the diameter and *3 cm* for the minimal distance between markers were determined, so that the markers can be distinguished from each other and are visible all the time.

3.2 Software frameworks

The following two frameworks provide the basis for a bidirectional communication between the user and the tabletop system. One direction is the user inputting data to the system via the Madgets. This is done (amongst others) by the SLAP framework. The other direction is the system outputting data to the user. This is accomplished by the Actuation framework, which manipulates the Madgets using the electromagnet array to represent an update of their current state or to react to internal events.

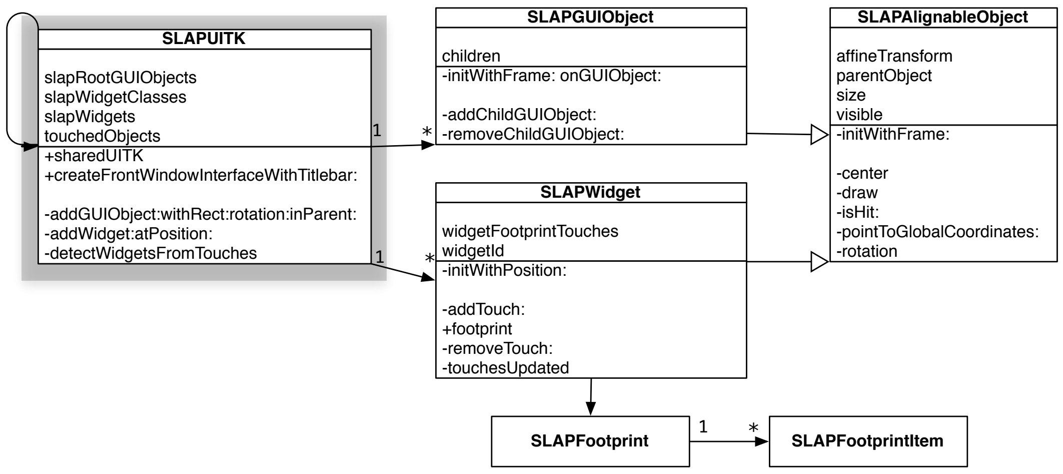


Figure 3.4: Class diagram of the most important classes of the SLAP framework.

3.2.1 SLAP framework

Figure 3.4 shows the most important classes of the SLAP framework. As stated above, one of its main functionality is the tracking of Madgets. Note that it is working with instances of the `SLAPWidget` class. It is the Actuation framework that synchronizes these with `Madget` instances.

The `SLAPUITK` singleton periodically invokes `detectWidgetsFromTouches` to traverse all incoming touch events in a pattern matching algorithm. The search-patterns are footprints of Madgets represented by `SLAPFootprint` objects. They are retrieved by invoking the `footprint` method on the `SLAPWidget` classes listed in the `slapWidgetClasses` property of the `SLAPUITK` singleton. If a pattern matches a footprint the correspondent `SLAPWidget` is created and its position and orientation is set accordingly.

The SLAP framework also provides a basic GUI system allowing the composition of `SLAPGUIObject` objects and handles touch events to, e.g., drag or resize those objects. The properties and methods for positioning, resizing and to perform a hit test are encapsulated in the `SLAPAlignableObject` class, which is the super class of `SLAPGUIObject` and `SLAPWidget`. The `SLAPGUIObject` class extends this functionality with methods for compositing. No visuals are provided but can be implemented in subclasses by overwriting the `draw:` method.

The `touchedObjects` property of the `SLAPUITK` singleton keeps track of all touch events that are associated with instances of `SLAPGUIObject` or `SLAPWidget`.

3.2.2 Actuation framework

Figure 3.5 shows the most important classes of the Actuation framework. It is initialized by invoking the `createWithSLAPUITK:` method on the `ActuationFramework` class with a reference on the above mentioned `SLAPUITK` singleton. This sets up the `ActuationFramework` singleton which stores a weak reference on the `SLAPUITK` instance and creates and retains an `ActuationProcessor` instance.

The `ActuationFramework` singleton periodically queries the `SLAPUITK` instance for a list of recognized `SLAPWidget` objects. Upon detection of a new object the correspondent `Madget` class gets instantiated. If a correspondent object already exists, its position and orientation is synchronized to match its `SLAPWidget` pendant. The association of `SLAPWidget` subclasses to `Madget` subclasses is set up via the `SLAPWidgetClasses:forMadgetClasses:` method of the `ActuationFramework` singleton. The list of created `Madget` objects is managed in the `madgetArray` property.

The `Madget` class is initialized with a reference on its correspondent `SLAPWidget` instance and provides methods to synchronize with it. The `footprintItemList` property holds a list of `FootprintItem` objects encapsulating information about the permanent magnets attached to the physical Madget. The standard initialization `initWithSLAPWidget:` creates `FootprintItem` instances analogous to the footprint definition of the associated `SLAPWidget`. This is due to the assumption, that there is a permanent magnet attached to every marker of the Madget.

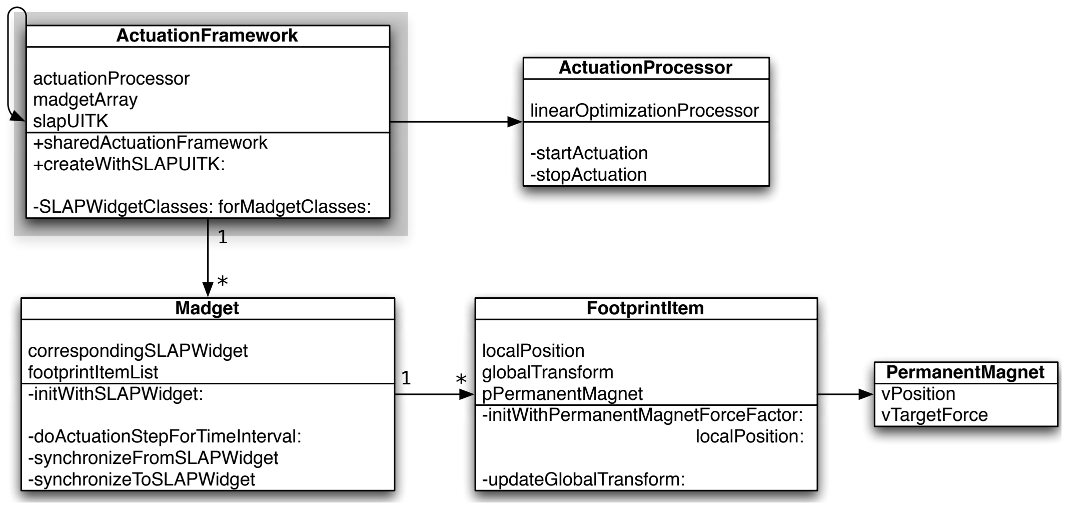


Figure 3.5: Class diagram of the most important classes of the Actuation framework.

Every `FootprintItem` object has its instance of `PermanentMagnet`. The most important variables are three dimensional vectors called `vPosition` and `vTargetForce`. The former specifies the relative position of the represented magnet according to the owning `Madget` instance. The latter vector is used to specify in which direction the represented magnet should be moved.

The `ActuationProcessor` instance is responsible to perform the actuation. When `startActuation` is called a new thread is created working off the following steps:

1. `doActuationStepForTimeInterval:` is invoked on every `Madget` object in the `madgetArray` list of the `ActuationFramework` singleton.

2. A linear equation system is created to solve the assignment of electromagnets to perform the desired actuations specified in the above mentioned `vTargetForce` vectors.

3. The solution of the linear equation system is sent to the hardware layer controlling the electromagnets.

4. The time passed until the last calculation is measured to optionally adjust the frame rate.

5. Start over.

In step one `doActuationStepForTimeInterval:` is sent to a `Madget` instance along with the time interval since the last execution. It should set the `vTargetForce` values as mentioned earlier to let the system actuate the Madget according to its current state. In the second step the `PermanentMagnet` instances of the various Madgets are collected to retrieve a list of all permanent magnets currently present on the tabletop. A solution for the desired actuations is calculated and realized in the third step. The last steps are self-explanatory.

Chapter 4

Low fidelity prototype

"One of the most laudatory terms used to describe an interface is to say that is is 'intuitive'. When examined closely, this concept turns out to vanish like the pea in a shell game and be replaced with the more ordinary but more accurate term 'familiar'."

—Jef Raskin, the Humane Interface

The first section 4.1 describes a storyboard comprehending the necessary steps to construct and integrate a new Madget into the tabletop system. At the end, design alternatives influenced by hardware considerations are discussed.

The next section 4.2 is about the first paper prototype implementing the UI based on one of the storyboards. The evaluation and revision of the UI is discussed in section 4.3.

Section 4.4 is about the evaluation of the second paper prototype which implements the revised UI.

4.1 Storyboard

The storyboard in Figure 4.1 has been the first approach to the UI of the toolkit to get an unbiased system design that reflects the user's mental model more closely. It shows the programming of a Madget with a slider and a LED. The story goes as follows:

1. The Madget prototype is laid onto the tabletop.

2. The system has perceived the markers attached to the Madget and shows a digital representation of it. It differentiates between two types of markers: *Dynamic markers* indicating movable parts and *static markers* attached to the static body. A legend explains the symbols used for the different types.

3. The user draws onto the digital representation the path on which the slider can move.

4. An overlay shows details about the drawn path that was transformed into a straight line: The name of the path and the number of distinguishable positions on this path. Additionally, a button is displayed to control the strength of the rendered friction.

5. When moving the slider, the digital representation updates automatically. This way, the user can test if the position is tracked and updated correctly.

6. + 7. When dragging the digital slider, the system actuates the Madget to keep it synchronized. This is to test whether the system is able to actuate the Madget.

8. + 9. The user draws the symbol of a LED which is transformed into a drag-able icon.

10. The user drags the icon onto the digital representation to plan the installation of a LED into the Madget.

11. Rulers are displayed to help the user maintain the minimal spacing for optimal operation of the Madget.

12. The Madget is rebuilt to hold a LED and enable power transfer to it.

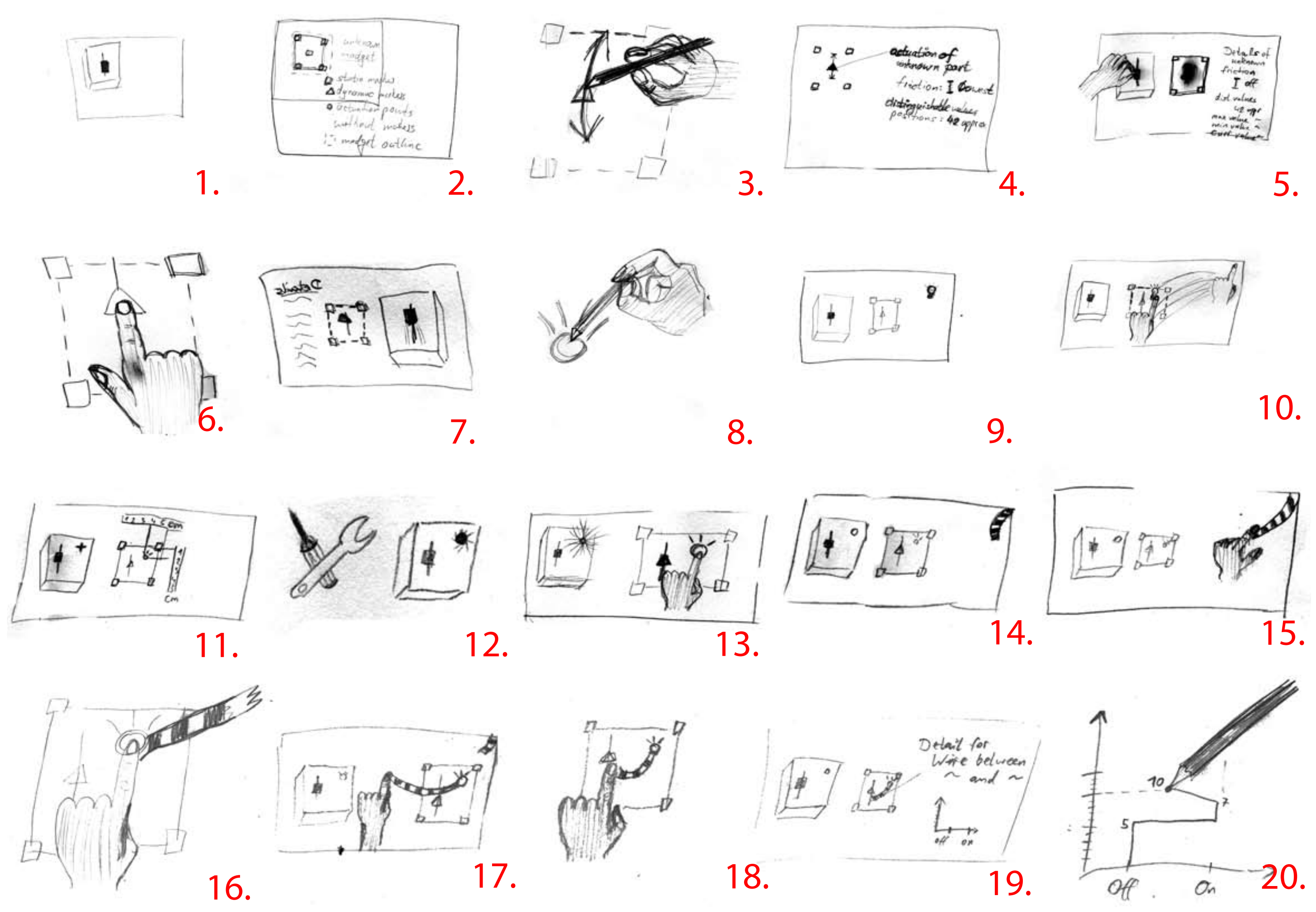


Figure 4.1: Photo of a storyboard incorporating pen input to implement the sketching paradigm.

13. The Madget is laid onto the tabletop and the LED icon in the digital representation is pushed to test the functioning of the LED.

14. A rubber strap appears in the right top corner.

15. - 18. The rubber strap is dragged to connect two items inside the digital representation: The dynamic marker on the path and the LED icon.

19. An overlay shows details about the new established connection between the slider and the LED: The name of the connection and a graph.

20. By drawing into the graph the user defines the mapping between the states of the slider and the states of the LED.

Every time the Madget is laid onto the tabletop the digital representation of it, the *Shadow*, appears.

Definition: *Shadow*

SHADOW:
The *Shadow* is the digital representation of the Madget. It gives visual feedback about the perceived objects of the Madget and its state and provides means to interact with it. The types of items that are displayed depend on the progress of the interaction.

Design considerations

The UI embodies the *liveness* property, i.e., the Madget responds to changes immediately and information about the Madget updates continuously.

The Madget remains onto the tabletop throughout the whole interaction. This way, the Madget directly reacts to changes made to its properties via the UI and the other way around. This *liveness* property, which is inspired by the *Morphic* UI construction kit Maloney and Smith [1995], spares the separation between an edit and run mode. This eases the programming task in two ways:

1. The mental load on the user is reduced by saving the user to contexts.

2. No time is spent on a compile step, which would be necessary to switch from the edit to the run mode.

The interaction incorporates pen input to let the user define graphs or execute commands by drawing gestures. Similar to *VoodooSketch* Block et al. [2008] the user can draw shapes that are transformed into functional widgets like the LED icon to control the power transfer to the LED of the Madget. Additionally, the visual gestalt of the Madget could be drawn with the pen. The sketching of the UI seems more natural and is, thus, preferred over traditional interface builders (see Landay and Myers [1995], Plimmer and Apperley [2004]).

Sketching paradigm allows drawing icons that are transformed into functional widgets.

Rulers are displayed to help the user adhere to the various constraints mentioned in 3.1.2—"Construction constraints of Madgets".

Alternative UI

Ultimately, the available hardware and time factors have to be considered. The present Madgets tabletop does not support pen input and the setup change would cost time that is not available. A Wacom tablet is available and could be used side by side with the Madgets tabletop. But this would make the user switch forth and back between the pen to control the UI and a two-hand interaction to control the Madget. This interrupted work flow could impact the user's experience negatively. Thus, I rejected the sketching idea.

Sketching idea is rejected due to hardware considerations.

The alternative UI is similar to the presented one but uses solely touch input. Instead of sketching, new widgets are dragged from a panel containing predefined buttons and labels. These can be further refined via option dialogs. The definition of paths is done by demonstration according to the *Programming by Example* paradigm in Exemplar Hartmann et al. [2007].

Programming by Example paradigm eases the definition of paths.

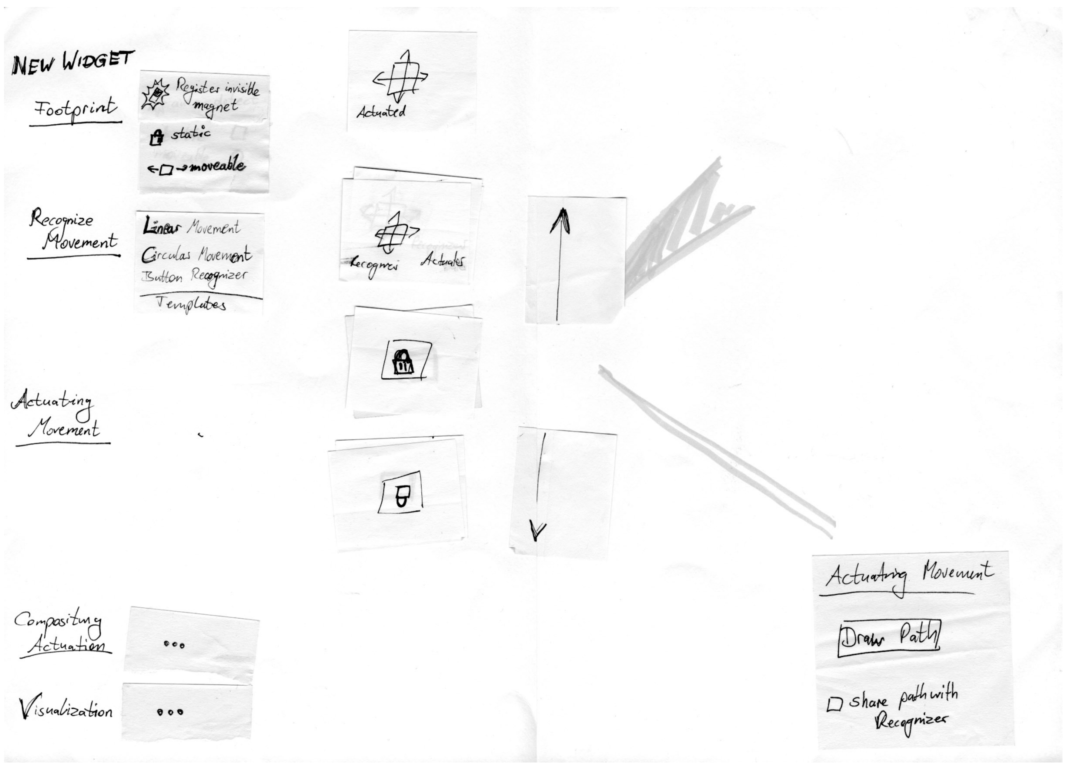


Figure 4.2: Photo of the first paper prototype. The menu on the left currently shows commands for the first two sub tasks. The post-its in the middle represents the digital representation of the Madget. The post-it in the right corner is a pop-up with options for an object of the Shadow.

4.2 Paper prototype of the UI

The storyboard gives a good impression of the comprehensiveness of the task of integrating a new Madget. But the UI is still vague and must be further elaborated and evaluated. This section describes a paper prototype that has been developed to test the appropriateness of the UI and the presented interaction technique.

Figure 4.2 shows a photo of the first paper prototype. As one can see, the graphical user interface (GUI) is divided vertically into two parts: On the left hand side is the *Control area* containing the menu. The *Workbench area* on the other side is initially empty. This area provides sufficient free space, so that the Madget can be laid down there, and to contain the shadow.

Details about a recognized path or options of an actuation are displayed inside pop-ups. These pop-ups are non-modal, i.e., the user is free to control different parts of the toolkit without having to close pop-ups first.

The whole task of programming a Madget is partitioned into several sub tasks. The menu in the Control area presents those tasks in the proposed order of execution; i.e., the first sub task is presented as top most menu item and the last sub task is at the bottom. The steps are the following:

1. Definition of the footprint.
2. Definition of movable parts that are monitored and mapped to numerical values.
3. Definition of magnetic parts that can be actuated by the system.
4. Definition of actuation patterns or conditional events.
5. Definition of the visual gestalt.

The first step is the definition of the footprint of the Madget. As soon as the Madget is laid onto the tabletop the shadow appears and shows a representation of its markers and their status. They are initially not categorized and have to be assigned *static* or *dynamic*. The menu in the Control area provides buttons for both types; although the dynamic type is called *moveable*. The assignment of a marker to one of those types is achieved by pressing the marker and the button for the desired type at the same time.

Magnets attached to the Madget can not be perceived by the system. Therefore, the position of magnets is manually entered by touching the corresponding position inside the Shadow while holding the *Register invisible magnet* button pressed.

The second step comprehends the definition of paths on which *dynamic markers* (objects categorized as movable in the previous step) can move. These paths can be either a straight line, a circle or a movement orthogonal to the tabletop surface; e.g., a push button requires an actuation orthogonal to the tabletop surface. The physical prototype lays on the tabletop while the user holds the button for the desired path pressed and demonstrates the movement on the Madget. Consequently, a new path will be generated from the demonstrated movement.

The third step is used to define which parts of the prototype should be moved and how. The available path types are the same as in the previous step. The path is entered with the finger while one of the button for the desired path type is pressed down. Additionally, there is an option to use pulse width modulation which is necessary for power transfer to a LED.

The fourth step is required to composite actuations in a sequence or trigger them upon certain events. But it is not further specified in this paper prototype.

The last step is about composing the visual gestalt of the Madget by dragging building bricks like rectangles or labels onto the Shadow.

4.2.1 Design considerations

"MODES" AND "MONOTONY":
In his book "the Humane Interface" (Raskin [2000]) Jef Raskin uses the term *monotony* to describe an UI that has only one way for every command to be invoked.

Definition: *"Modes" and "Monotony"*

A *mode* is defined with respect to an action to invoke a command: If the system responds to the user's action differently depending on its state, and the user is currently not paying attention to this state, then the UI is modal with respect to this action. Consequently, *modelessness* is defined as lack of modes.

The UI is designed to be *humane*.

The design of the UI strikes for *modelessness* and *monotony* to be as "humane" as possible — Jef Raskin proposed these properties in his book "The Humane Interface" (Raskin [2000]) to increase the user's performance and positive experience. [1]

To satisfy the monotony property there is only one button for every action of the UI; these are inside the Control area or inside pop-ups.

In order to conform to the modelessness property some commands have been designed as *quasi-modes*; i.e., the mode is only active as long as the corresponding button is pressed down. For example, the button to define movable parts of the Madget must be hold down to demonstrate the movement of a dynamic part. As soon as the user releases the button the demonstration ends and the recorded data is evaluated. This way, the user can not confuse the current mode the system is in.

The *Programming by Example* method is not implemented for defining parts that ought to be actuated. (In contrast to the second menu "Recognize movement".) This is due the fact that the system is not able to track magnets. Under these circumstances, the paths have to be manually entered with the finger. But for reasons of monotony, there should not be two different input methods.

4.3 Evaluation of the paper prototype

The UI of the paper prototype is evaluated in a small user study with two participants. The following sections describe the tasks (4.3.1), presents the results (4.3.2) and discusses improvements (4.3.3).

[1] "I believe that an interface that is both modeless and [. . .] monotonous [. . .] would be extraordinarily pleasant to use. A user would be able to develop an unusually high degree of trust in his habits. The interface would, from these two properties alone, tend to fade from the user's consciousness, allowing him to give his full attention to the task at hand." (p.68)

4.3.1 Tasks

As first assignment the user has to implement the Madget on the left of Figure 4.3 consisting of a rotatable knob, which can be pushed down. The construction is already

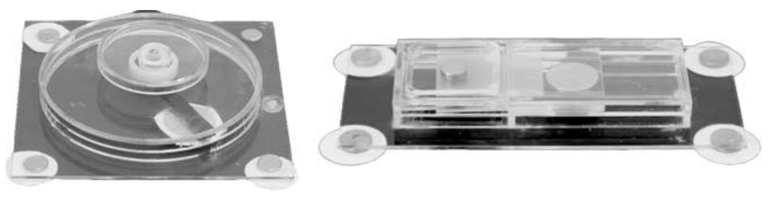


Figure 4.3: Photo of the *Knob* (left) and *Clutch* (right). These Madgets are used in the first evaluation of the paper prototype.

complete and has the required magnets and markers attached to it. To complete this task, the system must measure the angle of the rotatable knob and detect if it is pushed down. After that, a pie chart should be added for visualization.

The second assignment is the integration of the *Clutch*: A Madget consisting of a push button which can be locked by a bar. The bar can not be reached from above but has a magnet attached to it, i.e., if the Madget lays on the tabletop only the system can move the bar. The Clutch must be programmed to register the status of the push button. Also, the path on which the bar is actuated must be defined. After that, the background should change colors to indicate the status of the button.

4.3.2 Results

The participants of the evaluation have experienced the following problems:

The meaning of the "Register invisible magnet" function was unclear. The users did not know that this function has to be used to define the magnet of the Clutch. Since there is no marker attached to the bar its position cannot be recognized by the system.

Markers can be tracked by the system, which allows using the *Programming by Example* paradigm in the "Recognize Movement" menu. Magnets cannot be tracked and thus, require a separate input method in the "Actuating Movement" menu. The users were confused why some paths have to be defined redundantly in both menus and in separate ways. Besides, they were annoyed to have to indicate the type of the path in the "Recognize Movement" menu.

Furthermore, the participants were sometimes unsure how to proceed and found the menu confusing.

4.3.3 Discussion

The UI has been revised to address usability issues and to implement missing functionality. Figure 4.4 shows the new paper prototype for the revised UI, which has two new designated areas: *Storage area* and *Help area*.

The *Storage area* offers a saving and loading function. To help to recover recent work it lists the names and thumbnails of the Madgets.

Help texts were clearly missing to guide the user and explain the functioning of the commands. Therefore, the *Help area* is introduced; to provide a short description of the commands that are currently available.

The *Control area* is redesigned to enumerate the sub tasks. Only the commands of the current step are displayed, the other groups show only the heading. Thus, when working off these groups one after another, the user has less commands to choose from. This narrows the decision tree of the task helping the novice user navigate the system.

Madgets constrained to have a marker on every magnet to abolish the "invisible magnet" method.

The concept of *invisible magnets* was not well understood by the participants. To circumvent this problem the Madget is restricted to have a marker for every magnet. This does not essentially limit the set of realizable prototypes but renders

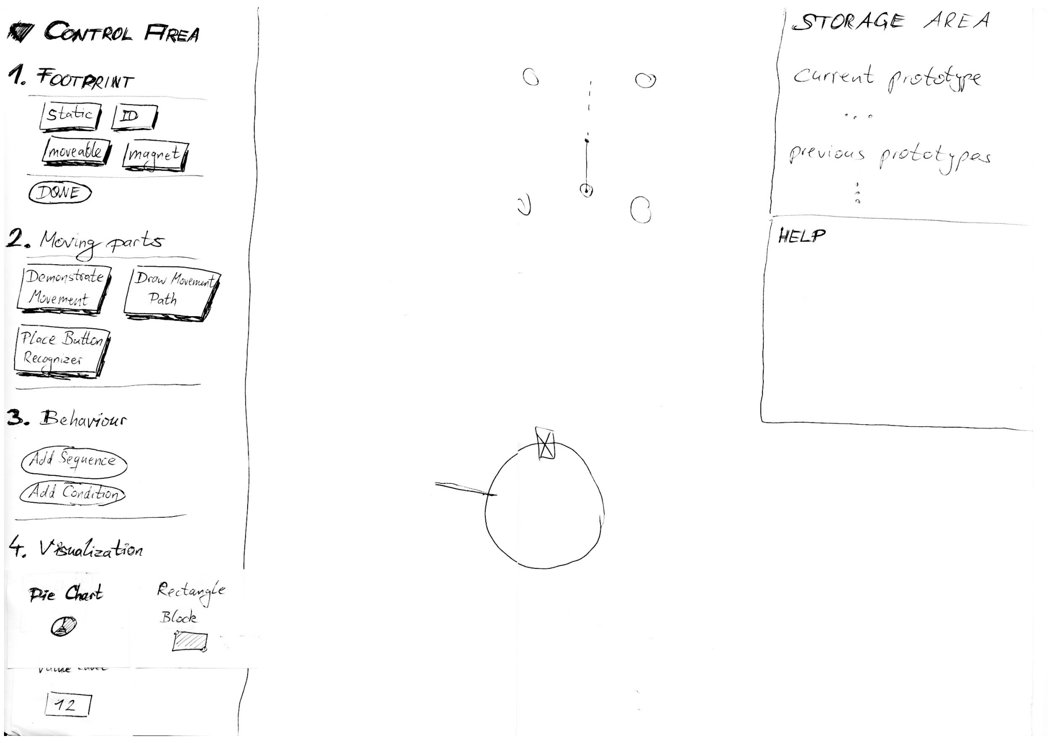


Figure 4.4: Photo of the second paper prototype for the revised UI.

the definition of magnets superfluous. This leads to a couple of advantages: A separate definition of paths for recognizing and actuating movements is not required anymore. Consequently, the two menus (“Recognize Movement” and “Actuating Movement”) are merged into one named “Moving parts”. Furthermore, it is possible to check whether an actuation is successful by comparing the corresponding marker’s position with the target position.

To spare the user to indicate the type of path when they are going to define a path, the method should automatically recognize the drawn shape.

The menu “Compositing actuation” is renamed to “Behavior”. The new name is a better indication for the contained commands and should help the user navigate the UI.

4.4 Evaluation of the second paper prototype

A new paper prototype has been developed to implement the changes of the UI discussed in section 4.3.3. Below, the results of another user study evaluating the revised UI are discussed. The tasks of the study are the same as before.

4.4.1 Results

While users interacted with the paper prototype the following problems occurred:

The participants were annoyed by the help texts. They found it too unstructured to browse and find the right topic for the current menu, e.g., whether buttons of a Madget have to be defined as dynamic markers.

They were missing the option to set the initial state of an actuation or to set the rectangle visualization block to be initially invisible. Furthermore, the UI lacks the possibility to label a position on a path in order to refer to it when programming an actuation.

It is possible to have several pop-ups open at the same time, since they do not close automatically. But then, the GUI gets easily cluttered and one participant complained to lose track of which pop-up is associated to which item.

4.4.2 Discussion

First at all, the missing functions have to be implemented; i.e., the initial state of an actuation, the "invisible color" of an visualization building block and the labeling of positions on a path.

The problem with the lost association of the pop-ups will be solved by implementing visual clues like a simple line connecting the pop-up with the associated object.

The help texts should be revised to provide two sentences for every command in the currently selected menu to support fast browsing. Nevertheless, a more detailed documentation should be provided as hardcopy form. It should cover different aspects of a Madget and how these are to be entered in the UI.

Chapter 5

Medium fidelity prototype

"Never worry about theory as long as the machinery does what it's supposed to do."

—Robert A. Heinlein

This chapter discusses the implementation of the medium fidelity prototype, a software prototype that runs on the Madgets tabletop to enable an evaluation of the system as prototyping tool.

Section 5.1 describes the scenario the medium fidelity prototype is limited to, the Madgets that should be supported and the feature set that is required to realize these.

Section 5.2 is about the architecture of the software prototype. At first, the involved classes in the communication of the toolkit with the SLAP & Actuation framework is explained. After that, the main classes of the software prototype are discussed involving a state machine that controls the whole system. Additionally, the encapsulation and visual representation of the Madget is explained.

In section 5.3 a pilot study is described to evaluate the toolkit. This study serves as preparation for the extended user study described in the next chapter.

5.1 Feature set of the medium fidelity prototype

The medium fidelity prototype is running on the Madgets tabletop to enable an evaluation of the system as prototyping tool. In order to access the hardware and to use the functionality of the frameworks (discussed in 3.2—"Software frameworks") the programming language Objective-C has to be used.

The scope of functionality has to be limited to enable an evaluation as early as possible, but a horizontal or vertical prototype is not goal-oriented: A horizontal prototype would not implement any feature in depth and thus, would not support a fully functional Madget. A vertical prototype would implement only one feature (or a few) completely and spare the remaining ones, which allows constructing at most one type of Madget. For a meaningful study of the prototyping capabilities the system must support several kinds of Madgets. Thus, the sensible solution is to limit the software prototype to a certain scenario that covers several Madgets with a high degree of freedom.

In the paper *Rendering physical effects in tabletop controls* by Weiss et al. [2011] the author presents a set of Madgets with dynamic physical properties. Attributes such as perceived weight, spring resistance, friction, and latching can be adapted through the tabletop actuation. Since this set fulfills the above mentioned criteria the scenario is tailored to the following Madgets:

- Radiobutton with dynamic spring resistance.
- Rotatable knob with dynamic friction.
- Slider with dynamic detents.

The functioning of the first Madget is pictured in Figure 5.1. It is a radio-button, i.e., at all times exactly one button out of a group of several is active. The others must be lifted to indicate their state as inactive. According to the paper "the magnetic force increases quadratically with height, [and thus,. . .] simulates a button containing a progressive spring.” The toolkit should support a way to vary the applied power to the electromagnets to manipulate the perceived resistance of the button.

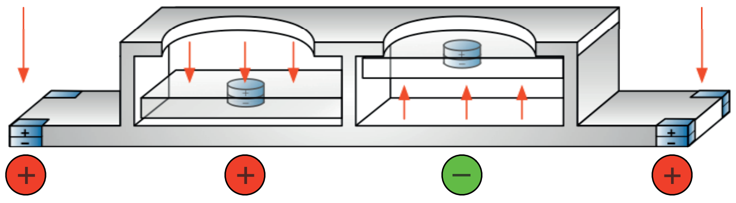


Figure 5.1: Schema of an actuated radio-button Madget. Magnets at the edges are pulled to hold the Madget in place. The left button is pulled to indicate its state as on. The right one is repelled to indicate an off state.

The second Madget is a knob, which can be rotated. Magnets are installed that can be pushed against the rotatable part to function as brake blocks. The assembly is shown in Figure 5.2. According to the paper the perceived friction depends on the normal force and on the materials of the brake. The toolkit should support a way to change the normal force to adjust the friction.

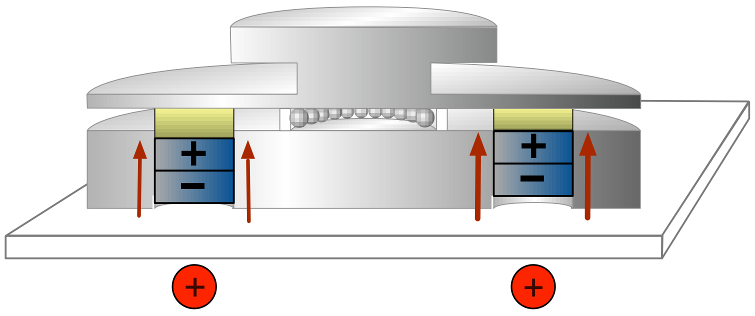


Figure 5.2: Schema of an actuation knob with dynamic friction. Two magnets are installed that can be elevated, so that the brake (yellow) pushes against the disk and causes friction. The red arrows indicate the normal force on these magnets produced by the actuation.

The third Madget is a slider that renders detents by actuating the carriage depending on its current position. Two cases have to be distinguished: In the first case, the carriage lays between two virtual detents and has to be actuated to the closest detent. As Figure 5.3 (a) illustrates, one magnet is negatively charged to repel the carriage from its current position and pulled to a positively charged magnet underneath the virtual detent. Figure (b) shows the second case, when the carriage lays above a virtual detent and is locked in place by positively charging the magnet underneath and negatively charging the adjacent magnets.

The toolkit should support a way to simulate these detents and adapt the step size to different scales.

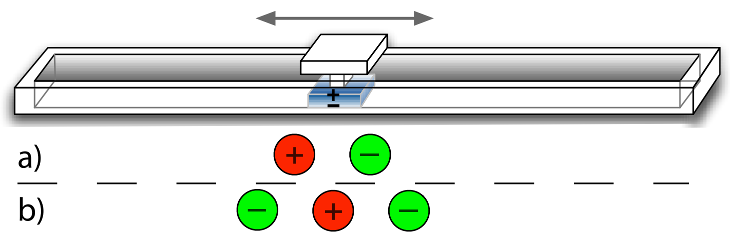


Figure 5.3: Schema of an actuated slider Madget. The carriage is locked in place by positively charging the magnet underneath and negatively charging the adjacent magnets.

Furthermore, the scenario makes the following restrictions to reduce the implementation time and effort: The programming of actuation sequences will not be part of the medium fidelity prototype, because the above listed Madgets do not need this feature. Moreover, a management of several prototypes is not necessary, since it suffices to restart the software in order to construct a new Madget. This spares the implementation of a loading and saving function. Also, a change of the static footprint of the Madget after it has already been defined is not supported. This simplifies the state graph of the toolkit since a couple of edge cases do not need to be taken account of.

5.2 Software architecture

5.2.1 Integration into SLAP-&Actuation framework.

The diagram in Figure 5.4 shows the classes that are involved in the integration of the SLAP & Actuation framework. The highlighted classes belong to the toolkit.

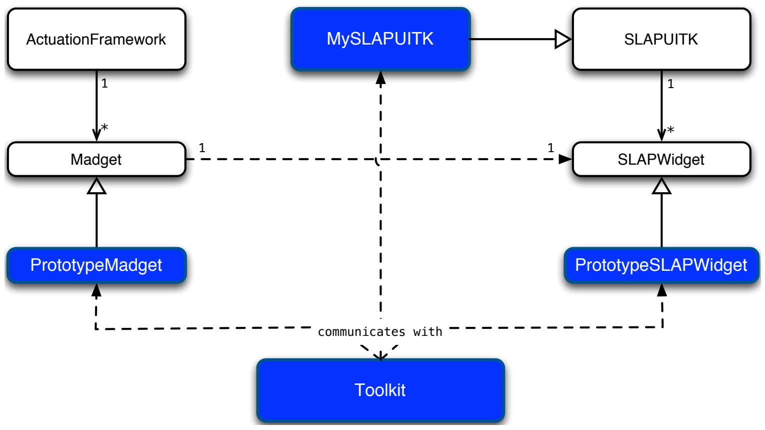


Figure 5.4: Diagram of the classes that implement the communication between the medium fidelity prototype and the Actuation- and SLAP framework. The highlighted classes are from the software prototype. The classes on the right belong to the SLAP framework and primarily manage the footprint recognition. The classes on the left are from the Actuation framework and offer the actuation functionality.

The `SLAPUITK` object manages a list of `SLAPWidget` classes in its *widgetClasses* property. When checking for new widgets this list is traversed to obtain the footprint definitions of the widgets. If one of those footprints match the current touch events the corresponding widget class is initialized and added to the *slapWidgets* list. The position of the widget is kept updated as long as the associated touch events are available.

To make use of the functionality the `SLAPUITK` offers, `PrototypeSLAPWidget` subclasses `SLAPWidget` and is added to the *widgetClasses* list. The

PrototypeSLAPWidget implements the *Singleton pattern*. Hence, the Toolkit object can obtain a reference on the PrototypeSLAPWidget instance when it is allocated by the SLAP framework. Furthermore, the object forwards method calls concerning touch events inside its footprint to the Toolkit. This information is required to obtain the position of dynamic markers.

The Madget class of the Actuation framework provides methods to actuate a tangible. The idea is to have a dedicated Madget subclass for every SLAPWidget subclass. The ActuationFramework singleton offers the *SLAPWidgetClasses:forMadgetClasses:* method to setup this association. It periodically queries the *slapWidgets* property of SLAPUITK to check for new widgets. If it finds a new entry the corresponding Madget subclass is initialized and its position is synchronized to its pendant. Therefore, the PrototypeMadget class is set up as corresponding class for PrototypeSLAPWidget, so that it gets allocated as soon as the user's Madget is detected.

PrototypeMadget subclasses Madget to get access to the actuation functionality the Actuation framework offers. It further has to override the following methods: In the initializer *initWithSLAPWidget:* the positions of the tangible's permanent magnets are specified. There has to be one permanent magnet for every marker of the footprint, which is specified by its corresponding SLAPWidget instance. PrototypeMadget overrides the default initializer to add additional permanent magnets according to the user's prototype. The method *doActuationStepForTimeInterval:* is invoked regularly to tell the object to update its actuation strategy. A vector can be specified for every permanent magnet setting the direction in which it ought to be moved. PrototypeMadget overrides this method to allow the toolkit to dynamically adapt the actuation strategy.

5.2.2 Classes of the toolkit

The class diagram in Figure 5.5 shows the Toolkit class and the classes it interacts with. The Toolkit object is responsible for resetting the state machine and initializing the

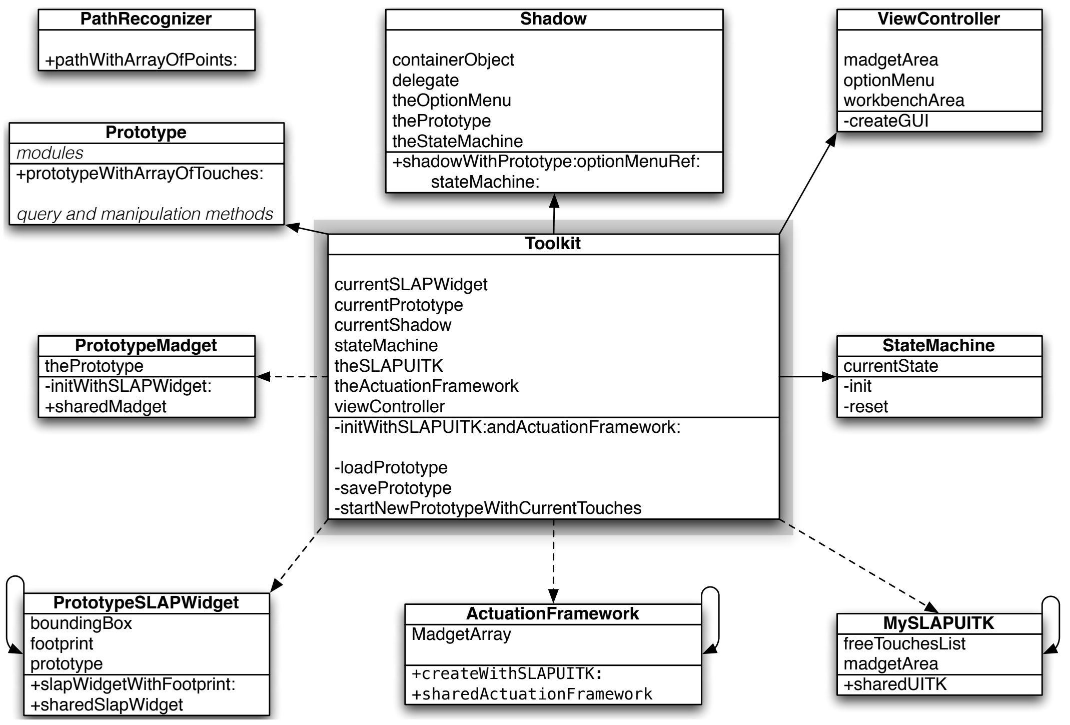


Figure 5.5: UML diagram of the main classes of the software prototype.

ViewController, which creates the GUI. It manages instances of the Prototype and Shadow class to represent the Madget of the user. Furthermore, it has weak references to the PrototypeSLAPWidget and PrototypeMadget objects to use the functionality the SLAP- & Actuation frameworks offer as described above.

The ViewController class creates the GUI of the toolkit and handles its events. A set of basic GUI widgets has been developed like colored rectangles, buttons or dragable elements. Also more complex widgets like menus or pop-ups have been implemented to match the user's expectations of an ordinary GUI. These widgets extend the SLAPGUIObject class to integrate the basic (GUI) functionality the SLAP framework provides (discussed in 3.2—"Software frameworks").

The PathRecognizer class implements functions to recognize the shape of a line or a circle from a list of points. This is used to apply the programming by demonstration paradigm (discussed in 4.1—"Storyboard") when the user demonstrates moveable parts of the tangible to the system. The low input resolution of the tabletop results in unpredictable variations in the recorded position data. Additionally, the user might demonstrate the path only partially. Thus, the resulting shape is likely to deviate from the exact one. The algorithm has to account for these inaccuracies.

The `StateMachine` class is designed to drive the whole software system. This centralized approach should prevent unforeseen states or sinks the user cannot leave. It implements the state graph illustrated in Figure 5.6.

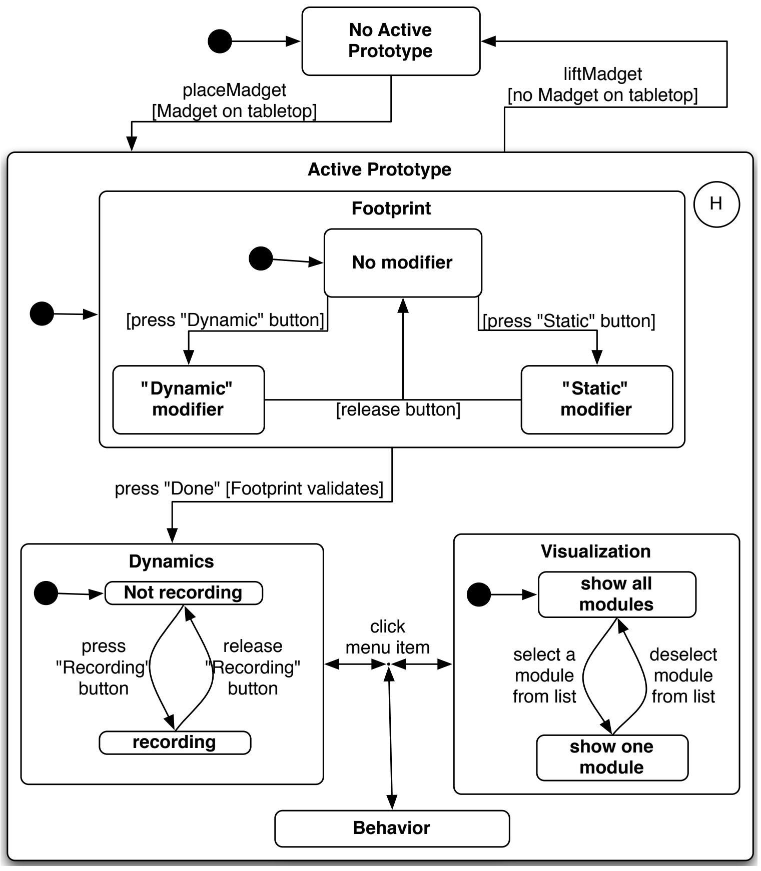


Figure 5.6: UML state chart of the state machine of the software prototype. Two top-level states to distinguish if a Madget is currently on the tabletop. One state is further refined to contain a sub state for every task of the toolkit. Among other things, these are used to determine the currently active menu and the visualization of the Shadow.

The state chart starts in *No Active Prototype* state and switches to *Active Prototype* state as soon as a Madget is present on the tabletop. The latter has a sub state for every task of the toolkit: *Footprint*, *Dynamics*, *Visualization* and *Behavior* state. These have further sub states to handle modifier methods; e.g., to assign the static or dynamic type to a marker. The *Footprint* state is the entry state, since the interaction starts with the definition of the footprint. If the user presses "Done" the footprint definition is validated and if successful, switches to *Dynamics* state. From there on the user can switch freely between the sub states of *Active Prototype* but is prevented from returning to *Footprint* state. If the Madget is not recognized anymore, e.g., because it was lifted from the tabletop, the state machine returns immediately to *No Active Prototype*. When the Madget is present again the previously active sub state of *Active Prototype* is entered.

The `Prototype` object represents the tangible control the user constructs. It encapsulates all information for a functional Madget like footprint, visual gestalt and paths of movable parts. It is discussed in more details in section 5.2.3—"Encapsulation of the Madget prototype".

The `Shadow` class implements the *Shadow* as introduced in the UI of the low fidelity prototype. It is discussed in more detail in section 5.2.4—"Representation of the Madget prototype".

`MySLAPUITK` is a subclass of `SLAPUITK` of the SLAP framework to access the *touchedObjects* variable which keeps track of all `SLAPAlignableObject` instances currently associated with touch events. Since the GUI of the toolkit uses subclasses of `SLAPAlignableObject` this variable is used to distinguish touch events that are associated to GUI elements from free touches, i.e., non-associated ones. This is required to determine the footprint of a new Madget. As soon as the footprint is defined it is detected as `SLAPWidget` instance as discussed earlier (see section 5.2.1—"Integration into SLAP-&Actuation framework").

5.2.3 Encapsulation of the Madget prototype

Figure 5.7 shows the `Prototype` class and all related classes it manages to encapsulate all information a Madget consists of. There is a specialized class for every component of a Madget:

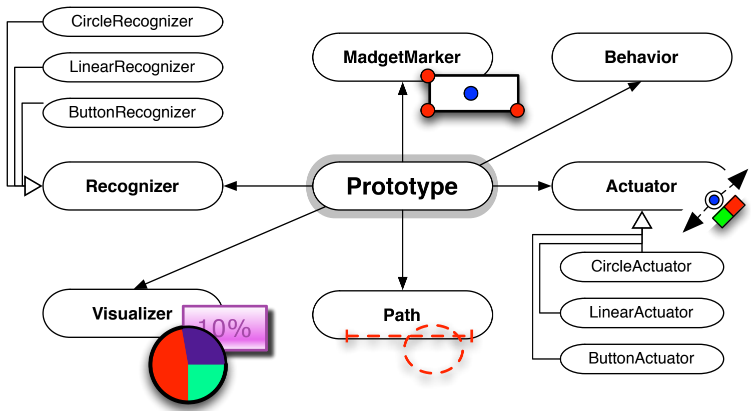


Figure 5.7: Diagram showing all required classes to encapsulate the Madget prototype.

The `MadgetMarker` objects represent marker and magnets attached to the tangible.

A `Path` object is associated with a `MadgetMarker` to encapsulate the path on which the marker moves.

A `Recognizer` object uses an instance of the `Path` and `MadgetMarker` class to map the position of the marker to a numerical value with a function f:

$$f : (x, y, z) \to \Re^{+}$$

The three dimensional vector (x, y, z) represents the position of the marker. The third component $z \in \{0, 1\}$ is used for height actuation, i.e. orthogonal to the tabletop surface. The only distinguishable states are 1 for laying on the surface and 0 if the marker is not perceived or lifted respectively. The subclasses `CircleRecognizer`, `LinearRecognizer` and `ButtonRecognizer` implement the mapping for the circle, line or button type of path; e.g. `LinearRecognizer` implements f in such a way, that

0 is the result if the marker is located at the start of the associated line and 1 if it is at the end. The other positions on the line are associated accordingly.

The `Actuator` class extends the `Recognizer` class with a method to actuate the position of the marker according to the inverse mapping f^{-1}. The subclasses `CircleActuator`, `LinearActuator` and `ButtonActuator` implement this method for the corresponding types of path.

`Visualizer` subclasses implement the building blocks the visual gestalt of the Madget is composed of. The `RectangleVisualizer` is an exemplary subclass that draws a rectangle area in the specified color. It can be used to set a background for the Madget.

`Behavior` instances encapsulate conditional events or actuation patterns. For example to change the background color of a Madget depending on the position of its slider a connection between an `LinearRecognizer` and `RectangleVisualizer` instance is required. A `Behavior` object would store this connection, query the `LinearRecognizer` to obtain a numerical value from the position of the slider and set the color property of the `RectangleVisualizer` accordingly.

5.2.4 Representation of the Madget prototype

The `Shadow` object implements the digital representation of the Madget as presented in the UI of the low fidelity prototype in 4.1—"Storyboard". Figure 5.8 shows its auxiliary classes to split up the visualization task according to separate subsets of the Madget's properties. Since not all of these properties are relevant at all times, the `Shadow` object initializes only one class at a time. To ease the switch between different visualization strategies and to keep the code more structured, the `Shadow` class implements the *Delegates pattern* and the visualization classes have to adhere to the `ShadowVisualization` protocol.

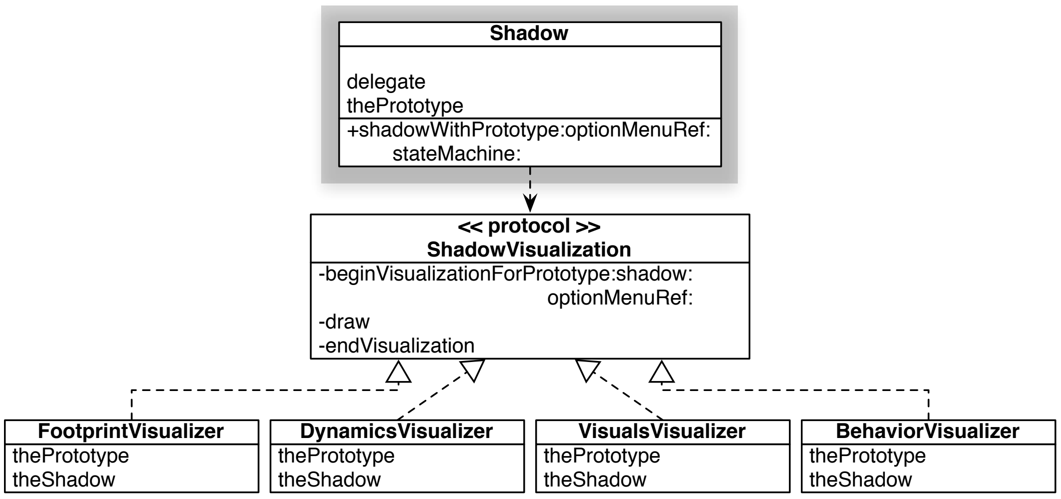


Figure 5.8: UML diagram showing all required classes to realize the digital representation of the Madget. The `Shadow` class implements the *Delegates* pattern to allow switching the visualization strategy as required. The different visualization techniques are implemented in separate classes implementing a common protocol.

The `FootprintVisualizer` is used when the footprint of the Madget is defined. It displays all markers that make up the preliminary footprint. The user can select these representations to define the static and dynamic part and markers that were mistakenly added and should be ignored.

The `DynamicsVisualizer` shows the paths on which dynamic parts move. Additionally it provides options to e.g. add detents or change the spring resistance of a button.

The `VisualsVisualizer` shows the graphical building bricks that were added to the visual gestalt of the Madget. It provides options to change their attributes like color and opacity.

The `BehaviorVisualizer` lists the properties of the Madget that can be combined. On the one side are Recognizer items and on the other side are Actuator and Visualizer items. Since the Recognizers supply data about the Madget's state, and Actuators and Visualizers take data to produce feedback, only connections from Recognizers to Actuators or from Recognizer to Visualizers are valid.

5.3 Evaluation of the software prototype

Before the actual user study (see chapter 6—"Final evaluation") a pilot study with two participants will test the setup and toolkit.

The photo in Figure 5.9 shows the setup for the pilot study. On the table in the back are laying materials to assemble the Madget like glue, markers and magnets. In the front is the Madgets tabletop. In order to use the laser cutter the participant has to go to the workshop.

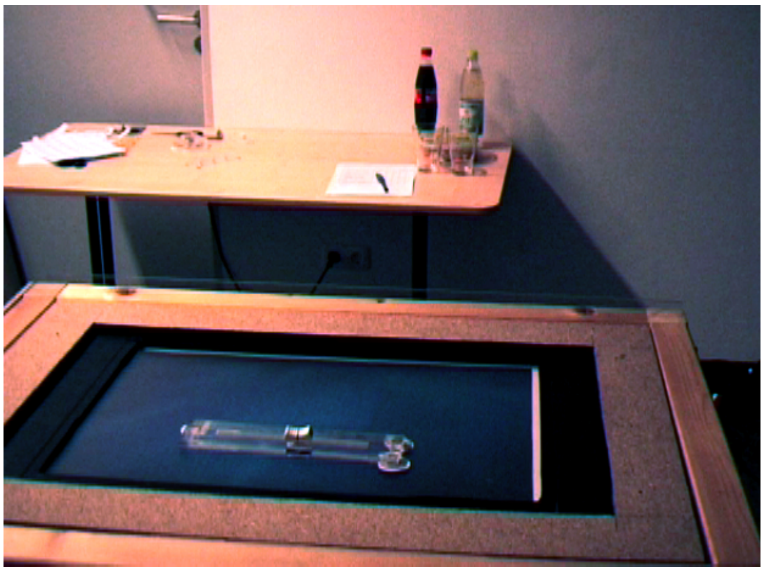


Figure 5.9: Photo of the setup for the pilot study of the medium fidelity prototype. In the front is the Madgets tabletop. On the table in the back are markers, magnets, a scissor and glue to assemble the Madget.

5.3.1 Tasks

The participant gets a brief introduction into the tabletop technology, the assembly of Madgets and the functions of the toolkit. Then, the slider Madget is presented to him.[1] To

[1]The Madgets in this section refer to the description of the scenario in 5.1—"Feature set of the medium fidelity prototype".

complete the first task the user has to work off the following steps:

1. Define the static footprint of the Madget and the dynamic markers.
2. Demonstrate the movement of the slider.
3. Add a rectangle in the visualization step and resize it to cover the whole Madget.
4. Optionally adapt the color property of the rectangle.
5. In the behavior menu connect the recognizer component to the visualization component, so that the opacity of the background is defined as function of the slider's position.

The second task is the constructing and programming of a radio-button Madget. At first the functioning of this Madget is explained. Then, the user is introduced to the vector drawing software *CorelDRAW*[2] with which the blueprint is created. The assembly out of several layers of $2mm$ thick acrylic glass is explained using the example of the slider Madget. A set of guidelines is provided comprehending:

- The maximal size of a Madget.
- The minimal distances of magnets and markers (as discussed in 3.1.2—"Construction constraints of Madgets").
- The advise to construct as lightweight as possible and reduce friction as far as possible.

To complete the second task the user has to work off the following steps:

1. Design the blueprint of the Madget using CorelDRAW.

[2] CorelDRAW Graphics Suite homepage, http://www.corel.com/

2. Use the laser cutter to print the blueprint out of a $2mm$ thick acrylic glass.
3. Glue the parts together and attach magnets and markers.
4. In the footprint menu: Assign the static and dynamic marker type appropriately.
5. In the dynamics menu: Define the buttons.
6. In the behavior menu: Connect the recognizer component of the first button to the actuation component of the second and vise versa, so that one button is lifted if the other is not.

5.3.2 Results

Table 5.1 shows the performance times of the user. The programming task of the radio-button could not be completed due to a software crash. This error has been resolved in the next iteration.

Introduction	20 minutes
Programming of slider	30 minutes
Instructions for radiobutton	5 minutes
Design blueprint	45 minutes
Laser cutting	7 minutes
Assembly	20 minutes
Programming	—
	$\sum$ 2h 7m

Table 5.1: List of tasks of the pilot study with measured duration.

During the test the user was confused what to do in which step, was unsure about the function of some buttons and whether he had completed all necessary steps. Also the outcome of some actions taken in the visualization and behavior menu was not clear.

When the user tried to control elements of the GUI the system sometimes misinterpreted the user's touches for the footprint of the Madget. As a consequence, the digital representation of the Madget was repositioned and the user was unable to execute the desired GUI function.

The GUI was cramped with pop-ups. Many actions open up new pop-ups which are not closed automatically, even if the user finishes a sub task and switches to a different menu.

The user did not pay attention to the polarization when glueing magnets to the static body of the Madget. The toolkit actuates these magnets homogeneously in order to hold the tangible in place. If these are installed differently the Madget is unintentionally moved around.

5.3.3 Discussion

A documentation is missing that describes which actions have to be taken in which step. This can help the novice user navigate the UI.

The user's confusion during the interaction points to a missing connection between the action he has taken and a perceivable result. When investigating the problematic cases a lack of feedback is noticeable. An example is the case of the visualization menu. When the user has clicked the button to add a rectangle it was added as a new entry to a list located inside a pop-up. The Madget's visual gestalt updated to show the new rectangle but that is not the user's locus of attention. Instead it is the Shadow which shows the new rectangle only if it is selected in the list. This is a mode error: The UI shows the desired result if it is in a different mode, but the current mode is not the user's locus of attention. To solve this problem the new list entry gets highlighted for two seconds to draw the user's attention to that particular area. Another solution would be to merge the different modes of this menu and let the Shadow always display all added building blocks.

The issue of the touch input that is mistaken with the Madget's footprint is caused by the SLAP framework. The "widget detection" algorithm always considers all touch events when looking for a matching footprint even if a touch event is located above a GUI element or `SLAPGUIObject` respectively. Furthermore, the algorithm is considering Madgets whose "footprint touches" are still available. While these decisions might be advantageous for the SLAP tabletop, they are leading to problems for the Madgets tabletop. This is due to the fact, that the algorithm is designed for a high input resolution allowing to distinguish between fingers and markers. To solve this problem the algorithm is restricted to touch events inside a dedicated area. The Madget has to reside inside this area further called *Madget area* (see area 6 in Figure 5.10).

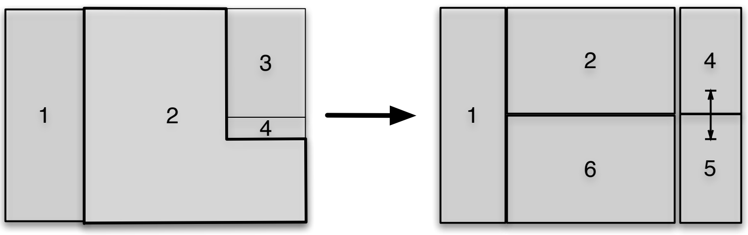


Figure 5.10: Revision of the GUI layout with different areas: (1) Control area (2) Workbench area (3) Storage area (4) Help area (5) Details area (6) Madget area.

Since the polarization of magnets is important for the correct functioning of Madgets a "pivot magnet" will be handed out during the test. This is a magnet whose sides are colored in red and green. The above mentioned documentation is extended with a note to always make the red side attract the installed magnets from below the Madget. This way, the orientation is as the software expects.

A new GUI widget is introduced in order to prevent the creation of too many pop-ups. It can be accessed by the `Shadow` class to place options inside that widget instead of pop-ups. This widget is shared among all visualization delegates of the Shadow, so that the content is updated when the state of the toolkit changes. The area this widget occupies is called *Details area* (see area 5 in Figure 5.10).

The *Storage area* is removed because loading and saving is not supported by the medium fidelity prototype.

Chapter 6

Final evaluation

This chapter describes the final evaluation of the toolkit with seven participants and discusses the results.

6.1 Tasks

The setup is based on the previous evaluation (see 5.3—"Evaluation of the software prototype"). But the tasks include Madgets that are described in the feature set of the medium fidelity prototype (see 5.1—"Feature set of the medium fidelity prototype").

A run takes approximately four hours; table 6.1 shows the duration estimates of the separate steps. The participants of the pilot study had difficulties constructing the Madgets without any assistance. Therefore, the second task is changed to an optimization task: The user will see an imperfect version of Madget C and is told its shortcomings, so she knows what to optimize. The *Rotatable knob with dynamic friction* is taken out of the study, since it seems to be too complicated to be completed in the allocated time.

The tasks are now as follows:

1. Introduction to the idea of ProMadgets, the technology of the tabletop and Madgets.

Introduction		15 minutes		
Documentation		15 minutes		
Demo		12 minutes		
1st task				
	Explain Madget B	7 minutes		
	Programming task	30 minutes		
Demo				
	CorelDRAW, Laser Cutter	10 minutes		
	Drawing of Madget A	5 minutes		
2nd task				
	Explain Madget C	7 minutes		
	Design blueprint	60 minutes		
	Lasercut	7 minutes		
	Assembly	20 minutes		
	Programming	15 minutes		
1st iteration				
	Revise blueprint		10 minutes	
	Lasercut		7 minutes	
	Assembly		10 minutes	
	Programming		5 minutes	
2nd iteration	Programming			5 minutes
		3h 23min	3h 55min	4h 0min

Table 6.1: List of all tasks of the user study for the final evaluation with estimated duration.

2. The user is asked to read the documentation, which includes guidelines on how to create Madgets and explains the toolkit (see A—"Materials of the user study" for a copy of the user test manual). It is assured that the user has understood everything and that the system is being tested and not the user.

3. Presentation of the toolkit's functions by demonstrating the programming of Madget A.

4. The user is told the functioning of Madget B and is asked to program it accordingly. (1st task)

5. Explanation of the vector drawing software *CorelDRAW* with the help of the blueprint of Madget A. Demonstration of the usage of the laser cutter.

6. The user is told the functioning of Madget C and is

asked to construct and program it accordingly. (2nd task)

Table 6.2 shows the combinations of Madgets used in the user study. Since the Detent slider and Radio-Button are more complicated than the other Madgets, these are preferably chosen as Madget C. This way, the level of difficulty is incrementing from first to second task.

Participant ID	Madget A	Madget B	Madget C
1	Button	Slider	Knob
2	Slider	Button	Knob
3	Slider	Knob	Detent slider
4	Knob	Slider	Button
5	Knob	Slider	Detent slider
6	Knob	Slider	Detent slider
7	Knob	Detent Slider	Radio-button

Table 6.2: List of all permutations of Madgets used in the different runs of the final evaluation. The second and third Madgets were randomly chosen.

6.2 Participants

Table 6.3 shows the background information about the participants. The average age is 26 years (SD = 2.83). All participants have a technical field of study.

Participant	Age	Field of study
1	29	Computer science
2	25	Mechanical Engineering
3	22	Mechanical Engineering
4	26	Computer science
5	29	Physics
6	23	Computer science
7	28	Computer science

Table 6.3: List of all participants of the final evaluation with their age and field of study. All users are tech-savvy.

6.3 Results

The first task was always completed successfully but performance times varied strongly (*AVG: 11.86 SD: 5.52*).

In Figure 6.1 the programming times for the first task are contrasted with the iterations of the second task. The graph shows a monotonous decrement of the duration for all participants.

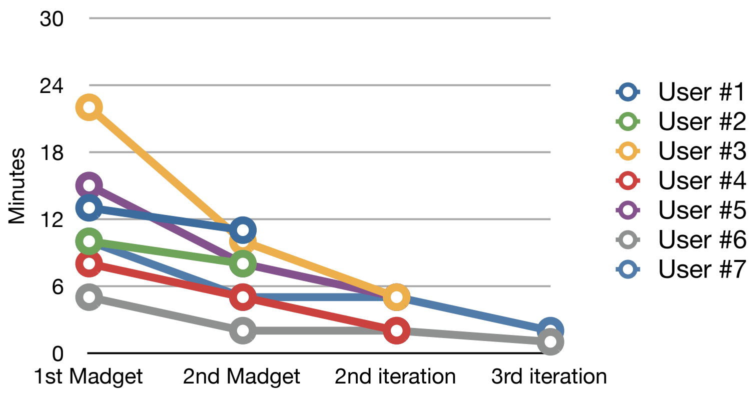


Figure 6.1: Graph showing the programming duration for both tasks of all participants. The values are monotonously decreasing indicating a good learning curve.

The second task could always be completed. The graph in Figure 6.2 shows the time users spent on average on the different aspects in percent.

The quality of the final prototype and the number of iterations are listed in table 6.4. While the Knob and Detent Slider were always fully functional, the radio-button Madget was only partially functional. The problem of the latter Madget was mostly that the tabletop provided insufficient magnetic force.

The user's performances are contrasted with each other in Figure 6.3 to bring out the individual differences. The construction time averaged to 76.86 minutes (*SD* 17.04) and programming time averaged to 10.14 minutes (*SD* 3.58).

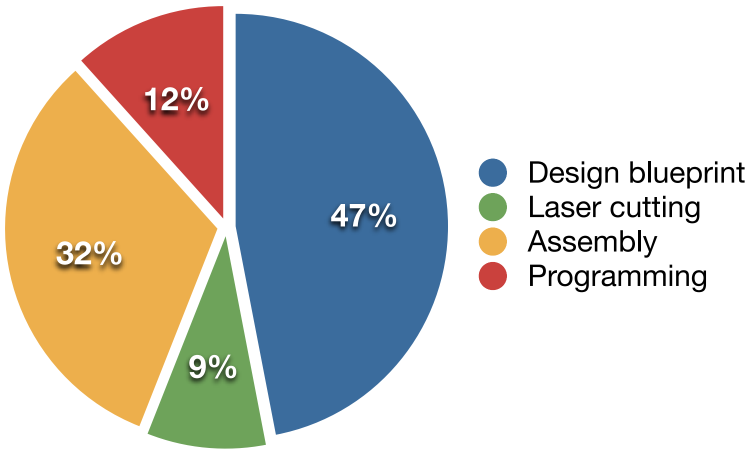


Figure 6.2: Time users spent on average on different aspects of the second task in percent. The yellow area includes the assembly of the Madget and all repairs. The constructing proportion (88%) is dominating the prototyping process.

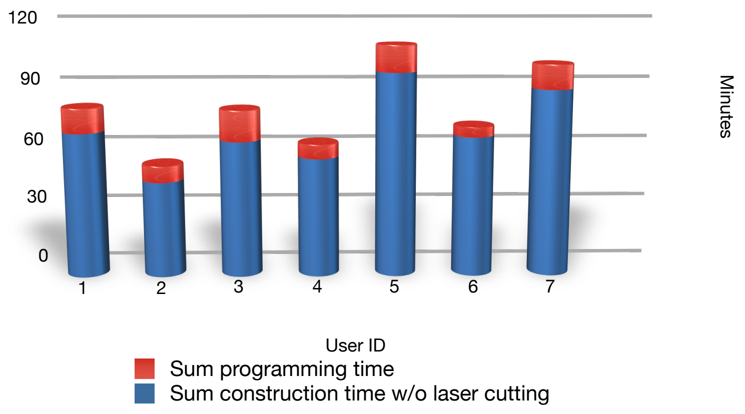


Figure 6.3: The graph contrasts the time users spent on programming (red) to constructing (blue). The time the laser cutter was used is excluded since it is independent of the user's performance. The proportions already indicate that the constructing dominates the prototyping process.

The result of the questionnaire is shown in Figure 6.4.

UserID	#Construction iterations	#Programming iterations	Quality of final prototype
1	1	1	Fully functional
2	2	1	Fully functional
3	1	1	Fully functional
4	2	2	Partly functional
5	2	2	Fully functional
6	4	3	Fully functional
7	3	3	Partly functional

Table 6.4: List of all participants of the final evaluation showing the number of iterations they required to complete the second task and the quality of their final prototype

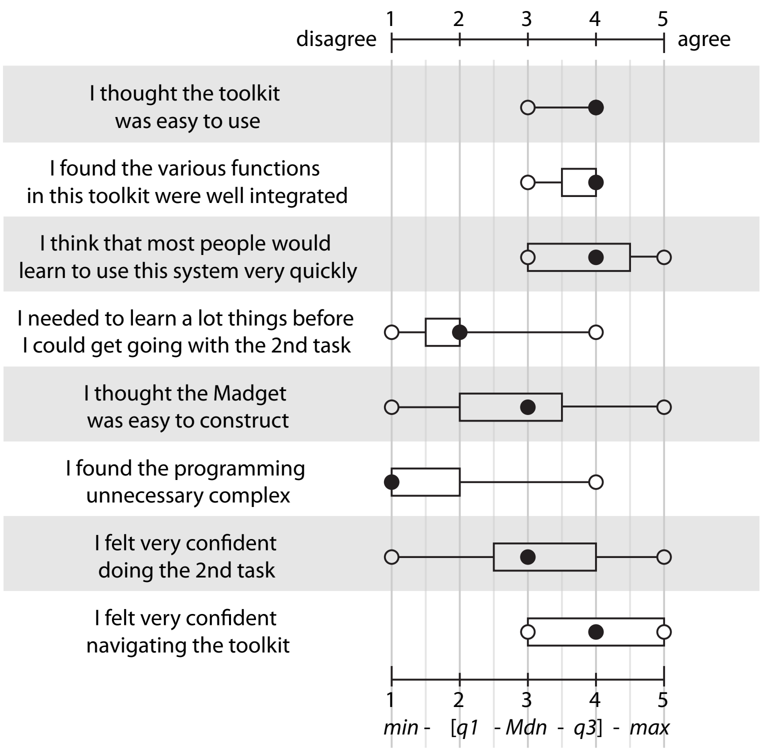


Figure 6.4: Visualization of the results of the questionnaire.

6.4 Discussion

Although the study is limited to a small set of Madgets it already shows that it is possible to prototype a fully functional Madget in under two hours.

The results show that the toolkit offers a low threshold.

The usability goal of the toolkit to have a low threshold is accomplished according to question one and three of the questionnaire: All users found the toolkit easy to learn and to use. The graph in Figure 6.1 supports this result: The programming times are monotonously decreasing, which indicates a good learning curve. A dependence on the level of difficulty of the Madget can be precluded, since the types and the order has been randomized (see table 6.2).

The revised help texts better assist the users navigating the UI.

Positive results of question two ("I found the various function in this toolkit were well integrated") and eight ("I felt very confident navigating the toolkit") indicate that the GUI is well structured. The revision of the help texts was successful, since no users complained about getting lost anymore. According to the result of question six ("I found the programming unnecessary complex") only one user found the programming unnecessary complex.

The teething problems of the tabletop hardware negatively influences the prototyping process.

The tabletop hardware turned out to be the bottleneck of the prototyping process. Too much time was spent on refining the Madget to circumvent its shortcomings. Only little time was spent on programming (12%, see Figure 6.2). More construction iterations (AVG 2,14; SD: 1,07) were required than program iterations (AVG: 1,86; SD: 0,9). Especially in the case of the radio-button the Madget had to be highly optimized to allow the elevation of the buttons.

A revision of the tabletop would ease the prototyping process in several ways:
Firstly, a stronger magnetic force can overcome more friction, thus, less optimization of the Madget is required. Secondly, a higher input resolution allows for recognition of smaller markers, which can be attached more closely together. Consequently, the whole Madget can be smaller scaled and, thus, will be lighter.

A further improvement of the prototyping process can be

The usage of the Actuation framework saves implementation time and effort but its limitations are also negatively influencing the prototyping process.

achieved by a revision of the toolkit: The incorporation of the Actuation framework saved a lot of time implementing the medium fidelity prototype. But the actuation technique it offers is too restrictive and needs to be extended. Currently, only a vector for every permanent magnet can be specified and the electromagnets are controlled automatically. A specification for the radius of influence for every permanent magnet is missing, influencing the amount of electromagnets that are used to actuate a certain permanent magnet. This would enable to drive several permanent magnets as one. For instance, the button plate of the radio-button Madget requires a permanent magnet on each edge to be elevated smoothly. The Toolkit currently does not support that.

Further studies are required to test a correlation of prototyping success and experience with the deployed tools.

The result of question five (“I thought the Madget was easy to construct”) shows how differently the participants assessed the difficulty of constructing a Madget. The value ranges from *1 - complete disagreement* to *5 - complete agreement* with first quartile on 2 and third quartile on $3,5$. One reason for that high variance is the difficulty with which users learned to use the new tools: Laser cutter and CorelDRAW. (As the range of the answer to question four shows.) Another reason is the experience with engineering drawings. Although the correlation was not found to be significant ($\tau = 0.0563$, $p = 1$), users with prior knowledge in that field have shown less problems with the design of the blueprint and the adherence to the guidelines.

Chapter 7

Summary and future work

In this final chapter I will summarize my research, state the contributions to the prototyping community and give an outlook on future research ideas.

7.1 Summary and contributions

I started with an introduction to prototyping in the field of interactive tabletops and tangible user interfaces. After discussing the appropriateness of recent systems in terms of prototyping I summarized the strengths of the Madgets tabletop. Then I pointed to the lack of toolkits (for interactive tabletops) that not only allow prototyping the physical shape of tangibles but also support the acquisition and processing of data.

For a toolkit to explore the prototyping capabilities of Madgets the following requirements have been identified: A low threshold for the registration and integration of the new tangible controls and a high ceiling to support all degrees of freedom the technology offers.

Based on the Madgets presented in the paper *Rendering physical effects in tabletop controls* by Weiss et al. [2011] a software prototype has been implemented that supports the realization of Madgets rendering physical resistance and dynamic detents. In order to evaluate this toolkit seven participants had four hours to construct and program several Madgets with increasing level of difficulty. One task was the construction and programming of a Madget from scratch, which comprised the design of the blueprint, the operation of a laser cutter and the assembly of the Madget.

As the test results show, the initial goal of a low threshold has been accomplished, so that a fully functional tangible control could be built from scratch in under two hours. But the study also indicated areas to improve for future work.

To sum up, this work contributes the investigation of the prototyping capabilities of the Madgets tabletop and gives empirical evidence of the usability and usefulness of this prototyping technique. This work should be beneficial to the prototyping community as it provides insights that are applicable to other tabletop technologies too.

7.2 Future work

The user study showed that the shortcomings of the Madgets tabletop hardware and the limitations of the toolkit have a negative influence on the success of the prototyping process.

In future work, the tabletop hardware needs to be revised to increase the maximal applicable magnetic force, the actuation resolution and the input resolution. This will reduce the amount of optimization that is required for some Madgets in order to function properly.

Also, the visual programming capabilities of the toolkit should be extended to support the full degree of freedom the tabletop hardware offers. This comprehends arbitrary pulse width modulation to, e.g., enable power transfer to an LED embedded into a Madget. Programmable actuation sequences allow the realization of more complex tangible controls, which would increase the usefulness of the tool.

Further studies are required to investigate the influence of prior knowledge in technical fields like experience with laser cutter or engineering drawings on the prototyping process. During the study, this effect has been observed but have not been found significant.

Appendix A

Materials of the user study

List of all documents that will follow:

1. User test manual.

Idea of Madgets

Madgets are input/output devices for the actuated tabletop, like keyboard and mouse for the desktop PC. While the keyboard and mouse are standard controls, Madgets are specialized controls to allow the user to solve certain tasks more efficiently.
Furthermore, they can be used as output and input devices at the same time.

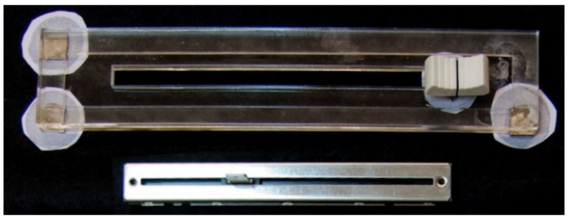
E.g., the knob in the picture on the right can be rotated to scroll through a video. Additionally, the timeline of the video is displayed as pie chart.

Why Madgets?

Madgets work without any electronics, which makes them easy to construct and thus, optimal to prototype. Ideas of new IO devices can rapidly be tested.
The ProMadgets toolkit is used to awake the Madgets to life without requiring any programming.
Physical properties can be rendered dynamically saving some design cycles refining the prototype.

What are Madgets?

Madgets are made of transparent acrylic, markers and magnets.
Markers are round pieces of paper attached to the corners, so that the position and orientation of the Madget can be tracked.
Additionally, markers are attached to moveable parts to perceive the state of the Madget;
E.g., the position of the slider in the picture on the right is perceived via the attached marker. The path on which it is able to move is used to derive its value.



The next section describes the process of constructing a Madget in more detail.

Madget construction

As one can see in the picture on the right, a Madget is made up of several layers of transparent acrylic. The layers are cut out of a 2mm acrylic plate using a laser cutter.
A blueprint must be drawn for the laser cutter to cut out the right parts.

The construction must adhere to several rules explained on the next page to achieve best performance of the produced Madget.

After the laser cutter has finished, the parts are glued together and markers and magnets are attached.

Guideline for Madget construction

For best performance, the construction must adhere to the following rules:

- ☑ Limit size to 25cm in width, 12cm in height
- ☑ Build rectangle ground
- ☑ Build as light weight as possible
- ☑ Moveable parts should slide smoothly
- ☑ Attach exactly 3 markers and magnets on the corners of the Madget in “L” shape. Attach those underneath the Madget, so that the markers touch the tabletop surface
- ☑ Keep distance between all markers at least one marker size
- ☑ Keep distance between all magnets at least one marker size

Programming the Madget

Every Madget can be integrated working off these four steps:

1. Definition of the footprint
   - Lift Madget from table before pressing “Start new prototype”. Hold button pressed while you lay the Madget onto the Madget Area. Move the Madget a bit around, then release the button.
   - Recognized markers are represented as buttons in the workbench area;
   - Categorize the markers into static and dynamic by holding down the command button for the desired type (in the Control Area) and the pressing the marker representations (in the workbench area) at the same time.
   - If not all markers are recognized, try moving the Madget around while holding the “start new prototype” button pressed.
   - Superfluous markers like faulty recognized ones can be set to be ignored!

2. Definition of dynamics
   - Demonstrate paths on which the markers declared as dynamic move. The Madget must be recognized, i.e., have a blue corner around it. If not, move it around until it gets recognized.
   - Hold “Demonstrate movement” pressed to demonstrate linear or circular movements. The recorded path is indicated as green dotted line.
   - Hold “Define button” pressed to select a marker declared as dynamic to move orthogonal to the tabletop surface. Simply choose one of the dynamic markers when they appear.
   - Details about the defined dynamics can be revised in the object inspector

3. Placement of visual elements
   - Press “Add rectangle” to add a rectangle shape to the Madget.
   - Added shapes can be edited by clicking the corresponding button in the object inspector. The edit mode starts and the shape can be resized, scaled, rotated and tinted.
   - If the Madget is recognized (blue border), you will immediately see the changes. Move the Madget around if not recognized.

4. Programming of autonomous behavior
   - You create autonomous behavior by clicking one element of the input side and of the output side at the same time.
   - Created connections are listed in the object inspector and can be disconnected by clicking on the corresponding button in this list.

Bibliography

Eric Akaoka, Tim Ginn, and Roel Vertegaal. Display-objects: prototyping functional physical interfaces on 3d styrofoam, paper or cardboard models. In *TEI '10: Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction*, pages 49–56, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-841-4. doi: http://doi.acm.org/10.1145/1709886.1709897. URL `http://portal.acm.org/ft_gateway.cfm?id=1709897&type=pdf&coll=Portal&dl=GUIDE&CFID=94160061&CFTOKEN=65157542`.

Daniel Avrahami and Scott E. Hudson. Forming interactivity: a tool for rapid prototyping of physical interactive products. In *DIS '02: Proceedings of the 4th conference on Designing interactive systems*, pages 141–146, New York, NY, USA, 2002. ACM. ISBN 1-58113-515-7. doi: http://doi.acm.org/10.1145/778712.778735. URL `http://portal.acm.org/ft_gateway.cfm?id=778735&type=pdf&coll=Portal&dl=GUIDE&CFID=94160061&CFTOKEN=65157542`.

Florian Block, Michael Haller, Hans Gellersen, Carl Gutwin, and Mark Billinghurst. Voodoosketch: extending interactive surfaces with adaptable interface palettes. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 55–58, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-004-3. doi: http://doi.acm.org/10.1145/1347390.1347404. URL `http://portal.acm.org/ft_gateway.cfm?id=1347404&type=pdf&coll=Portal&dl=ACM&CFID=91147169&CFTOKEN=55111441`.

B.H. Boar. *Application prototyping: a requirements definition strategy for the'80s*. Wiley, 1984.

A.K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. a cappella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 40. ACM, 2004.

P P Dourish and P Dourish. *Where the action is: the foundations of embodied interaction*. 2004. ISBN 0-262-54178-5.

Jerry Fails and Dan Olsen. A design tool for camera-based interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 449–456, New York, NY, USA, 2003. ACM. ISBN 1-58113-630-7. doi: http://doi.acm.org/10.1145/642611.642690. URL `http://doi.acm.org/10.1145/642611.642690`.

Hiroshi Ishii Gian Pangaro, Dan Maynes-Aminzade. The actuated workbench: computer-controlled actuation in tabletop tangible interfaces. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 699–699, New York, NY, USA, 2003. ACM. ISBN 1-58113-709-5. doi: http://doi.acm.org/10.1145/1201775.882330. URL `http://portal.acm.org/ft_gateway.cfm?id=882330&type=pdf&coll=Portal&dl=GUIDE&CFID=94194914&CFTOKEN=54446423`.

Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 209–218, New York, NY, USA, 2001. ACM. ISBN 1-58113-438-X. doi: http://doi.acm.org/10.1145/502348.502388. URL `http://portal.acm.org/ft_gateway.cfm?id=502388&type=pdf&coll=Portal&dl=GUIDE&CFID=94181837&CFTOKEN=41882439`.

Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–154, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: http://doi.acm.org/10.1145/1240624.1240646. URL `http://portal.acm.org/ft_gateway.cfm?id=1240646&type=pdf&coll=GUIDE&dl=GUIDE&CFID=96178478&CFTOKEN=81877251`.

Jonathan Hook, Stuart Taylor, Alex Butler, Nicolas Villar, and Shahram Izadi. A reconfigurable ferromagnetic input device. In *UIST '09: Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 51–54, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-745-5. doi: http://doi.acm.org/10.1145/1622176.1622186. URL `http://portal.acm.org/ft_gateway.cfm?id=1622186&type=pdf&coll=GUIDE&dl=GUIDE&CFID=95231286&CFTOKEN=18186340`.

Hiroshi Ishii and Brygg Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA, 1997. ACM. ISBN 0-89791-802-9. doi: http://doi.acm.org/10.1145/258549.258715. URL `http://portal.acm.org/ft_gateway.cfm?id=258715&type=pdf&coll=Portal&dl=GUIDE&CFID=94165030&CFTOKEN=63019040`.

H. Kimura, E. Tokunaga, Y. Okuda, and T. Nakajima. Cookieflavors: easy building blocks for wireless tangible input. In *CHI'06 extended abstracts on Human factors in computing systems*, page 970. ACM, 2006.

Hiroaki Kimura, Yohei Okuda, and Tatsuo Nakajima. Cookieflavors: Rapid composition framework for tangible media. In *Proceedings of the The 2007 International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 100–109, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2878-3. URL `http://portal.acm.org/citation.cfm?id=1306872.1306948`.

Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. Papier-mache: toolkit support for tangible input. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 399–406, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. doi: http://doi.acm.org/10.1145/985692.985743. URL `http://portal.acm.org/ft_gateway.cfm?id=985743&type=pdf&coll=GUIDE&dl=GUIDE&CFID=96286582&CFTOKEN=69872621`.

J.A. Landay and B.A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 50. ACM Press/Addison-Wesley Publishing Co., 1995.

Yang Li and James A. Landay. Informal prototyping of continuous graphical interactions by demonstration. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, UIST '05, pages 221–230, New York, NY, USA, 2005. ACM. ISBN 1-59593-271-2. doi: http://doi.acm.org/10.1145/1095034.1095071. URL `http://doi.acm.org/10.1145/1095034.1095071`.

John H. Maloney and Randall B. Smith. Directness and liveness in the morphic user interface construction environment. In *In Proceedings of User Interface and Software Technology (UIST 95) ACM*, pages 21–28. ACM Press, 1995.

J. Nielson. *Usability engineering*. Academic Press Cambridge, 1993.

B. Plimmer and M. Apperley. Interacting with sketched interface designs: an evaluation study. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1337–1340. ACM, 2004.

J. Raskin. *The humane interface: new directions for designing interactive systems*. Addison-Wesley, 2000. ISBN 0201379376.

Dan Rosenfeld, Michael Zawadzki, Jeremi Sudol, and Ken Perlin. Physical objects as bidirectional user interface elements. *IEEE Comput. Graph. Appl.*, 24(1):44–49, 2004. ISSN 0272-1716. doi: http://dx.doi.org/10.1109/MCG.2004.1255808. URL `http://portal.acm.org/ft_gateway.cfm?id=962719&type=external&coll=GUIDE&dl=GUIDE&CFID=94925978&CFTOKEN=62400019`.

M. Weiss, F. Schwarz, and J. Borchers. Actuated translucent controls for dynamic tangible applications on interactive tabletops. *Ext. Abstr. ITS*, 9, 2009.

M. Weiss, C. Remy, and J. Borchers. Rendering physical effects in tabletop controls. In *Proceedings of the 2011 annual* *conference on Human factors in computing systems*, pages 3009–3012. ACM, 2011.

Malte Weiss, Florian Schwarz, Simon Jakubowski, and Jan Borchers. Madgets: actuating widgets on interactive tabletops. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 293–302, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0271-5. doi: http://doi.acm.org/10.1145/1866029.1866075. URL `http://doi.acm.org/10.1145/1866029.1866075`.

Index

Typeset August 15, 2011