Maastricht University | universiteit hasselt
KNOWLEDGE IN ACTION

2016 | Faculty of Sciences

DOCTORAL DISSERTATION

# End-User Control over Physical User-Interfaces: From Digital Fabrication to Real-Time Adaptability

Doctoral dissertation submitted to obtain the degree of
Doctor of Science: Information Technology, to be defended by

**Raf Ramakers**

Promoter: Prof. Dr Kris Luyten
Co-promoter: Prof. Dr Johannes Schöning

# Acknowledgments

This dissertation has one author, yet it would not have been possible without the support and encouragement of many people.

First and foremost, I would like to address special thanks to my advisor Prof. Dr Kris Luyten, for giving me the opportunity to pursue a PhD. Kris sparked my interest in HCI research as an undergraduate research and gave me the freedom to persue my own research interests. He gave me the time to focus on fundamental research problems and offered great advice whenever I needed. Kris, thank you for supporting, supervising, and funding the various research projects and research stays that I engaged in. Thank you for all the opportunities that boosted my career. I would also like to thank my co-advisor, Prof. Dr Johannes Schöning for his endless encouragements, energy, and enthusiasm. As I oftentimes strongly believe in my own convictions, Johannes learned me to become a bit more pragmatic in my research approaches. Without his advice, this dissertation and other projects would not have existed at the time of writing. I would also like to thank Prof. Dr Karin Coninx for being part of my PhD. committee, her feedback on this work, and the opportunity to work in a great environment.

Furthermore, I want to express my thanks to the other members of my jury, Prof. Dr Jürgen Steimle, Prof. Dr Björn Hartmann, and Prof. Dr Wim Deferme. Thank you for your constructive and valuable feedback that helped to improve this dissertation. Thanks also to the chairman of my jury, Prof. Dr Marc Gyssens. I also would like to express my thanks to the panel of experts who provided feedback on my dissertation work during the UIST doctoral symposium: Prof. Dr Patrick Baudisch, Prof. Dr Daniel Ashbrook, Prof. Dr Eric Paulos, and Dr Andy Wilson.

The research in this dissertation is not only influenced by people, it is also inspired by other researchers. Over the past few years, I had the opportunity to work with some very inspiring researchers. Right after my master studies,

Prof. Dr Patrick Baudisch gave me the opportunity to start an internship at the Hasso-Plattner Institute. Working in such a creative environment was a remarkable experience and continued to influence my research style. Patrick, thank you for this experience and for your advice when we meet at conferences. Near the end of my PhD, Dr Tovi Grossman and Dr George Fitzmaurice gave me the opportunity to intern at Autodesk Research. Thank you for the amazing time in Toronto and for starting such a great project together. Also thanks for your trust in me until the last night when all the research results finally ended up in the paper. Thanks as well to Prof. Dr Daniel Wigdor for temporarily hosting me at the DGP Lab at the University of Toronto.

This work would not have been possible without the support of my colleagues and collaborators. In particular, I would like to say thanks to Kashyap Todi for being such a great collaborator on the PaperPulse project and for being a soundboard when experimenting with research ideas. I want to single out Dr Davy Vanacken for mentoring me during my master studies and for his advice on teaching. A big thanks as well to Dr Jo Vermeulen for all his feedback and input on my work. I also want to thank my co-authors from other institutes: Dr Fraser Anderson for the support throughout my internship at Autodesk and for sharing his knowledge on actuation techniques; Dr Dominik Schmidt and Dr Esben Pedersen for being such friendly collaborators on the Kickable project. Also thanks to Madeline Gannon for helping and intervening while I was fighting the 3D printers right before the CHI deadline.

I would like to express my gratitude to the EDM management, Prof. Dr Eddy Flerackers, Prof. Dr Frank Van Reeth, and Peter Vandoren, for the opportunity to work in such a supportive environment. I also thank Ingrid Konings and Roger Claes for helping me with administrative tasks and for taking care of logistics, especially when we shipped or bought material for conference demonstrations. One of the great things of working at the EDM are the friendly, talented, extremely helpful colleagues. I have not seen a place where it was that convenient to consult researchers across different disciplines. Thanks to Prof. Dr Peter Quax for his help and advice on electronics, networking, and drone related stuff. Danny Leen and Tom De Weyer, thank you for your help and assistance in the FabLab and MakerSpace. I also want to say thanks to Steven Maesen, Dr Patrick Goorts, Jeroen Put, and Nick Michiels for their advice and suggestions on computer graphics and computer vision related problems. Also thanks to Johannes Taelman for sharing his expertise on embedded system design, Karel Roberts for creating visual designs used in the PaperPulse project, and Prof. Dr Mieke Haesen and Stanislas De Vocht for their help and support while valorizing the some of the research results in

this dissertation.

I am also grateful to all the other colleagues whom I traveled, lunched, assisted courses, and discussed research with or who joined me to the gym: Dr Gustavo Rovelo, Donald Degraen, Kris Gabriels, Dr Maarten Wijnants, Dr Jan Van den Bergh, Pavel Samsonov, Supraja Sankaran, Marisela Gutierrez Lopez, Eva Geurts, Bram Bonné, Sven Coppers, Jens Brulmans, Thomas Kovac, Lode Jorissen and many others.

On a more personal note, I would like to thank my family and friends. In particular my mother for her support, and making my life easier than it should be. Thanks to my friends for the fun, stability, and support in difficult times. Special thanks to Jeroen Witters and Jim Stukken, who have been close friends for a very long time and who were always in for a good conversation. Last but not least, I want to thank my partner Evelien Ritzen, who has been on my side for over eight years now. Thank you for being so patient with me, for supporting me during the most stressful times, and for joining me in all my endeavours.

# Abstract

Graphical user interfaces are at the core of the majority of computing devices, including WIMP (windows, icons, menus, pointer) interaction styles and touch interactions. The popularity of graphical user interfaces stems from their ability to adapt to a multitude of tasks, such as document editing, messaging, browsing, etc. New tools and technologies also enabled users without a technical background to author these kind of digital interfaces, for example, filters for quick photo editing, interactive profiles on social media, or easy-to-use content management systems. In contrast, physical interfaces are hard to author as they embody electronic input and output sensors directly in the material. These physical interfaces are defined by both their physical form and functionality. As such, authoring physical interfaces oftentimes requires expertise of materials, electronics, design, and programming.

This dissertation investigates novel tools and techniques to bridge the knowledge gap that novices experience while authoring physical interfaces. On the one hand, we present software-hardware tools to support novices while conceiving new or repurposing existing physical interfaces using digital fabrication machinery. On the other hand, we tap into the future by exploring how transformable materials allow for adapting the form-factor of interfaces on the fly.

In this thesis, we make four main contributions to the body of authoring tools related Human-Computer Interaction (HCI) knowledge. Each of the contributions presents novel tools and systems which addresses a subset of the problems that non-experts experience while authoring physical interfaces. With prototype implementations, example walkthroughs, and user evaluations, we demonstrate how the different techniques alleviate the problems they address.

Through the concepts and systems presented in this dissertation, we lower the barriers for non-experts to make and deploy physical interfaces. For users

with a technical background, we make the engineering processes more convenient and efficient. In a larger sense, the novel concepts and systems presented in this dissertation, move interactive interfaces from traditional graphical user interfaces to physical spaces.

## List of Scientific Contributions

All research presented in this dissertation is published at primary forums of dissemination of research results in Human-Computer Interaction (HCI): the ACM Conference on Human Factors in Computing Systems (CHI) and the ACM Symposium on User Interface Software and Technology (UIST). The following overview lists the publications that contributed in a direct way to this dissertation:

- [Ramakers 16] Raf Ramakers, Fraser Anderson, Tovi Grossman, George Fitzmaurice. RetroFab: A Design Tool for Retrofitting Physical Interfaces using Actuators, Sensors and 3D Printing. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI) 2016, pp. 409-419. (Best Paper Honorable Mention Award)

- [Ramakers 15a] Raf Ramakers. Reconfiguring and Fabricating Special-Purpose Tangible Controls. In Adjunct Proceedings of the ACM Symposium on User Interface Software and Technology (UIST EA) 2015.

- [Ramakers 15c] Raf Ramakers, Kashyap Todi and Kris Luyten. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI) 2015, pp. 2457-2466.

- [Ramakers 15b] Raf Ramakers, Kashyap Todi and Kris Luyten, An End-User Development Approach for Designing and Fabricating Interactive Paper, Workshop on End User Development in the Internet of Things Era, ACM Conference on Human Factors in Computing Systems (CHI EA) 2015.

- [Ramakers 14] Raf Ramakers, Johannes Schöning and Kris Luyten. Paddle: Highly Deformable Mobile Devices with Physical Controls. In Pro-

ceedings of the ACM Conference on Human Factors in Computing Systems (CHI) 2014, pp. 2569- 2578.

- [Ramakers 13] Raf Ramakers, Kris Luyten and Johannes Schöning. Learning from 3D puzzles to inform future interactions with deformable mobile interfaces. Workshop on Displays Take New Shape: An Agenda for Interactive Surfaces, ACM Conference on Human Factors in Computing Systems (CHI EA) 2013.

In addition to the abovementioned works, I published other articles related to the work presented in this dissertation:

- [Schmidt 14] Dominik Schmidt, Raf Ramakers, Esben Pedersen, Johannes Jasper, Sven Köhler, Aileen Pohl, Hannes Rantzsch, Andreas Rau, Patrick Schmidt, Christoph Sterz, Yanina Yurchenk and Patrick Baudisch. Kickables: Tangibles for feet. In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI) 2014, pp. 3143-3152.

- [Ramakers 12] Raf Ramakers, Davy Vanacken, Kris Luyten, Karin Coninx and Johannes Schöning. Carpus: A Non-Intrusive User Identification Technique for Interactive Surfaces. In Proceedings of the ACM Symposium on User Interface Software and Technology (UIST) 2012, pp. 35-44.

I also demonstrated the work presented in this dissertation at two exhibitions:

- [Ramakers 15d] Raf Ramakers, Kashyap Todi, Kris Luyten. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. ACM SIGGRAPH 2015, Studio.

- [Ramakers 15e] Raf Ramakers, Kashyap Todi, Kris Luyten. PaperPulse: An Integrated Approach to Fabricating Interactive Paper. ACM Conference on Human Factors in Computing Systems (CHI EA) 2015, Interactivity.

# Contents

# List of Figures

# Chapter 1

## Introduction

With advancements in Do-It-Yourself (DIY) maker machinery (e.g. 3D printers, laser cutters, conductive inkjet printers) it becomes possible to produce physical artifacts in low volumes at reasonable costs. Hence, there is an increasing trend towards mass customization of physical artifacts, for example, designers making customized household gadgets. At the same time, there is an increasing interest in bringing concepts of interactive technologies, that we know from digital devices (e.g. intercommunication, dynamic content, real-time responses, etc.), to these physical artifacts by embedding sensor technologies.

Although DIY machinery makes it economically viable to produce highly customized interactive objects, designing these artifacts is still a tedious process involving experts in different disciplines, including visual/industrial designers, electrical engineers, and programmers. Researchers already explored techniques to facilitate making interactive physical artifacts for programmers [Greenberg 01, Hodges 13, Lee 04]. However, enabling users without a technical background to author physical interfaces did not receive much attention.

In this dissertation, I investigate and engineer systems and technologies that allow people without a technical background to make and adapt physical interfaces that match the constantly evolving needs of users.

## 1.1 Background

The human-machine interactions with one of the first computing devices in time, the ENIAC [Goldstine 46], was eminently physical, there were no graphical displays. Information entered the system using punch cards and the machine was controlled with patch cables, switches, and dials. Every functionality of the system was "physically embodied" and hence had its own dedicated physical handle, an interaction paradigm often referred to as "space-multiplexed input". The Sage [Everett 57] and later the Sketchpad system [Sutherland 64], introduced a graphical display at the center of the computer and some functionalities started to move to the screen and thus become "digitally embodied". However, every input functionality in these systems had its own physical handle. With the introduction of the Xerox Star [Johnson 89], a new interaction paradigm emerged that was entirely centered around the graphical display. Users interact with these systems using one of the most general-purpose of input devices: mice and keyboards. The only physical aspect of these input devices is that one can touch and hold them. Operations with these input devices are often referred to as "time-multiplexed", since the physical handle is being repeatedly attached and unattached to various logical functions of the graphical user-interface. With advancements in interactive display technology over the past decade, the direct touch interaction modality has replaced mice and keyboard for many mobile devices. This input modality requires users to time-multiplex, mostly the index finger, over various controls. Regardless whether the display is desk-mounted, head-mounted, hand-held, wrist-worn, or embedded in the physical environment, the prevailing combination of screens and general-purpose input devices is the predominant interaction style of computing devices nowadays.

One of the main reasons behind this trend towards more display centered interaction styles and general-purpose input modalities is that machines in this configuration, are known to be easy to modulate. In contrast to the ENIAC in which specialized controls are used for arithmetic, the screen, mouse, and keyboard of the Xerox Star served a multitude of office tasks. Modern computing devices today are repeatedly modulated and an increasing number of domains start to adopt some sort of digital embodiment on these systems. Examples include, finding additional information online for products in retail stores, following a recipe on a tablet while cooking, or tracking your progress while running.

In response to this trend, over the past two decades, researchers explored physical user interfaces (also often referred to as Tangible User Interfaces [Ishii 97,

Fitzmaurice 95]) which retain the physical handles of the earliest computing devices while preserving the computational power of computers nowadays. In these novel interfaces, researchers give physical forms to digital information and use these physical embodiments to manipulate the digital content. Figure 1.1 shows Urp, a seminal tangible interface by Underkoffler and Ishii [Underkoffler 99]. Urp makes it easier for urban planners and architects to experiment with shadows casted by buildings. Architects simply manipulate physical representations of these buildings on top of a display. Shadows casted by these buildings are rendered in real-time on the display corresponding to the position of these building and the orientation of the sun. Figure 1.2 shows another instance of a tangible interface, the Kickables system [Schmidt 14]. In contrast to traditional tangibles representations which are controlled using hands, Kickables are operated by feet. Although alternative versions of both systems could be designed that embody all controls digitally using mice, keyboards, or direct touch (using hands or feet), studies reveal several benefits that come with physical handles. These benefits include, bimanual interaction and eyes-free operation [Fitzmaurice 95], encouraging epistemic interactions [Kirsh 94], strong physical affordances [Ishii 97], facilitating collaboration [Underkoffler 99], memory recollection [Mugellini 07], etc.

Unlike traditional general-purpose computing devices, well designed physical embodiments cannot be modulated with every type of content as the physical design needs to reflect the digital information. Additionally, physical mediums are often hard to author as compared to digital mediums. Nowadays, digital mediums, such as graphical interfaces are easy to author by non-experts using well designed interfaces. As such, digital interfaces are easy to adapt to changing user needs. Examples include personalized profiles on social media, operating system UI-themes, search results, photos (using easy to use filters), websites (e.g. WordPress), questionnaires (e.g. Google Forms), configurations to prevent access to certain tv-channels, or specialized updates to make interfaces more convenient to operate by elderly or disabled individuals. In contrast, authoring interactive physical interfaces, such as Urp or Kickables, often requires a lot of expertise and skills in different domains, including materials, electronics, programming, and design. Even for physical mediums that are not digitally augmented, such as paper, or clay, authoring their physical shape is traditionally hard. Hence, people come from across the world to see origami structures and sand or stone sculptures created by artists and engineers with specialized skills and expertise.

In this dissertation, I investigate and engineer systems and technologies that allow non-experts to make and adapt physical interfaces to match the

**Figure 1.1:** Urp [Underkoffler 99]: a workbench for urban planners and architects to experiment with shadows casted by buildings.

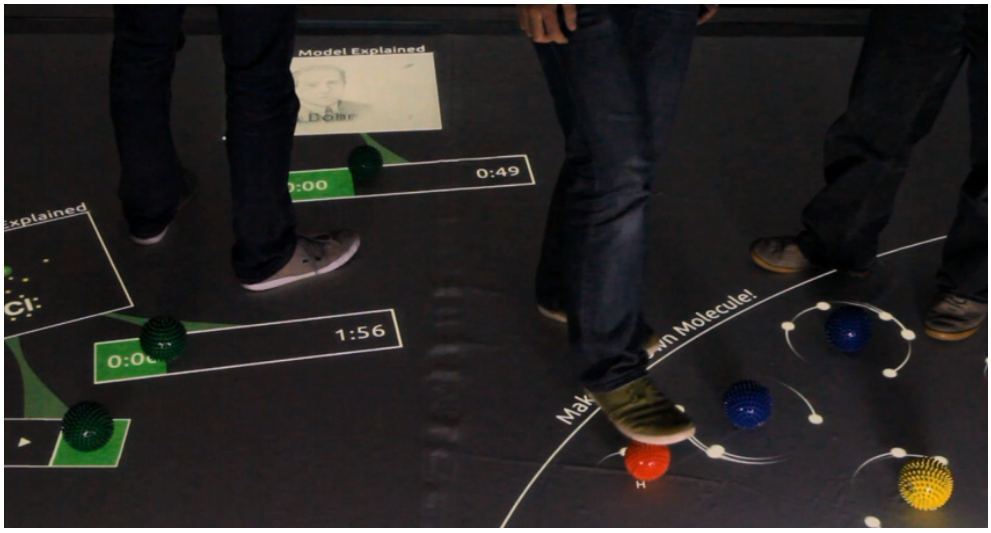


**Figure 1.2:** Kickables [Schmidt 14]: Tangibles for feet

constantly evolving needs of users. As such, these systems enable users without a technical background to take control over physical interfaces.

Many systems in this thesis focus on software design environments for authoring physical user interfaces. Traditionally, design environments support many features, to address the needs in established fields they target, such as Microsoft Visual Studio for programming, Eagle for electronic designs, and Adobe Illustrator, Autodesk Inventor, or similar CAD environments for 2D or 3D graphical designs. These design environments are often referred to as "expert tools". In contrast, end-user development environments target non-professional use and therefore provide a low thresshold for users to get started. Examples include, spreadsheets for analyzing and storing data models, Google Sketchup[1] for 3D modeling, and Scratch [Resnick 09] for programming. To facilitate the authoring process even further, "integrated end-user development environments" combine and streamline various functionalities available in individual development environments in order to ease the workflow for the user. Examples include, d.tools [Hartmann 06] which provides an integrated environment for designing, testing, and analyzing different versions of physical prototypes. Autodesk's 123D Circuit tool[2] integrates programming, electronics, testing, and deployment of embedded circuit designs.

## 1.2 Research Goals and Challenges

With advancements in Human-Computer Interaction (HCI), digital interfaces became easy to customize by non-experts. Examples include, profiles on social media, operating system themes, search results, photos (using easy to use filters), websites (e.g. WordPress), questionnaires (e.g. Google Forms), etc. Authoring physical interfaces, in contrast, is challenging as authoring these interfaces consist of many different aspects, including, shaping physical materials, graphical designs, electronics, and sensor-based programming. One needs expertise in these different domains to make interactive physical interfaces or adapt existing ones. In this dissertation, I investigate and engineer systems and technologies that allow users without a technical background to author interactive physical interfaces.

To enable non-experts to take control over physical interfaces, this dissertation focuses on the following three research goals:

[1] www.sketchup.com
[2] www.123dapp.com/circuits

**(G1) Establish integrated software-hardware solutions that enable users without a technical background to make new interactive physical interfaces by authoring the form-factor and behavior.**

**(G2) Establish integrated software-hardware solutions that enable users without a technical background to author the form-factor and behavior of existing interactive physical interfaces.**

**(G3) Establish integrated software-hardware solutions that enable users without a technical background to adapt the form-factor and resulting behavior of interactive physical devices in real-time.**

The following research challenges support these research goals and address the lack of expertise often experienced when working with technology for authoring physical interfaces:

**(C1) Availability of machinery and knowledge required to author physical materials**
Although many tools and techniques exist for making low-fidelity physical constructions, such as paper-prototyping, clay, or LEGO, precise manufacturing of high-fidelity materials, require professional tools and techniques. Without extensive training, non-experts are oftentimes unable to make precise paper models using origami folding techniques, weld materials, or operate a milling lathe or advanced CNC-machinery. Similarly, integrating electronic circuit traces into physical materials traditionally requires complex chemical etching methods. In this dissertation, we investigate inexpensive Do-It-Yourself (DIY) machinery for authoring physical materials (G1)+ (G2), as well as easy-to-use techniques for modulating physical materials in real-time (G3).

**(C2) Technical expertise and spatial reasoning skills required for making designs**
With digital fabrication machinery, a digital representation of the design is converted into a physical instantiation of that model. These types of machines generally require less technical expertise since they do not rely on the users' ability to shape materials and instead only require an accurate digital model as input from the user. The quality of the digital model thus depends on the users' experience with advanced modeling tools (e.g. vector graphics or 3D modeling environments) as well as their ability to convert an idea for a physical interface into digital models

often consisting of multiple parts (G1). Additionally, adapting existing physical interfaces traditionally requires users to model the existing interface (G2). Using these professional design tools however, requires extensive training.

**(C3) Electrical engineering knowledge required**
Seamlessly augmenting physical artifact with digital information often requires electronic components and microcontrollers to be embedded inside the physical material. Making electronic circuits however, requires knowledge of many electronic components and principles, including, analog-to-digital and digital-to-analog conversion techniques, signal processing, noise reduction, multiplexing strategies, etc. People without a technical background do not necessarily have this expertise.

**(C4) Advanced sensor-based programming knowledge required**
Creating new (G1) or adapting existing (G2)+ (G3) physical interfaces often requires changing the behavior, and thus the code, that controls electronic components. Specifying the behavior of sensor-based systems is very challenging for non-programmers. Furthermore, embedded systems that are often used to deploy these systems, have limited memory and processing power. Hence, a variety of programming skills are required.

**(C5) A wide diversity of heterogeneous tools and systems available**
Designing 2D or 3D models, writing program code, and designing electronic circuits all require using different modeling tools, electronic design tools, and programming environments, such as Autodesk Inventor, Adoble Illustrator, Eagle, or Microsoft Visual Studio. It is extremely challenging for non-experts to master these different tools without extensive training. Many physical interfaces have strong dependencies between, for example, the locations of components on electronic circuit boards and the 2D or 3D designed model. Users therefore frequently have to switch between different tools and deal with multiple aspects of the design process simultaneously.

## 1.3 Contributions

This thesis makes several major contributions to the field of human-computer interaction. Through the development of several systems, we contribute new

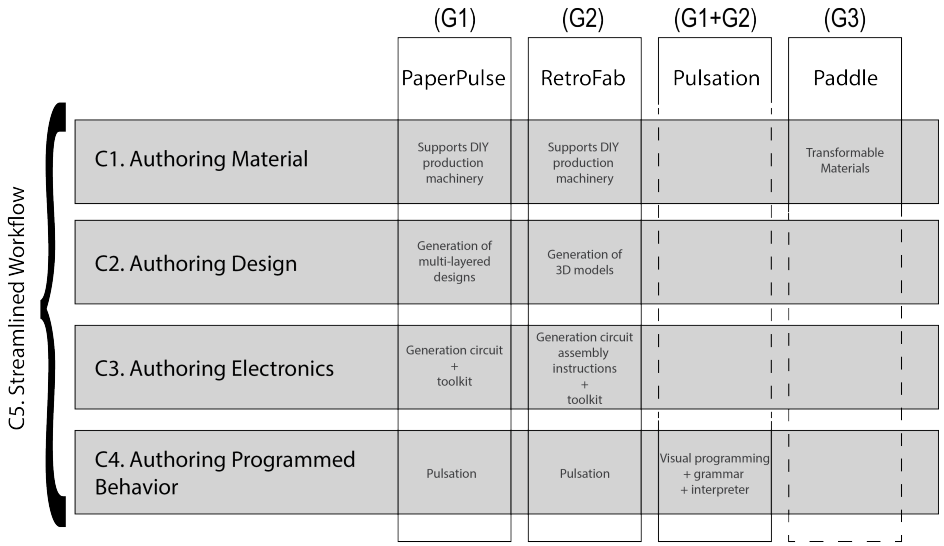


**Figure 1.3:** Linking the research challenges and goals to the contributions of this dissertation: Horizontal items define the research challenges (C1-C5). Vertical items define the contributions. The research goals (G1-G3) are at the top. The intersections summarize how the respective contributions resolve the research challenges.

knowledge and insights to enable people without a technical background to author various kinds of physical interfaces. Each of these systems target specific research goals and challenges outlined in Section 1.2. Figure 1.3 summarizes how the contributions link to the different research goals and challenges. Although all systems developed in this thesis are unique contributions in their own right, the concepts and techniques used to reach these goals and address the challenges, however, have broader implications that can directly be applied to other systems or inspire the design of novel systems in the future.

The following systems are developed in the context of this dissertation and support the research goals and challenges outlined in the previous section:

1. **PaperPulse [Ramakers 15c] enables users to design interactive paper interfaces (G1).** PaperPulse is an integrated design and fabrication environment in which users start by overlaying their visual design with electronic components (C2). After users specify the behavior of the electronic components using the Pulsation logic specification technique (C4), a multi-layered paper design is generated with embedded electronic circuit traces to power all components (C3). These layers are

then printed on paper using an off-the-shelf printer filled with conductive ink (C1). PaperPulse is an instance of an integrated end-user development environment (Section 1.1) and combines design, electronics, and programming in a single streamlined workflow, to guide non-experts in making new physical paper interfaces (C5).

2. **RetroFab [Ramakers 16] enables users to adapt existing physical interfaces (G2).** Instead of manually modeling existing physical interfaces in a software environment in order to adapt them, RetroFab allows users to adapt existing interfaces using 3D scans (C2). After users annotate the controls in the 3D scan, the system generates a 3D model that retrofits the original physical interface. Additionally the system suggests a redesign of the physical interface that the user can adjust (C2). To redirect interactions from the redesigned interface to the legacy interface, RetroFab supports a toolkit of actuators and sensors that the system automatically puts in place inside generated model (C3). The behavior of the electronic components, and thus the behavior of the legacy interface, is altered using the Pulsation logic specification technique (C4). Retrofit models can be produced with any conventional 3D printer (C1). After placing this 3D printed model over top of the original interface, the retrofit interface allows for new functionalities, as specified by the user. Similar to PaperPulse, RetroFab is an instance of an integrated end-user development environment (Section 1.1) and combines 3D modeling, electronics and programming in a single streamlined workflow to guide non-experts in adapting existing physical interfaces (C5).

3. **Pulsation enables non-programmers to specify the behavior of physical interfaces (G1+G2).** Although Pulsation can be used to program any kind of sensor-based system, it is seamlessly integrated in the PaperPulse and RetroFab vertical design tools. The visual Pulsation logic specification technique, targets non-programmers and is optimized for specifying functional relationships between electronic components. These relationships are especially relevant for the behavior of physical interfaces (C4). Pulsation makes it convenient for non-programmers to get started (low threshold), and at the same time allows for complex behaviors (high ceiling) by supporting advanced timing models. As such, it is possible to precisely specify output patterns of electronic components over time, or match complex input signals. In contrast to traditional visual programming techniques, Pulsation is specified in the context of the electronic components and does not require switching to separate

views or tools for adapting the behavior (C5). Pulsation can directly be deployed on embedded systems but also runs on desktop computer systems.

4. **Paddle [Ramakers 14] enables users to make changes to physical interfaces in real-time (G3).** Making changes to the physical material of interfaces in real-time, requires investigating materials that quickly transform between different states (C1). In the context of the Paddle system, we investigate transformation techniques used in the design of 3D puzzles. These techniques are used in the design of a new mobile device that users manually transform into various physical handles. Especially in mobile settings, fast transformation techniques of materials are relevant since physical construction kits or machinery are too clumsy and too slow for mobile ad-hoc changes.

## 1.4 Dissertation Outline

This dissertation consists of eight chapters including this introductory chapter. As a guide to the organization of the remainder of this dissertation, an overview of the chapters is provided below:

Chapter 2 *Related Work* Previous endeavors related to the research challenges outlined in this dissertation are surveyed. I provide an overview of fabrication techniques for making electronic circuits, sensors, and 3D structures. Following this discussion, we consider the wide variety of visual programming paradigms that are available nowadays. I also survey literature that combines and streamlines multiple aspects of design and fabrication processes into a single streamlined workflow. Finally, an overview is provided of existing techniques to author interfaces in real-time using transformable materials.

Chapter 3 *PaperPulse: Designing and Fabricating Physical Interfaces* I present an integrated design and fabrication environment that allows novices to make interactive physical interfaces on flexible substrates. A toolkit of electronic components is presented that easily integrates in flexible substrates. PaperPulse also integrates an early version of the Pulsation logic specification technique. With example designs and user studies, I demonstrate the versatility and utility of the PaperPulse design tool for novices.

CHAPTER 4 *RetroFab: Adapting Existing Physical Interfaces* I present an integrated design and fabrication environment that allows novices to adapt existing interfaces by making retrofit structures. A toolkit of electronic components is presented to facilitate retrofitting the most commonly used components in existing physical interfaces. Similar to PaperPulse, RetroFab integrates an early version of the Pulsation logic specification technique. With example designs, use cases, and a user study, I demonstrate the versatility and utility of the RetroFab design tool for novices.

CHAPTER 5 *Pulsation 2.0: Visual Programming for physical interfaces* The user studies with PaperPulse (Chapter 3) and RetroFab (Chapter 4) revealed several limitations of the Pulsation logic specification technique that is integrated in these design environments. This chapter, presents a novel logic specification technique, called Pulsation 2.0, that addresses those limitations. Pulsation 2.0 is optimized for specifying logic behavior of sensor-based systems on microcontrollers but also runs on desktop computers for simulation purposes.

CHAPTER 6 *Paddle: Real-Time Physical Transformations* I investigate how alternative techniques that allow for making changes to physical interfaces in real-time. In this chapter, materials that can be physically transformed are investigated. These materials are enriched with digital technology to allow non-experts to author them. These techniques are used in the design of a new mobile device that supports a variety of physical configurations. Especially during mobile use, physical controls need to transform in real-time

CHAPTER 7 *Discussion and Future Work* I consider aspects that bestride the individual chapters. This includes research problems at the intersection of digital fabrication and transformable interfaces. Furthermore a deeper discuss of the different user groups that can benefit from this work is provided. Finally, I elaborate on the user-experience challenges that come with computationally generated solutions and discuss how physical user interfaces can seamlessly integrate in environments in the future.

CHAPTER 8 *Conclusion* To conclude, I summarize the contributions of this dissertation.

# Chapter 2

## Related Work

In this chapter, previous endeavors related to the research challenges outlined in Section 1.2 are surveyed. First, an overview of fabrication techniques for making electronic circuits and sensors, lays the foundation for authoring physical materials (C1) and making electronic circuits (C3). Next, the state of the art in digital fabrication and 3D modeling reveals techniques to facilitate the design process to make artifacts (C2) and additional processes for authoring physical materials (C1). Following, an overview is provided of visual programming paradigms (C4). Building on top of these discussions, we review environments that streamline the process for making physical interfaces (C5). Finally, an overview is provided on techniques for authoring physical materials (C1) in real-time by covering research efforts related to transformable user interfaces.

## 2.1 Fabricating Electronic Circuits and Sensors

A wide variety of techniques exist to fabricate circuits and electronic sensors. While breadboarding is the most popular technique for experimenting with electronics, more advanced production techniques, such as chemical etching and conductive inkjet printing oftentimes result in better integrated and higher quality circuits. As such, there is an increasing interest to make these advanced circuit production techniques accessible for non-experts.

### 2.1.1 Electronic Rapid Prototyping Toolkits

Breadboarding is one of the most popular techniques for rapid-prototyping with electronic components for novice and expert users. Breadboards are solderless and therefore make it convenient to interconnect traditional through-hole components. For starters however, the wide variety of components, such as resistors, capacitors, shift registers, etc. needed to connect input/output components is often overwhelming. The Phidgets platform [Greenberg 01] and Calder toolkit [Lee 04] abstracts low level circuit details, such as H-bridge constructions for controlling DC-motors, and offer input/output components that directly connect to pins on the development board. To prevent users from attaching components incorrectly to pins, .Net Gadgeteer [Hodges 13] standardizes the connectors among all components and includes an FFC-connector (Flexible Flat Cable Connector) in every supported unit. Little Bits [Bdeir 12] further facilitates the circuit construction process by eliminating wires. Here, modular building blocks are interconnected using magnetic connectors that snap together to form a functional circuit. Similarly VoodooIO [Villar 07] eliminates the need for additional electronic components and wires by providing a flexible foam substrate in which prefabricated components, exposing pins, connect in any configuration. Every VoodooIO component integrates a controller to transfer its ID and state over the flexible substrate to a connected microcontroller.

Although these toolkits make prototyping with electronics accessible for a wide variety of users, the circuits produced are less permanent, less robust, and do not seamlessly integrate in every substrate, as compared to more circuit board fabrication techniques.

### 2.1.2 Circuit Board Fabrication Techniques

Overall two techniques exist for fabricating integrated circuits: additive and subtractive processes. Subtractive methods remove conductive material (e.g. copper) from a substrate, leaving only the desired conductive pattern behind. Additive methods on the other hand, add conductive materials on top of a non-conductive substrate. In the following sections, we discuss the most popular fabrication technologies used for both processes. A complete overview of all circuit board manufacturing techniques, since its invention in 1930, is out of scope – for this, we refer the reader to the work of Jawitz [Jawitz 97].

### Subtractive Circuit Fabrication Processes

Nowadays, a common method in industrial PCB fabrication is chemical etching [Jawitz 97]. PCB laminates, such as Bakelite or Kapton, are precoated with copper. A mask, representing the circuit, is applied on top of the substrate to protect the desired copper. Chemicals, such as ferric chloride are then used to etch (remove) the remaining copper. Various techniques exist for producing the chemical etching mask. We highlight three common methods. First, with silkscreen printing, an etch-resistant ink is applied on top the desired copper regions. In the second method, photoengraving, the copper-clad laminate is entirely coated with a UV-sensitive photoresist. The photoresist is then turned into an etching mask by selectively removing the photoresist at locations were the chemical etching is allowed. A mask can be transferred from a printed mask (using a regular printer) to the photoresist using UV light and a chemical solvent. Alternatively, when higher resolutions are desired, the mask is directly exposed to the photoresist using optical projection. The latter photoresist removal technique is often referred to as "Direct Imaging". The last technique to realize the etching mask is called "Toner Transfer". Laser printer toner is an ideal etch resist as it carries a high percentage of pulverized plastic. The toner is transferred from a special transfer paper on top of the copper by applying heat (e.g. using a iron). Emerging the board into water allows the paper to loosen, leaving the toner etching mask on the board.

Instead of chemical etching, a CNC milling machine[1] or laser engraver can also selectively etch the copper layer. Although these machines are very precise, they come at a premium price. To save costs, one can also use a vinyl cutter to etch or cut out copper traces [Savage 12].

Although most etching techniques originate from industrial production methods, many subtractive techniques are also available in DIY kits. In contrast, Printem [Varun 15] is a technology specifically targeted for non-expert use. In this technique, the inverse mask of the circuit is printed on top of the special printem film using a regular inkjet or laser printer. Inside the printem film, copper gilding foil is sandwiched between two substrates using a regular adhesive on the bottom and a UV adhesive at the top. After exposing the masked top-layer to UV light, the regions exposed to light stick to the top layer, while the masked regions stick to the bottom layer. When separating the layers, the top layer has the copper imprint of the circuit traces attached.

[1] www.othermachine.co

### Additive Circuit Fabrication Processes

Different techniques exist to add conductive traces on top of various substrates. Designing circuits using adhesive copper tape [Qi 14] is tedious but very versatile as this kind of tape sticks to a wide variety of surfaces, including paper, wood, walls, skin, etc. On the flipside, making turns using tape is hard and stretching the surface could break the tape. Conductive paint, such as graphite-based Bare conductive paint[2], makes it easier to produce hand drawn traces [Mellis 13]. However, the ink has a high resistance (55 $\Omega/\square$) and is very brittle. The silver-particle-based Circuit Scribe pen[3] has better conductive properties but can only be applied on paper-based substrates because of the ballpoint tip. As this pen produces conductive traces by dragging it over a substrate, this fabrication process can also be automated using a plotter[4].

With advancements in conductive inks, new types of additive fabrication processes were introduced. Some conductive inks can be directly applied to substrates using a screen printing mask, thus eliminating toxic chemicals needed for etching. Some inks however require an additional chemical or thermal sintering step for the ink to reach full conductivity. Recently, there is an increasing interest in silver nanoparticle ink, as the ink is viscous enough to be used in markers[5], industrial inkjet printers (e.g. Fujifilm Dimatix), or even off-the-shelf desktop inkjet printers [Kawahara 13]. A chemical sintering process between the ink and a substrate with a special resin coating ensures the conductivity of routed traces. Although researchers experimented with conductive circuit traces printed on origami folding structures [Olberding 15], the traces printed using silver nanoparticle ink are brittle and often break at folding creases.

Conductive threads provide a mean to integrate circuits in substrates that are bendable or stretchable, such as paper or textiles. These kinds of threads can be integrated during the paper [Coelho 09] or textile [Poupyrev 16] production process, or sewed into the fabric at a later stage [Post 00, Buechley 08]. Inline with work on wearable sensors, Weigel et al. presented iSkin [Weigel 15], an electronic circuit that can be worn directly on the skin as the substrate is flexible, stretchable, and translucent. iSkin consists of different layers of non-conductive polydimethylsiloxane (PDMS) and conductive carbon-filled PDMS (cPDMS). Detailed circuit patterns are produced from cPDMS using a laser engraver.

[2] www.bareconductive.com
[3] www.circuitscribe.com
[4] www.instructables.com/id/Paperduino-20-with-Circuit-Scribe
[5] www.agic.cc

### 2.1.3 Thin-Film Electronic Sensors

Once the electronic circuit traces are produced, electronic components, such as input and output components, a microcontroller, and a battery have to be connected. Although techniques exist to attach through-hole or Surface-Mounted Devices (SMD) directly to thin-film electronic circuits [Mellis 13], these components are often bulky or expose small connection surfaces which frequently cause electrical discontinuities. Hence, researchers presented various novel sensor and processing technologies that integrate reliably and seamlessly in flexible thin-film electronic circuits.

To allow non-experts to connect traditional electronic components reliably to thin-film electronic circuits without requiring soldering, Hodges et al. presented CircuitStickers [Hodges 14]. CircuitStickers are flexible PCBs that integrate traditional electronic components and expose enlarged connection pads at the bottom of the sticker. As the bottom of the sticker is covered with Electrically Conductive Adhesive Transfer (ECATT) tape, the stickers directly attach to conductive circuit traces. Similarly the Lilypad platform [Buechley 08] offers stitchable battery holders, sensors, and microcontrollers that easily integrate in fabrics using conductive threads.

Continuing the seamless integration of sensors in substrates, Olberding et al. [Olberding 13] present a thin-film capacitive multi-touch sensor that can be cut to the desired shape using scissors. Electrodes are positioned in a star or three shape topology to support various convex and concave shapes. A simple calibration step is required to normalize the sensor readings from partially cut electrodes. Foldio [Olberding 15] shows how to bring capacitive sensing beyond flat surfaces by augmenting corners, edges, and faces of paper origami structures with sensor pads. Additionally, techniques are presented to track specific deformations of origami structures, such as the angle of folds, or the amount of shearing, linear elongation, or rotation. Using integrated (printable) receiving and transmitting electrodes for capacitive sensing, these deformations can represent input parameters in the interactive system. Qi and Buechley [Qi 10] took this idea one step further and explored techniques to electronically track the state of paper popup structures [Carter 99], such as tabs and rocker arms. As RFID antennas are convenient to print on substrates, researchers investigated techniques that leverage passive RFID tags to recognize user interactions. PaperID [Li 16] monitors low-level channel parameters of the RFID communication using machine learning techniques to identify user interactions, such as touch, sliding, turning, swiping, and movements of tags or hands. RapID [Spielberg 16] uses a similar technique to realize an easy-to-use

framework for detecting covered RFID tags.

Besides binary or linear input sensors, a wide range of thin-film flexible 2D touch pad sensors have been presented. PyzoFlex [Rendl 12] is a pressure sensitive foil consisting of a piezoelectric material that is sandwiched in between a layer of driving and sensing electrodes. The piezoelectric material develops an electrical charge proportional to the change in mechanical stress. However, this raw signal data is not usable for absolute measurements as the electric charges decay with a time constant. This discharge follows a predictable exponential function and every deviation of the predicted value must be caused by a new pressure change at that location on the sensor. Using a similar layered construction with a different sensor layout resulted in the FlexSense [Rendl 14] system. In addition to detecting pressure changes, FlexSense accurately reconstructs the deformation of an A4 sensor surface. The surface integrates 16 piezoelectric sensors of which readings over time are integrated in a machine learning algorithm. In order to detect touch, pressure, proximity, and sheet folding in a single substrate Gong et al. presented PrintSense [Gong 14], which combines different kinds of capacitive sensing techniques. PrintSense consists of 42 $1.5cm^2$ electrodes of which one interleaved set will be used as receiving electrode while the others serve as sending electrodes. When a finger touches an electrode, its rate of discharge changes. In proximity mode, every electrode is connected to an analog sensing circuit, which detects noise in any of the readings caused by hovering body-parts. In this mode, two additional 3x6cm electrodes provide greater transmission range. Additionally, by measuring the distance between specific pairs of electrodes, folding over an edge of the sensor detectable. By shaping the electrodes as "inter-digitated" (IDT) fingers, different levels of pressure are detected using galvanic skin response (GSR) measures. A similar architecture of electodes is used by Gong et al. [Gong 11] to realize thin-film pressure sensitive floor pannels.

To realize touch sensor pads that are worn directly on skin, the sensor has to be stretchable as well as flexible. Kramer et al. [Kramer 11] presented a thin-film keypad consisting of twelve keys. The keypad is composed of polydimethylsiloxane (PDMS) embedded with microfluidic channels of Eutectic Gallium-Indium (eGaIN) alloy. The sensor however is fabricated with a complex photolithography process and a substantial amount of pressure is needed to operate the pads (approximatly 10 times as much force as operating a regular keyboard). In contrast, the sensors pads used in iSkin [Weigel 15] are convenient to customize by cutting the carbon-filled PDMS film, that is used for both the receiving as well as transmitting electrodes, to a particular shape using a laser engraver. The two electrodes are then separated using a perfo-

rated layer of non-conductive PDMS. The custom-shaped electrodes are used for both resistive as well as capacitive sensing, in order to distinguish between light and firm touches.

Researchers also explored integrating sensors in fabrics. Parzer et al. [Parzer 16] realized a pressure-sensitive textile by separating two fabrics consisting of sensing lines using a force sensitive material (Eeonyx EeonTex LG-SLPA). Alternatively, driving and sensing lines needed for a 2D touch pad can be integrated in the fabric during the weaving process [Poupyrev 16]. Sugiura et al. [Sugiura 12] presented a sensor to measure tangential forces in stockings. The expansion and contraction of the stocking is estimated by considering the transmissivity of infrared light passing through the stocking using a photoreflector.

Besides input sensors, researchers also explore technologies for making thin-film output sensors. Foldio [Olberding 15] supports actuated bending of paper as an output modality. This sensor consist of a printed resistor realized using screen printing or conductive inkjet printing, that is overlaid with polyethelyne tape. Once the resistor heats up, the polyethelyne patch expands while the base paper layer retains it length. This results in an actuation of the compound material. Perumal and Wigdor [Varun 15] as well as Coelho et al. [Coelho 09] demonstrate how to integrate a speaker in a substrate by realizing a voice coil and placing an electromagnet behind it. Alternatively, piezo speakers can be integrated into paper to realize a paper headphone [Shorter 14]. More high-fidelity visual output is offered by thin-film ElectroLuminescence (EL) displays [Olberding 14]. An electroluminescent panel is in essence a capacitor where the insulator (dielectric) between the outside plates is a phosphor that gives off photons when the capacitor is charged. By making one of the contact panels transparent and printing conductive material on the other substrate, in a specific shape, this particular shape emits light. Devendorf et al. [Devendorf 16] show how displays seamlessly integrate in fabrics by weaving conductive treads in textiles. The treads are coated with thermochromic pigments and thus change color in response to heat caused by an electric charge.

For the remaining components required to make electronic circuits, such as a power supply [Bates 00] and a microcontroller [Myny 12], thin-film solutions are becoming available. For example, Enfucell[6] is a commercially available soft thin-film battery with a 90mAh capacity. Karagozler et al. [Karagozler 13] showed how a high voltage spike (lasting up to 50ms) is generated by rubbing Teflon material over a conductive sheet. Researchers produced 8-bit organic microprocessors on plastic foils. This microcontroller is programmed by print-

[6] http://www.enfucell.com

ing assembler instruction sets in a Write Once Read Many (WORM) memory using conductive inkjet technologies [Myny 12].

## 2.2 Digital Fabrication and 3D Modeling

The current popularity of 3D printing and digital fabrication has led to a number of recent works in the HCI literature related to digital fabrication processes and 3D modeling.

### 2.2.1 Simplifying and Accelerating the Fabrication Process

This dissertation builds upon work that automates, simplifies, or accelerates the fabrication process of 3D objects. For example, Fabrickation [Mueller 14b] allows users to rapidly prototype 3D objects by substituting parts of the model with LEGO blocks, thus reducing print time. Autodesk's 123D Make software solution[7] converts 3D models into flat laser cut designs that can be assembled into a 3D model using either finger joints or by stacking layers. WirePrint [Mueller 14a, Peng 16] allows one to fabricate a wireframe preview version of a 3D model and evaluate its look and feel before fabricating the final high-fidelity version. The time between design iterations is also reduced by printing on top or removing parts of the object produced in the previous design iteration. This approach is taken by Teibrich et al. [Teibrich 15]. They integrate a milling head to remove obsolete parts and use a rotating platform to access different regions of the object without having to remove all previously printed layers. Alternatively, 3D printers can print directly on top of existing objects [Chen 15].

Lau et al. provide a system that decomposes 3D models of furniture into its constituent components and connectors, allowing users to easily fabricate the discrete elements [Lau 11]. LaserOrigami [Mueller 13] eliminates the manual assembly process of multiple parts by automatically folding acrylic sheets using a laser cutter. LaserOrigami achieves this by heating up selected regions of the workpiece until they become compliant and bend down under the force of gravity. As laser cutters are traditionally faster than 3D printers, CardBoardiZer [Zhang 16] converts a 3D model into planar or volumetric foldable patterns that are fabricated using a laser engraver or vinyl cutter. However, laser cutters do not offer multi-material fabrication. Hence, Perumal and Wigdor [Wigdor 16] devised a multi-material layered sheet to use in a laser cutter.

[7] www.123dapp.com/make

By stacking a rigid, bendable, and flexible layer, the laser engraver selectively cuts through one or multiple layers to realize joints with different properties. A similar approach is taken by HotFlex [Groeger 16] which is a substrate allowing for post-print customizations of 3D printed objects. By inserting heating elements during or after the 3D printing process, the material computationally switches between a solid and deformable state. As such, variations on the original design are convenient to realize without starting over the 3D printing process.

### 2.2.2 Facilitating the 3D modeling Process

To facilitate the digital modeling process, MixFab [Weichel 14] provides users with a mixed-reality environment where real-world objects are incorporated into the design of virtual 3D objects which can then be fabricated, thus making it easier for users to design with existing, tangible objects. SketchChair [Saul 11] specifically eases the 3D modeling process for making custom designed chairs. The process starts from a 2D side view of the chair that the user sketches, after which the system generates a 3D model by extruding the sketch. When the user changes parameters of the design, the chair's stability is tested using a physics simulation. The final model is converted into 2D parts consisting of finger joints that are produced using a laser cutter. Similarly Teddy [Igarashi 07] allows non-experts to model stuffed animals starting from a 2D sketch. Sketches are converted into 3D models by inflating enclosed regions. Plushie [Mori 07] builds further on this work and converts the final 3D model back into a 2D fabric layout that can be stitched together to realize the final stuffed animal. ModelCraft [Song 06] facilitates making changes to existing 3D models by allowing users to make pen annotations onto a physical representation of the model consisting of Anoto paper. These pen annotations represent extrusions, cuts, and fillet operations. All operations are digitized and applied to the original 3D model.

Reform [Weichel 15b] introduces the concept of "bidirectional fabrication" using a 3D printer that extrudes clay. When the 3D print is finished, the user manipulates the clay object further to adjust the design. In the next design iteration, the printer first scans the adjusted clay model to digitize all changes. There are also a number of electronically augmented measurement tools to facilitate modeling existing physical objects. SPATA [Weichel 15a] presents a digitally adapted version of a caliper and protractor. Using these tools distances and angles are measured in physical space and automatically entered in the digital model that the user is working on. To automatically

model more complex object curvatures AR-Jig [Anabuki 07] consists of an array of pins that are pushed against the object in order to obtain the inverse shape.

The Constructable system [Mueller 12] reduces the need for making upfront design decisions when using laser cutters. Instead of designing a 2D sketch in a design environment, users interact with the laser cutter by drafting directly on the workpiece using hand-held laser pens. An overhead camera tracks these interactions, and instructs the laser cutter after beatifying the hand drawn paths. In a similar vein, Tactum [Gannon 15] facilitates the design of wearables by manipulating a projected 3D mesh directly on the body using touch interactions. When the design is completed, the final mesh is produced using a 3D printer. This traditional two-step fabrication approach (design then fabricate) is eliminated in ExoSkin [Gannon 16] where the user moves around a skin-safe clay extruder directly on the body. Projected dynamic toolpaths assist the user in realizing designs. Similar handheld extruders are used in the 3Doodler[8] and Protopiper [Agrawal 15] system to provide a means to manually sketch respectively small or large (room-size) physical objects.

### 2.2.3 Adding Interactivity to Fabricated Objects

A number of projects have developed techniques to aid the construction of electronically augmented physical objects. Makers' Marks [Savage 15] is a system that allows users to create complex physical objects that incorporate elements such as hinges, parting lines and electronics using sculpting material and stickers for annotation. As it is often hard to represent the exact location and orientation on curved 3D models using stickers, Jones et al. [Jones 16] present an alternative approach. Instead of using stickers, a toolkit of widget blanks are designed with unique fiducial markers on top. The user partially embeds these widget blanks in the clay. After the clay model is scanned, the system recognizes and replaces the widget blanks with mounting structures of electronic components after which the final model is 3D printed. The Enclosed [Weichel 13] design tool also makes it easier to fabricate interactive objects. This software environment enables users to easily generate laser-cuttable enclosure structures that hold together electronic components. PipeDream [Savage 14] automatically routes pipes through 3D models to allow for post hoc insertion of conductive materials. In a similar vein, SurfCuit [Umetani 16] routes circuit channels on top of 3D models based on standard circuit schematics. After the 3D printing process is finished, the

[8] www.the3doodler.com

user manually inserts copper tape in the circuit channels. Sauron [Savage 13] automatically modifies the internal geometry of 3D models to make it suitable for vision-based sensing. In this system, all interactive widgets are passive and tracked from a camera integrated in the 3D model. Sauron assist users in placing the camera and mirrors in order to track the state of all interactive widgets. Alternatively, the state of interactive widgets can also be tracked using a wireless accelerometer integrated in every knob [Hook 14].

A variety of projects also explore adding interactivity at the time of fabricating the original object. Ishiguro et al. [Ishiguro 14] presented a technique to 3D print speakers directly into a 3D model. The technology is based on principles of electrostatic sound reproduction. As the speaker only consists of an electrode and a diaphragm, the sensor can take on any shape and can be produced using conventional 3D printers. Willis et al. [Willis 12] 3D prints optical light sensing pipes. By printing the pipes in transparent material and the cladding in support material, the difference in material density allows for Total Internal Reflection (TIR) to occur. As such, light signals can be transmitted through the tubes. Using these tubes, buttons, movement sensors, and touch sensors are realized. To seamlessly integrate conductive tracks directly in objects at fabrication time, Fused Deposition Modeling (FDM) printers start to support conductive filaments. PrintPut [Burstyn 15a] leverages this technology to realize 3D printed touchpads, pressure sensors, sliders, and flex sensors. The Voxel8 3D printer[9] is specifically targets fabrication of electronically augmented 3D objects by supporting conductive silver ink.

## 2.3 Visual Programming Methodologies

To allow users without a programming background to specify logic behavior, researchers investigated techniques including visual programming [Kelleher 05], programming-by-demonstration [Hartmann 06, Hartmann 07], and tangible programming [Lysecky 09, Bdeir 12]. The work in this dissertation relates to visual programming methodologies. In visual programming paradigms, visual building blocks are used to compose programs instead of writing code statements. These visual programming approaches simplify programming by either preventing syntax errors or by tailoring or simplifying the language for a specific domain of programming problems [Kelleher 05]. Visual programming approaches make it convenient for new user populations to program systems or facilitate teaching of programming languages.

[9] www.voxel8.co

### 2.3.1 General-Purpose Visual Programming

LOGO [Abelson 74] is considered as the first child-friendly programming language (1967). LOGO is a general-purpose functional programming language, although the language is well known and mainly used for its turtle graphics features. Turtle graphics consist of a turtle robot that is programmed to move on a Cartesian plane. In many implementations, the turtle robot has a pen attached and leaves behind a trace everywhere it moves. As such, arbitrary complex shapes are created by programming the turtle robot to move. Over the years, many alternatives and extensions for LOGO have been created, including KTurtle[10] and StarLogo [Colella 01]. StarLogo includes features to program multiple turtle robots in parallel and is therefore ideal in agent-based simulations. Although the original version of LOGO is textual, the StarLogo Nova[11] implementation offers visual building blocks and drag-and-drop interaction styles to compose code statements.

Drag-and-drop programming languages consist of visual blocks, representing code constructs, such as variables, keywords, statements that are nested to realize arbitrary complex statements. These visual building blocks oftentimes have specific shapes to make clear which types of blocks are valid in a specific context (Poka-yoke constraining) and thus avoid syntax errors. Scratch [Resnick 09] is the most popular drag-and-drop programming language and is often used in schools to introduce children between 8 and 16 years to programming. Scratch is most suited for making animations and small games. Oftentimes it is also used in the classroom to enrich math and science projects with computing. A variety of alternatives for Scratch exist, including Snap [Harvey 14] which extends Scratch with options, including function as first class citizens. Other systems build on top of these languages and offer additional functionalities for programming Arduino development boards, such as Scratch4Arduino[21], Snap4Arduino[12] or household appliances e.g. Scratchable devices [Ash 11]. The Alice 3D programing platform [Conway 00] shares inspiration with Scratch but targets first year computer science students. The platform offers an object oriented language oriented towards making virtual reality scenes. Using this platform, students experiment with concepts in computer graphics, such as vectors, cameras, transformations, etc. The AppInventor platform [Wolber 11] offers a similar drag-and-drop programming environment but includes functionalities to facilitate making Android applications.

[10] https://edu.kde.org/kturtle
[11] http://www.slnova.org
[12] http://snap4arduino.org

Although these general-purpose visual programming paradigms offer a wide-variety of functionalities (high ceiling) similar to regular textual programming languages, users first need to learn different programming constructs, such as variables, functions, loops, etc.

### 2.3.2 Special-Purpose Visual Programming

Instead of learning non-programmers all constructs available in traditional programming languages, special-purpose visual programming approaches use alternative visualization techniques and focus more on specific types of applications. Finite-State machines are visualized using state diagrams and are well suited for applications that can be modeled in different states. The D.tools [Hartmann 06] design environment offers designers state diagrams to model basic interactions with prototyped devices. Finite-state machines however have several limitations. For example, there is no memory available for saving data, all memory must be encoded in states.

A variety of programming techniques consist of visual building blocks that are linked together. Here we can identify two types: *control flow* and *dataflow* programming. Control flow programming is the traditional type of programming, a sequence of instructions is executed sequentially. Instructions use or operate on data that is linked to the instruction. In dataflow programming in contrast, some data moves through a sequence of instructions. Instructions manipulate the data once all inputs become valid. The output is passed onto the next instruction. PureData [Puckette 96] is one example of a visual dataflow programming language. It is most often used to process and transform audio signals. In PureData, various operations can be linked after which the audio signal is passed at a certain rate through the different operations. LabView [Wells 96] is another instance of a visual dataflow programming language which is most often used to engineer control systems. PureData and LabView both target expert usage.

Although originally inspired by LabView, the visual programming languages supported by LEGO Mindstorms, including RCX code [Barnes 02], RoboLab [Erwin 00], and Mindstorms NXT [Kim 07] are control flow languages. These visual languages offer easy to compose operations for controlling robots. Some visual programming languages even consists of both dataflow and control flow visual programming techniques. eBlocks [Lysecky 09], for example, supports both conditional statements to jump to the correct branch of code, as well as interlinked sequences of operations that manipulate and transform signals. Control flow visual programming paradigms are also widely used

to facilitate rendering graphics data. Examples include quartz composer[13] for composing graphics in user-interfaces, NodeBox[14] for rendering parametric 2D graphics, and Grasshopper Rhino[15] for rendering parametric 3D models.

With the increasing popularity of easy-to-deploy sensor units and personalized online services (e.g. social media, online retail stores, cloud storage, etc.), programming techniques have been developed that allow users without any programming background to interconnect sensors and online services. These systems oftentimes aim to simplify the visual language and allow novices to specify logic without following any tutorials. Therefore, these languages are limited to simple trigger-action programs and thus provide a low ceiling. The most popular platform in this category, IFTTT "If This Then That"[16], offers a simple interface to interconnect pairs of online services or network connected sensors. Atooma[17] is a similar platform but supports multiple triggers and actions in a single conditional rule. Ur et. al [Ur 14] showed that these kind of extensions are still easy to grasp by inexperienced users.

In contrast to the trigger-action programming paradigms discussed in the previous paragraph, Spacebrew[18] and IBM NodeRed[19] visual programming platforms target engineers and electronic hobbyists. These systems facilitate interconnecting custom-build sensing units (e.g. using Arduino). With Spacebrew, one visually interconnects systems that expose websockets using a publisher and subscribe mechanism. IBM NodeRed, provides more advanced options to process the input signal before redirecting it further to the target websocket.

## 2.4 Design Environments for Sensor-Based Interactions

Besides the individual systems that facilitate fabricating electronic circuits, produce 3D objects, or program logic, as discussed in the previous sections, a variety of systems integrate multiple of these aspects into a single streamlined workflow. These integrated design environments assist users in transitioning

[13] https://developer.apple.com/library/content/documentation/GraphicsImaging/Conceptual/QuartzComposerUserGuide
[14] https://www.nodebox.net
[15] http://www.grasshopper3d.com
[16] https://ifttt.com
[17] https://www.atooma.com
[18] http://docs.spacebrew.cc
[19] http://nodered.org

between design iterations or/and provide an end-to-end solution to assist from ideation until realization.

To assist programmers in specifying logic for electronic components, researchers developed toolkits consisting of easy to assemble electronic components and microcontrollers with well-defined programming interfaces, such as .Net Gadgeteer [Hodges 13], Phidgets [Greenberg 01], iStuff [Ballagas 03], The Calder Toolkit [Lee 04]. Autodesk's 123D Circuits platforms[20] supports immediately testing of code and the electronic circuit using a software simulator. Midas [Savage 12] provides an integrated environment for augmenting existing objects with electronics. After loading a photo of the design and overlaying it with capacitive sensor pads, the system generates circuit traces to the edges of the design. These traces are then cut out of copper foil using a vinyl cutter. Touches on the sensor pads can be linked to prerecorded actions on a computer or forwarded to other applications using websockets. Foldio [Olberding 15] goes beyond flat designs and allows users to add sensor pads to 3D models. The 3D model is then converted into a paper folding model which also routes all sensor pads using circuit traces to a single connection region. Foldio designs are produced using conductive inkjet printers or silkscreen printing.

Several design environments also empower non-programmers to define sensor-based logic. Some use visual programming techniques with building blocks, such as ScratchForArduino[21], Modkit [Millner 11], and eBlocks [Lysecky 09]; other systems support specifying basic events by linking handwritten keywords to electronic components [Block 08]. Boxes [Hudson 06] provides a toolkit consisting of thumbtacks, tin foil, and a microcontroller to augment very early stage cardboard prototypes with interactivity. The microcontroller uses capacitive sensing to recognize user's touches on thumbstacks and tin foil and communicates with a computer that links these events to recorded mouse actions. To realize higher-fidelity prototypes, d.tools [Hartmann 06] provides an integrated environment for designing, testing, and analyzing different versions of prototypes. In design mode, non-experts specify sensor logic by relating input events to output actions using a state chart visualization. Instead of specifying input events using a keyboard and mouse, input signals are demonstrated in physical world, with the respective sensors, using a programming by demonstration paradigm (Section 2.3). Alternatively, d.tools allows for writing more complex program code with the help of a programmer. In test mode, user interactions with the prototype are logged and synchronized with a video recording of the test setup. In analyze mode, the designer gets an overview of

[20] www.123dapp.com/circuits
[21] www.s4a.cat

the interaction model and the video recordings. aCAPpella [Dey 04] provides a design environment for linking more high-fidelity input sensors, such as webcams, audio, RFID readers to output actions. The system first records the events from all the sensors that are attached. Then the user selects the relevant events from the recordings and links these events to output actions. When these highlighted events are recognized in the future, the system executes the associated output action.

Some design environments are also entirely physical, such as the Little Bits toolkit [Bdeir 12] which provides physical building blocks for both sensors as well as logic elements, such as filters, delays, and inverters. PICL [Fourney 12] is a general-purpose hardware circuit learning unit that accepts analog sensor readings as input and converts it to an output value via a learning function. This learning function is specified by the user using the controls on the PICL unit. PICL can be trained as a linear classifier (output 0 or 5v) or a linear regression mapping (output in between 0 and 5v).

## 2.5 Real-Time Transformable User Interfaces

Fabricating physical models using DIY or industrial machinery (Section 2.2) is oftentimes a slow process lasting a few hours up until a few days to realize the final artifact. As such, real-time changes of physical models are not yet possible with these technologies. Instead of using machinery to change a physical artifact, in this section, we explore artifacts that consist of "smart materials" instead. Smart materials transform themselves or can be transformed to different physical configurations quickly. Although transformable materials are traditionally used in paper-like interfaces [Gallant 08, Holman 05] and deformable mobile devices [Lahey 11], advancements in the transformable properties of these materials and engineering principles advanced make it possible to start using these concepts for switching between different form-factors in real-time [Follmer 13].

### 2.5.1 Manual Transformable Interfaces

Holman et al. [Holman 05] presented a system called PaperWindows which allows paper to be used as a medium for representing digital information. PaperWindows tracks sheets of paper and users' fingers with an optical tracking system. By projecting digital content on top of sheets of paper, every sheet provides a window to digital information. Data is transferred between windows by gesturing with hands and fingers. Building on top of this work, researchers

explored various interaction concepts that leverage bending, folding, or rolling of paper-like devices. Gallant et al. [Gallant 08] explored interactions with squeezing, folding, or bending flexible substrates. Although an optical tracking system was used to track the state of the foldable device, output was provided on a separate monitor. FoldMe [Khalilbeigi 12] is a collection of transformable devices with two or three fixed hinges. The position and angle of the slates are tracked using optical tracking system in order to project visual content on top. Various interaction techniques are explored, including flipping and folding, to navigate through content. Additionally, some scenarios use the angle of the slate to provide continuous control over parameters. Using a similar tracking and projection system, Xpaaand [Khalilbeigi 11] bridges the gap between displays of different sizes. Xpaaand works similar to an ancient paper scroll which is rolled in and out. Resizing the device is used as a means for (semantic) zooming, navigating through content, and switching between applications. Besides rolling paper in and out Lee et al. [Lee 08] explores the concept of adjustable screen sizes by augmenting folding fans and umbrellas. To enable markerless tracking of paper sheets, Steimle et al. [Steimle 13] presented Flexpad, a system that reconstructs the shape of a sheet of paper from depth information of a single Microsoft Kinect camera. By distorting the projected information accordingly, the paper is augmented with digital information. Flexpad presents a diverse set of spatial interaction techniques including, curved cross-cuts in volumetric imaging, animating virtual paper characters, and slicing through time in videos.

Besides interaction techniques, various projects also investigated engineering techniques to make transformable devices self-contained. Starting with Hinckley et al. [Hinckley 09] dual screen tablet computer which integrated two PDA's connected with a hinge. The postures of the device support different nuances of individual or collaborative work by measuring the orientation and angle between both displays. The Gummi [Schwesig 04] prototype was the first self-contained bendable device consisting of a rigid screen mounted on top of a flexible layer which integrates bend sensors. Besides navigating through content, such as maps and videos, bending interactions were also used for text-entry. PaperTab [Tarun 13] brought the original idea, mocked up in PaperWindows [Holman 05], to life. PaperTab uses touch sensitive electronic ink (E-ink) displays, each integrating an electro-magnetic tracker for location tracking. Using similar display technology, PaperPhone [Lahey 11] presents a bendable mobile device in which bend gestures are used to, for example, navigate through contacts, or control a music player. ReFlex [Strohmeier 16] is a similar prototype of a bendable phone but also integrates vibrotactile sensation

and audio feedback for providing feedback during the bending interaction. For example, when navigating an e-book using Reflex, users feel and hear pages flip. These kinds of e-reading experiences with bendable devices are also explored in Bookisheet [Watanabe 08]. However this earlier prototype did not include a display and instead provided visual output on a separate screen. In a controlled experiment Wightman et al. [Wightman 11] showed that for single page navigation, bending interactions are as efficient and result in fewer errors as compared to button techniques on traditional e-readers and mobile devices. HoloFlex [Gotsch 16] explorers the concept of holographic imaging on flexible mobile devices. Bend gestures are used to navigate the Z-axis, a concept also explored extensively by Leflar et al. [Leflar 14]. Rendl et al. [Rendl 16] explores novel opportunities for using a flexible touch and grip sensitive e-ink display as a second interactive screen for smart phones. Their prototype, called FlexCase, integrates the FlexSense [Rendl 14] foil (Section 2.1.3) and an e-ink display inside a display cover for regular smart phones. Flexible phones have also been explored in the context of wearable computing. For example, when bending a phone to a wristband, the content adjusts [Tarun 11]. Additionally, when moving or turning the arm, content moves along to ensure readability [Burstyn 15b].

Several researchers conducted user elicitation studies to gain a deeper understanding of how bends, folds, and free-form deformations intuitively map to actions in digital interfaces. Studies using PaperPhone [Lahey 11] show that there is a strong agreement in the users' mental model for performing bend gestures when spatial or directional cues play an important role. For example, there is an intuitive mapping for linking bend gestures in an icon navigation task (up/down/left/right) or page navigation in e-books. In contrast, this mapping is harder when controlling, for example, a music player. Kim et al. [Lee 10] asked their participants to deform three types of passive deformable materials: a thin plastic sheet, paper, and an elastic cloth, according to how they thought would be appropriate for various actions, including, activating/deactivating the device, zoom-in/out, copy, delete, etc. The researchers identified 7 types of deformations, including bending, twisting, folding, rolling, crumpling, tearing, and stretching. Their results show a higher agreement for materials that are more pliable. In line with these results, experiments by Kildal et al. [Kildal 12] show a preference towards more flexible materials, as these are more comfortable and accurate to bend. Instead of mapping deformations to actions in interfaces, bending gestures are also used for entering passwords [Maqsood 14] or select targets on phones that are out of reach for thumb interactions [Girouard 15].

Researchers also conducted controlled experiments to assess the performance and user experience when using bend gestures for interacting with digital information. Using a flexible input-only device connected to an external display for output, Kildal et al. [Kildal 13] compared traditional direct touch (Touch Condition) with two hybrid deformation-plus-touch input techniques: bending and twisting the device, plus either front- or back-touch (DeformTouch vs DeformBackTouch condition). While participants performed an image-docking task, all three interaction techniques showed the same efficiency in task completion. However, participants perception on their performance and physical demand while performing the tasks, was significantly better for DeformTouch as compared to DeformBackTouch or the Touch condition. More than a third of the participants also reported bending inwards to be unnatural and requiring more force than bending outwards. This last observation is consistent with study results of Warren et al. [Warren 13]. Ahmaniemi et al. [Ahmaniemi 14] compared the performance of two-implementations of bend gestures (absolute mapping, and relative mapping using speed) for performing different target selection tasks. Although absolute mapping showed improved performances, the relative bend gesture mapping showed a better match with the Fitts' Law model and scored better on user experience. In contrast to earlier studies, showing a clear preference for bending and folding interactions during directional manipulations [Lee 10], such as page navigation, speed, or height control, the performance results of Ahmaniemi et al. show that bending equally suits non-directional manipulations, such as color depth control.

A number of interfaces have also been presented that go beyond transformable 2D surfaces and tap into manually transformable 3D sculptures and objects. The seminal work by Piper et al. [Piper 02], Illuminating clay, consist of a workbench to perform hands-on landscape analysis using clay. The clay terrain model is captured and analyzed using a laser scanner and topography information is projected back on the clay model with a projector. Topobo [Raffle 04] is a construction kit for making characters, such as animals, that are animated with integrated mechanical actuators. After assembling the character, the animation is programmed by demonstrating the movements with the character. After this recording, the character repeats the demonstrated movements. Haptic Chameleon [Michelitsch 04] presents a vision for clay-like transformable physical controls that adapts its affordances and constraints to the task at hand. The Rock-Paper-Fibers prototype [Rudeck 12] partially implements this vision using a bundle of optical fibers that users manually reshapes to represent controls, including a radio-buttons, a slider, play, or pause button. However, users still interact with these controls using finger strokes.

In contrast, when interacting with real world controls, one uses a wide variety of precision and power grasps. This is the essence of the Paddle transformable phone (Chapter 6).

### 2.5.2 Actuated Shape-Changing Interfaces

Besides manual transformable interfaces, there is also a diverse track of research focusing on actuated shape-changing user interfaces. Although, changing the shape of physical materials has been used for a wide variety of purposes, we focus on their usage in interactive information systems in mobile settings. For an overview on shape-changing interfaces in other areas, such as art, fashion clothing, and furniture, we refer the reader to the article of Rasmussen et al. [Rasmussen 12].

Early versions of shape-changing phones integrate mechanical motors (e.g. servos) to control, for example, the thickness of the phone [Hemmert 10]. Changing the thickness of phones can be a subtle alert for incoming messages or reflect the number of pages left while reading a book. Alexander et al. [Alexander 12] scaled this approach to a matrix of 9 displays that independently tilt. As such, the tablet-sized form factor takes on different configuration to physically reflect the digital content or support collaborative settings. Taking this approach even further, inFORM [Follmer 13] is a shape changing table consisting of a matrix of 900 independently moving rods. By projecting content on top of these rods, the shape of the table reflects digital information or provides physical affordances and constraints for interactive controls.

To integrate mechanical actuation mechanisms more seamlessly into mobile form factors, researchers investigated integrating Shape-Memory Alloys (SMA) wires in devices. By integrating four SMA wires in an E-ink display form factor, MorePhone [Gomes 13] realizes a phone that bends all corners independently to provide various kinds of visual and haptic alerts to users. Hawkes et al. [Hawkes 10] takes these engineering principles a step further and presents a substrate consisting of custom designed shape-memory hinge actuators to automatically fold simple origami structures. To capture the wide variety of possibilities for shape change, Roudaut et al. [Roudaut 13] presented the concept of "shape resolution" according to the Non-Uniform Rational B-splines (NURBS) geometrical model. In contrast to display or touch resolution, shape resolution represents the properties of a surface to change in shape. Similar to manually transformable interfaces, researchers also conducted user elicitation studies to investigate how output parameters of information systems map to changes in shape [Pedersen 14].

# Chapter 3

## PaperPulse: Designing and Fabricating Physical Interfaces

### 3.1 Introduction

Advancements in fabrication tools for electronic circuits, such as conductive pens, threads, inkjet printers [Kawahara 13] and vinyl cutters [Savage 12] introduced accessible techniques to augment flexible substrates with electronics. Hence, an increasing number of domains, such as research, maker movement, engineering, and marketing express their interest in these technologies. Early prototypes include, low-cost paper versions of PCBs [Kawahara 13], interactive books [Qi 10, Qi 14], and posters [Shorter 14]. In the context of this dissertation, we explore using novel fabrication techniques, such as conductive inkjet printing, to make fully-integrated physical interfaces (C1). More specifically, in this chapter, we present a novel design and fabrication environment, PaperPulse, that empowers users without a technical background to make new interactive paper interfaces using these technologies (G1).

PaperPulse is an integrated software environment that seamlessly guides users and automates several aspects of the visual design, electronic designs, and programming of physical paper interfaces (C5):

- Paper constructions, such as origami structures, are often unstable and fragile. To design high-fidelity interactive paper artifacts, PaperPulse devises a special layering technique and a toolkit of custom designed widgets (Section 3.4) to ensure every design is flat and rigid. These multi-layered

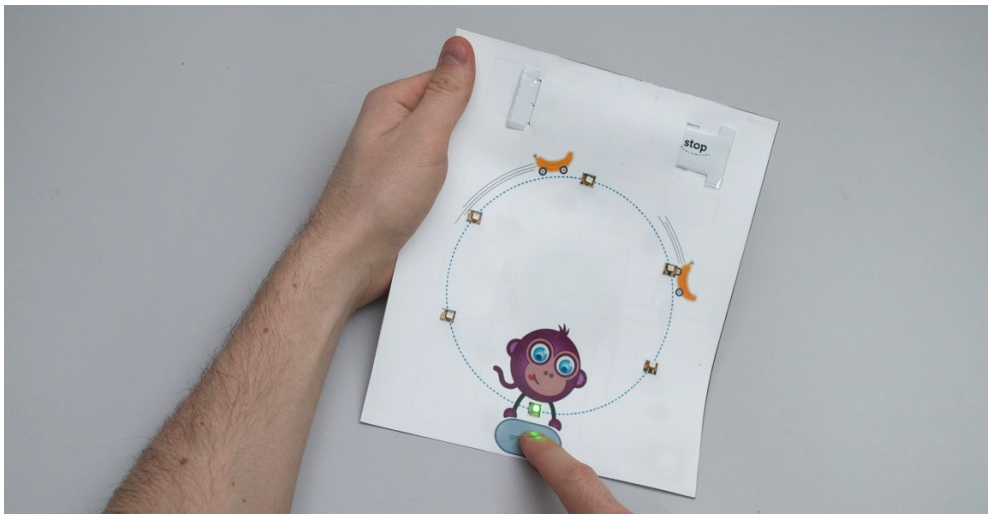


**Figure 3.1:** An interactive paper game designed with PaperPulse

paper artifacts are traditionally hard to design, since they require cutting and gluing different parts. Various aspects of a multi-layered paper design need to be consistent, such as the position of folding lines, slots, and tabs [Carter 99, Annett 15]. Additionally, brittle electronic circuit traces cannot intersect cuts or folds and it is non-trivial how to power moving parts without hard-wiring them. PaperPulse automatically handles these complex design issues when making physical paper interfaces (C2).

- To make electronics available for non-experts, construction kits targeting programmers, such as .NET gadgeteer [Hodges 13] or Phidgets [Greenberg 01], or non-programmers, such as littleBits [Bdeir 12] provide modules to rapidly build hardware prototypes. However, these kits are often bulky and have limited predefined behaviors. Thus, for instance, it is not feasible to create interactive physical paper interfaces that embed electronics in their paper designs, such as interactive greeting cards that can be handed out, or paper games as shown in Figure 3.1. PaperPulse seamlessly embeds electronics in visual designs and uses algorithms to generate a detailed electronic circuit based solely on the specified behavior of the system (C3).

- To enable non-programmers to specify the behavior of interactive paper interfaces, we present a visual programming paradigm, called Pulsation.

> With Pulsation, behavior is specified by visually creating functional links between electronic components. To verify these functional relationships, Pulsation supports a simulator to test the behavior of designs in software before fabricating the paper interface. In this chapter, we demonstrate the different constructs in Pulsation and show how they integrate within the PaperPulse design environment. Based on the lessons learned in this Chapter, we will continue exploring visual specification paradigms for physical interfaces in Chapter 5.

In the following, we first give an overview of the system. Next, a walk-through of the system shows step-by-step how to design a paper game. We then present and discuss the PaperPulse widget toolkit as well as the Pulsation visual programming paradigm. Afterwards, details on the architecture and implementation of the system are provided. Various paper artifacts designed with PaperPulse to show the validity of our approach and report on design sessions with users to demonstrates the utility and usability of PaperPulse. We end with a discussion and summary of the presented work.

## 3.2 Brief System Overview

PaperPulse is an integrated design and fabrication environment that allows users without a technical background to create physical interfaces by augmenting flexible substrates, such as paper or transparent plastic sheets, with electronics. PaperPulse guides the user through the design, electronics, and programming aspects of making interactive paper interfaces.

Figure 3.2 shows how PaperPulse streamlines the design and fabrication process of interactive paper artifacts:

a. The user adds interactive elements (e.g. push buttons, sliders, LEDs, microphones) to the visual design and specifies the logic between components using Pulsation.
b. PaperPulse generates different layers of visual elements and electronic circuit traces optimized for printing with an inkjet printer filled with conductive ink [Kawahara 13].
c. By following a custom-generated tutorial, the user attaches PaperPulse electronic stickers and assembles the different parts.
d. Next, PaperPulse generates code that is directly uploaded to the microcontroller attached to the paper.
e. The design can now be used as a standalone interactive paper interface.

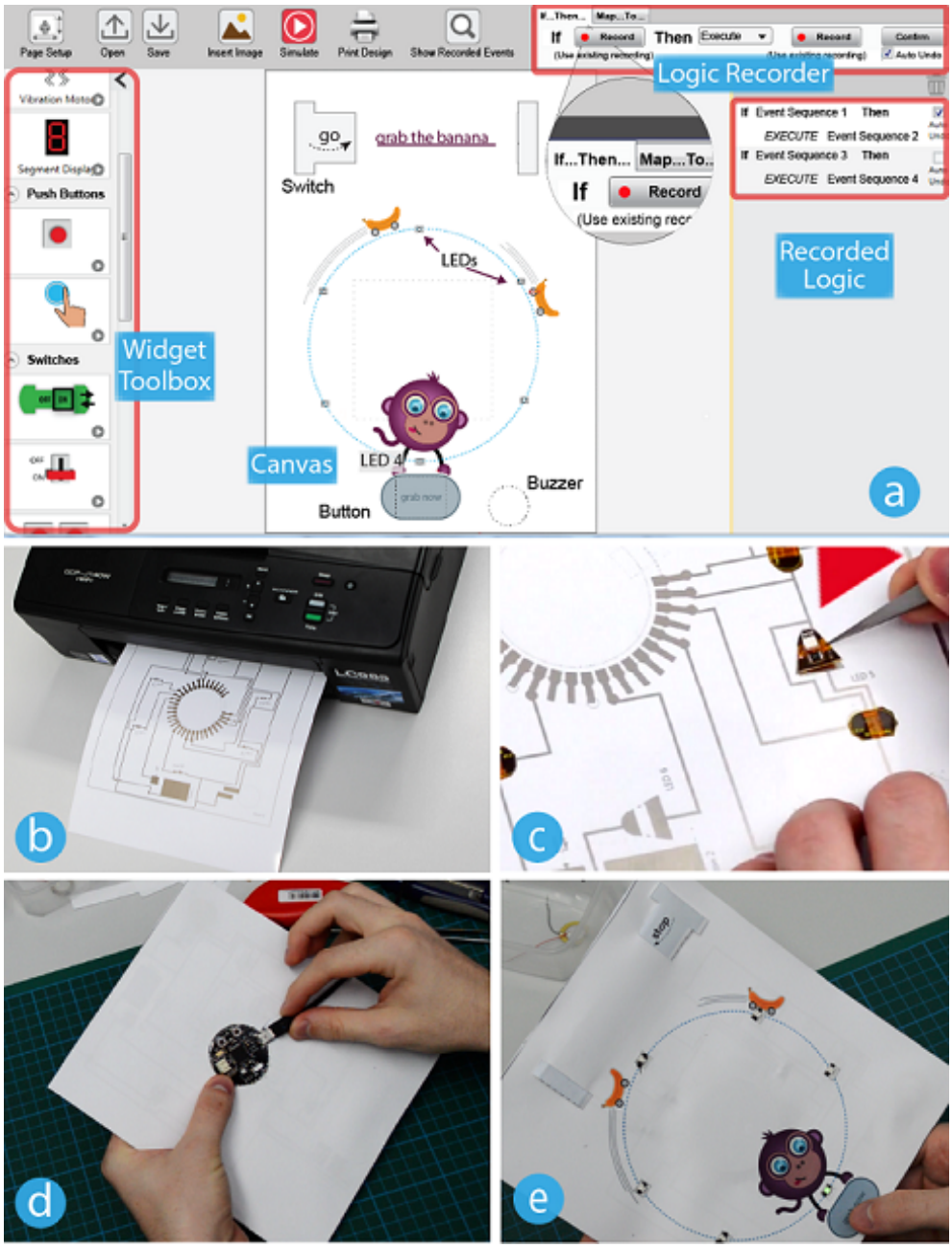


**Figure 3.2:** The *PaperPulse* workflow streamlines the entire process of creating interactive paper artifacts. (a) Design and specify logic; (b) print sheets; (c) assemble; (d) upload generated program to microcontroller; (e) final paper artifact.

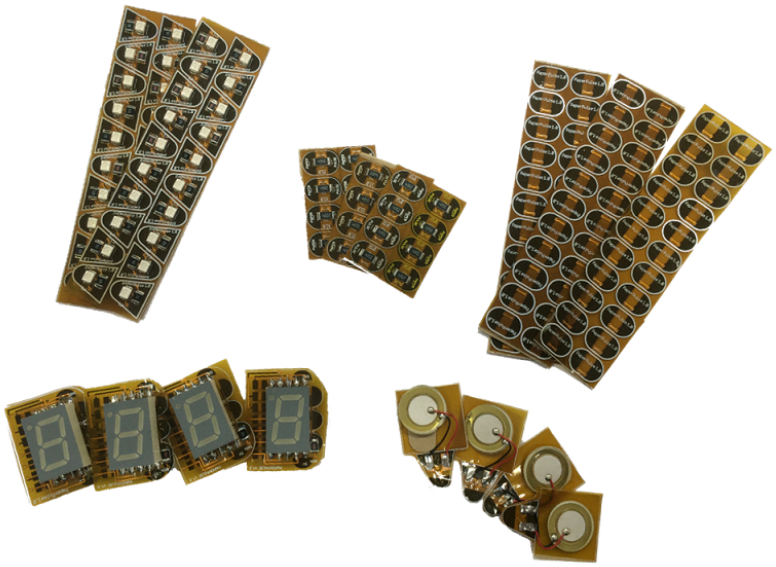

**Figure 3.3:** Some of the PaperPulse electronics stickers: (a) LEDs, (b) resistors, (c) bridges, (d) seven-segment displays, (e) buzzers

Although electronic circuits traces generated with PaperPulse can be fabricated using various techniques (e.g. a conductive pen, vinyl cutter), the circuits traces are optimized for printing on resin coated substrates (e.g. paper or plastic) using a conductive inkjet printer [Kawahara 13]. Besides these circuit traces, PaperPulse includes a physical toolkit of electronic stickers (Figure 3.3), to finalize the design. These stickers have a conductive and adhesive back and are easy to attach firmly to printed circuit traces. Different variations of these stickers exist to support a wide variety of components including, circuit bridges, resistors, buttons, switches, LEDs, seven-segment displays, buzzers. In contrast to traditional multi-layered PCBs, our resin coated substrates only have a single layer. When the circuit requires multiple layers, electronic circuit bridge stickers are used to bridge circuit traces and realize complex non-planar circuit graphs (Figure 3.2c).

Our design tool allows users to focus on the graphical design thus users only interact with a single canvas. However, every design has a unified layering approach consisting of 3 layers: a base layer, a top layer, and an optional widget-specific layer (Figure 3.4). The base layer includes the main electronic circuit while the top layer mainly consists of graphical visual elements. De-

pending on the type of widgets (section 3.4) used in the design, a third, widget-specific layer is required. As shown in Figure 3.4a-b-c, every widget used in the design, contributes elements to multiple layers. These elements consist of circuit traces, visual elements, and assembly instructions, such as dotting lines for cutting, dashed lines for folding, and hashed regions for gluing (Figure 3.4). Some sheets (i.e. the top layer) also have information present on the back of the paper while others require conductive as well as non-conductive information on the same page. The custom generated tutorial guides users to correctly align paper in the conductive inkjet printer, or a regular color printer for non-conductive elements. This layering approach is also vital for the seamless integration of electronics and visual elements, since all conductive traces are concealed.

PaperPulse supports both Netduino[1] and Threadneedle[2] development boards. Pins on the Netduino connect to paper circuits using bulldog clips. In contrast, Threadneedle exposes flat connection pins that seamlessly attach to the printed circuit using Electrically Conductive Adhesive Transfer (ECATT) tape (Figure 3.2d).

## 3.3 Walkthrough: The Hungry Monkey Game

The following walkthrough illustrates the process of designing and fabricating a paper game with PaperPulse (Figure 3.2). The game consists of a loop of six LEDs that consecutively turn on and off. The objective of the game is to "grab the banana" by pressing a button at the moment when a particular LED lights up. A buzzer rings for a short duration each time the player succeeds in doing so.

### Step 1: Designing the Interactive Paper Layout

The user starts by specifying the dimensions of the paper design. PaperPulse then allows to import pre-designed visual elements (i.e. images) and to arrange them onto the canvas (Figure 3.2a). Next, the user overlays the design with interactive components (six LEDs, a buzzer, a pull switch, and a button), available in the widget toolbox (Figure 3.2a). To give users a better idea of the look and feel of different components, tooltips with video previews [Grossman 10] are available in the widget toolbox.

[1] www.netduino.com
[2] modlab.co.uk

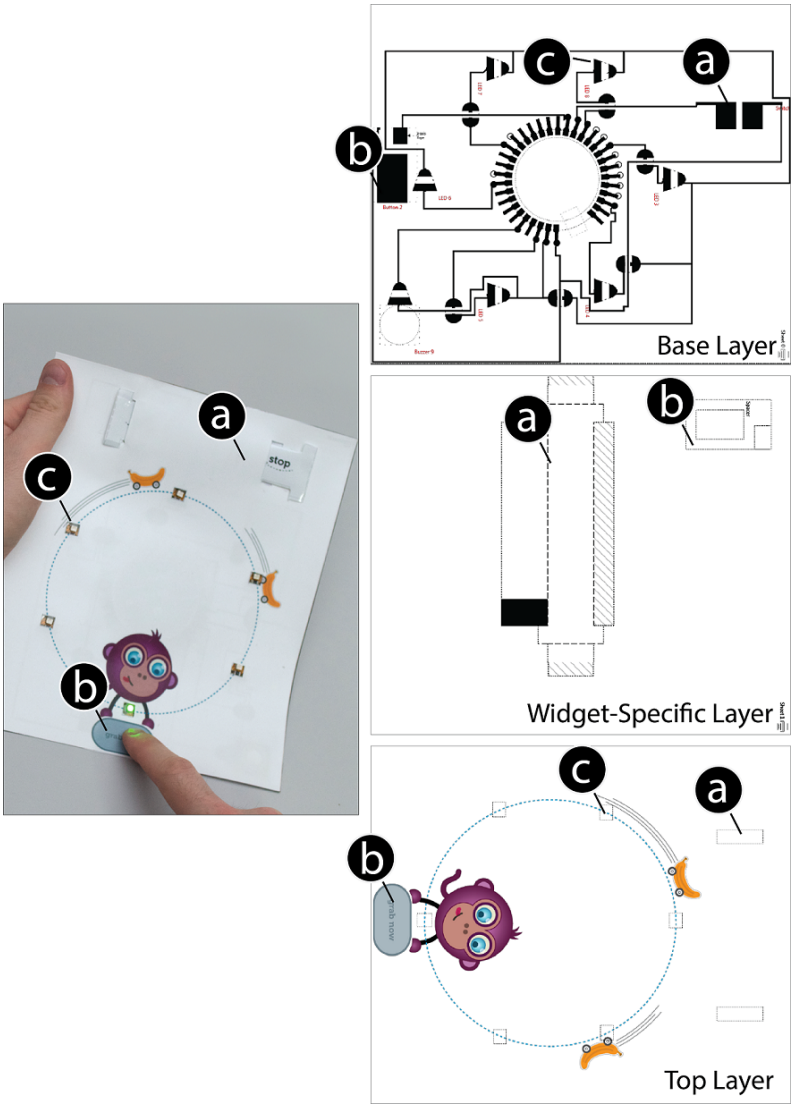


**Figure 3.4:** Widgets in a design e.g. (a) pullchain switch, (b) paper-membrane push button, (c) electronic circuit LED sticker, contribute content to multiple layers of a PaperPulse design

### Step 2: Defining and Verifying Logic Iteratively

Figure 3.5 illustrates how the user links the switch to the loop of LEDs, to start the game:

a. The user starts a new input recording in the *if*-part of the logic recorder.
b. She demonstrates the switch changing to the 'on' state using the widgets on the canvas.
c. The user then starts a new output recording in the *then*-part of the logic recorder.
d. She demonstrates the blinking pattern of the LEDs by turning their brightness consecutively to 100% and back to 0%.
e. Next, the user specifies the timing for these recorded actions by setting them to occur at intervals of 0.3 seconds. She also specifies the looping behavior by setting the loop option to *Infinite*.
f. When the *if–then rule* is confirmed, PaperPulse automatically infers the behavior for the 'off' state of the switch.

To verify the recorded rule, the user starts the simulator to interact with the widgets and observes the corresponding output (Figure 3.7). By observing fulfilled conditions and executed actions in the *Debug View*, the user can identify possible mistakes in the recorded rules.

Next, the user records the logic for the "grab now" button. If pressed at the moment the LED under the monkey ("LED 4") lights up, the buzzer should ring to indicate that the game is completed. Figure 3.6 illustrates the recording of this behavior:

a. She records the *if*-part of the logic by demonstrating the button press and turning the brightness of LED 4 to 100%.
b. The recording is fine-tuned by specifying that the two conditions need to be satisfied *simultaneously*.
c. The user records the *then*-part of the logic by turning the volume of the buzzer to the desired intensity.
d. Next, she specifies the timing of the output action (buzzer ringing) to ensure that the buzzer stops after two seconds.

### Step 3: Printing and Assembly

Once the design is complete, the user specifies the position of the microcontroller and verifies that electronic connection pins for the widgets do not overlap. The user can adjust widget properties like position, size, and orientation accordingly.

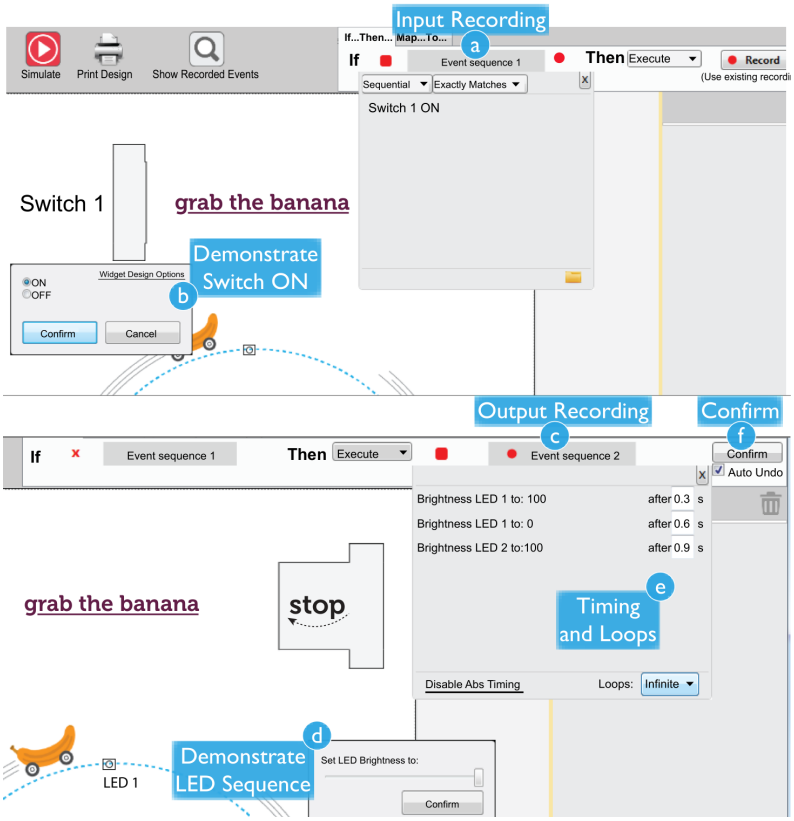


**Figure 3.5:** Recording an *if–then* rule for the LED sequence when a switch is turned on.

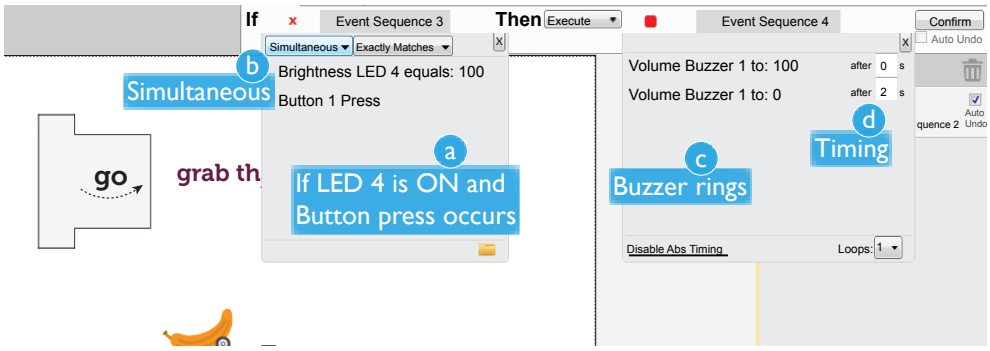


**Figure 3.6:** Recording another rule to ring a buzzer if the button is pressed at the moment "LED 4" turns on.

The printing process starts by generating:

1. An electronic circuit that connects widgets to pins on the microcontroller while limiting the number of intersecting circuit traces.
2. PDF files consisting of the electronic circuits, widget-specific assembly lines (e.g. cut lines, fold lines), and visual elements.
3. Microcontroller code.
4. A customized tutorial to guide the user through the printing, deployment, and assembly.

Following the tutorial (Figure 3.8a), the user is instructed to print the generated PDF files on three sheets of paper, using a conductive inkjet and a color printer, as required (Figure 3.8b). She then attaches electronic bridge stickers (Figure 3.2c) at intersecting traces that could not be resolved by the auto-routing algorithm. The remainder of the tutorial provides instructions to cut, fold and glue layers of paper, attach electronic components, such as LEDs, resistors, and attach the microcontroller and upload the generated code (Figure 3.8c-d).

As shown in Figure 3.2e, the resulting end-product can now be used as a standalone paper game after connecting a battery.

## 3.4 PaperPulse Widget Toolkit

Designing reusable widgets that can be printed or connected reliably to circuit traces is non-trivial. Challenges include, realizing high-quality electronic

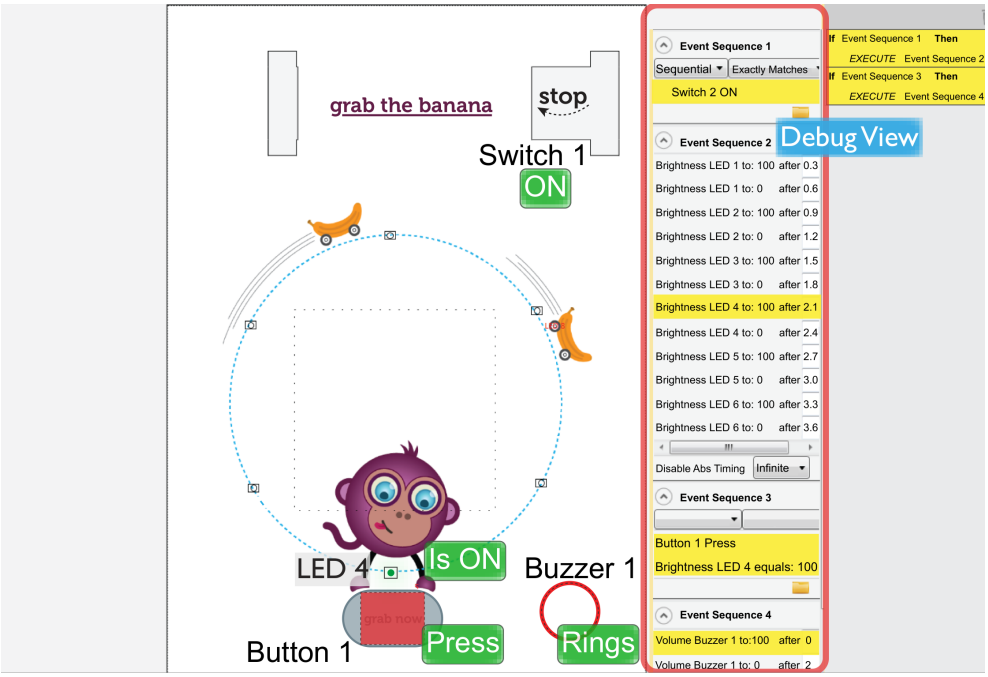


**Figure 3.7:** The PaperPulse Simulator enables testing of recorded rules.

connections between contact pads on flexible substrates, ensuring continuity of brittle circuit traces over folding structures, powering of moving parts, increasing the firmness and durability of widgets and the entire interactive paper interface. To provide users with appropriate widgets, suitable for their interactive paper interfaces, we present three families of standard widgets: electronic circuit sticker widgets, paper-membrane widgets, and pull-chain widgets. Every widget family realizes basic controls such as push buttons, switches, sliders, and radio buttons. Each family is unique in its own way, and provides some strengths to distinguish itself from the others. Figure 3.9 illustrates how each approach realizes a linear slider.

The three widget families consist of a different number of layers. Our uniform three-layering approach (Figure 3.4) allows widgets of all three families to co-exist in a single design. Every widget design ensures that all conductive lines are traced back to the base layer, which is connected to the microcontroller.

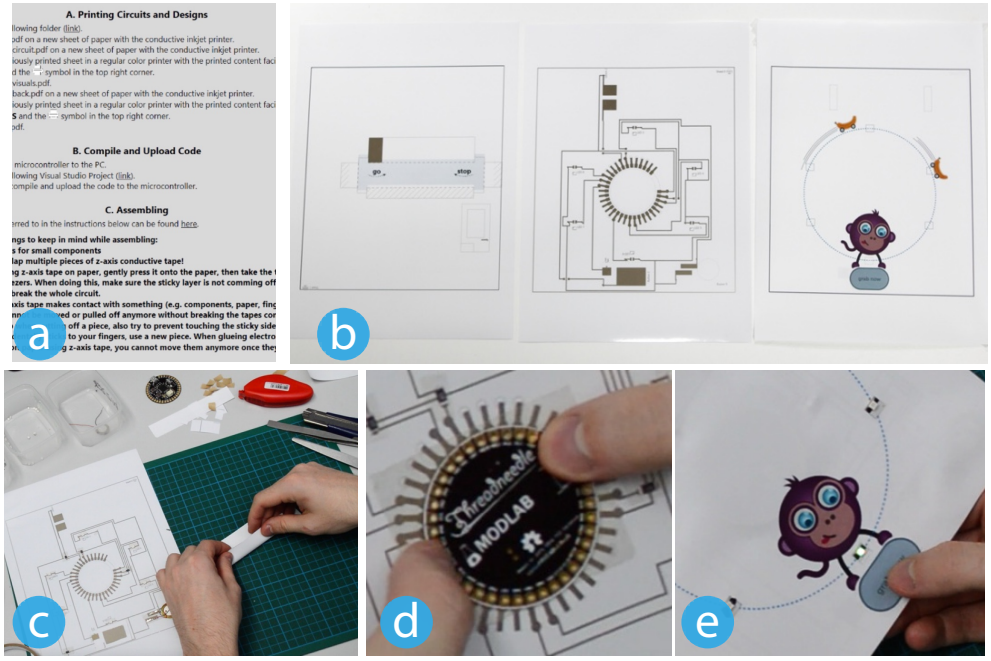


**Figure 3.8:** Printing and assembling process: (a) instructions generated; (b) sheets printed (c) circuit and widgets assembled; (d) generated code uploaded to the microcontroller; (e) the final paper artifact.

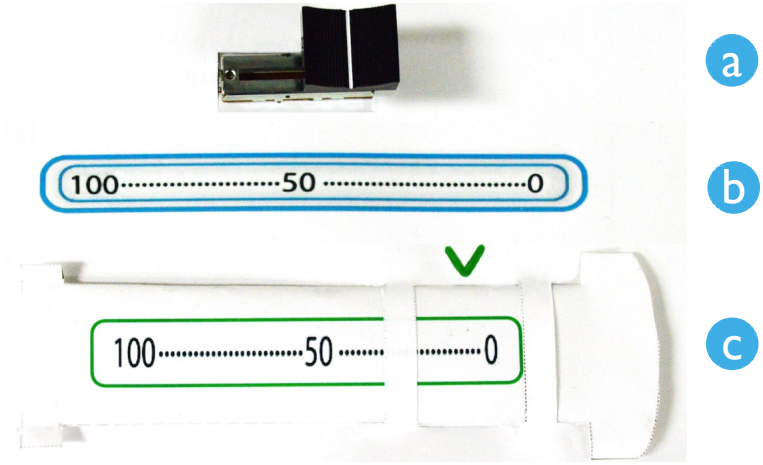


**Figure 3.9:** The three families of PaperPulse widgets: (a) Electronic circuit sticker slider; (b) Paper-membrane slider; (c) Pull-chain slider.

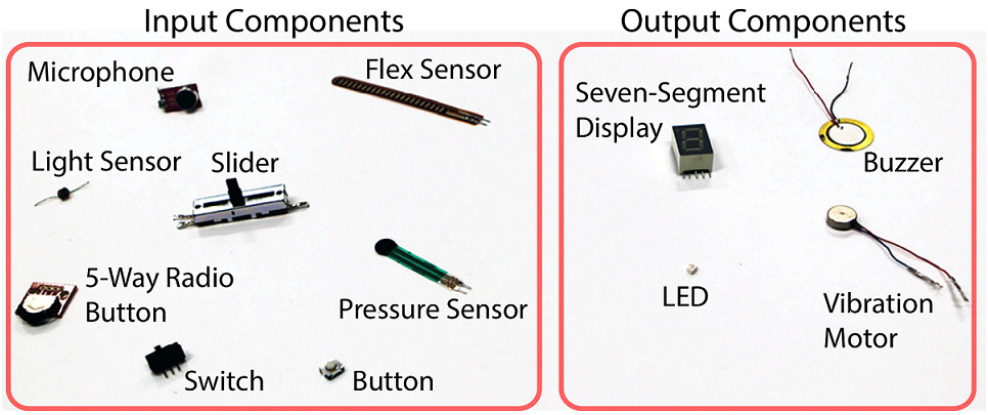


**Figure 3.10:** The electronic circuit sticker widgets currently supported by PaperPulse.

### 3.4.1 Electronic Sticker Widgets

PaperPulse currently supports eight circuit sticker input sensors and four output components (Figure 3.10). Some components expose flat connection pins on the bottom (SMDs[3]) while others have regular pins (through-hole components). The connection terminals of both mounting technologies are too small to be connected reliably onto substrates using ECATT-tape or conductive paint[4]. Therefore, these components are first connected to a custom-fabricated flexible PCB substrate that exposes larger contact pads on the bottom of the PCBs (Figure 3.3). We refer to these type of widgets as electronic circuit stickers since they expose a conductive and adhesive back (using ECATT-tape) and are therefore easy to attach firmly to printed circuit traces. Many electronic circuit stickers also integrate multiple electronic components that are always used in combination. Examples include, resistors for LEDs or shift registers for seven-segment displays. Hence electronic circuit stickers also reduce the number of components users have to attach manually.

Although electronic circuit sticker widgets require only little manual assembly, they have a fixed design and often protrude from the surface. When augmenting paper designs with electronics, it is often desirable to resize components and integrate them seamlessly with visual elements on paper. This is accomplished with *paper-membrane* and *pull-chain* widgets.

[3]Surface Mounted Devices
[4]www.bareconductive.com

### 3.4.2 Paper-Membrane Widgets

Figure 3.11 shows two paper-membrane widgets. The main design rationale behind paper-membrane widgets is to create an electronic circuit between the base layer and back of the top layer and separate them with thin air gap using a paper frame (widget-specific layer) that serves as a spacer (Figure 3.11a). Pressing on the top layer connects it to the bottom, closing the circuit and thus realizing a push button. The top layer is powered from the base layer by connecting regions *Z1* and *Z2* using ECATT-tape.

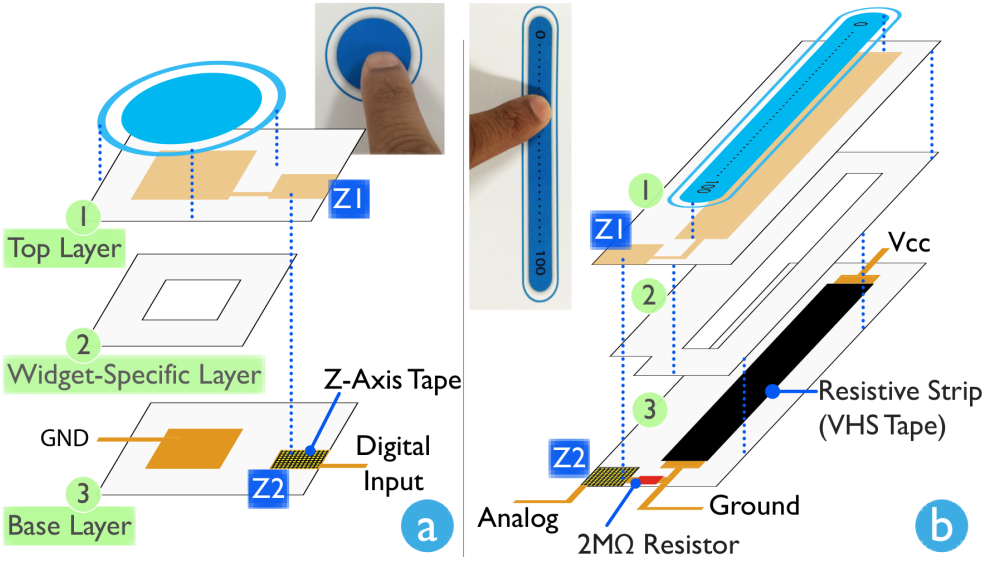


**Figure 3.11:** Design of paper-membrane widgets: (a) push button (b) slider.

Figure 3.11b shows the design of a paper-membrane slider in which the principle of a voltage divider is applied to measure the position where the top (wiper) and base layer make contact. A resistor of $2M\Omega$ in the design serves as a pull-up resistor to detect when the slider is not touched. To increase sensor resolution, the resistive strip should have a large resistance range. Although resistive strips can be printed (by reducing the opacity, and hence quantity of conductive ink) or drawn using graphite [Holman 11], we noticed that due to wear-and-tear the resistance of these strips often changes at frequently touched spots. For paper-membrane sliders, we therefore use resistive 8 mm VHS tape[5] as sensor strip, resulting in a more durable paper-membrane slider.

[5]Several other kinds of tapes could also exhibit linear resistance.

Paper-membrane widgets support radio buttons and switches by incorporating multiple paper-membrane push buttons in a single widget with a shared software state. In contrast to electronic circuit sticker widgets, paper-membrane widgets are customizable. On the other hand, they do not offer tangibility. This is the essence of pull-chain widgets.

### 3.4.3 Pull-Chain Widgets

Pull-chain widgets draw inspiration from planar paper pop-up mechanisms [Carter 99]. Similar to electronic circuit sticker widgets, pull-chain widgets provide tangibility but at the same time do not protrude from the surface. Since they are designed entirely out of paper, pull-chain widgets are customizable and blend seamlessly into paper designs.

Although pull-strip mechanisms are traditionally used as sliding mechanisms [Qi 10], we see them as omnivalent pulling mechanisms in the same way as old-fashioned pull chains were used to control electrical appliances, such as light bulbs and fans. Figure 3.12 shows a pull-chain switch, slider, radio button and push button (using a crossing interaction technique [Apitz 04]).

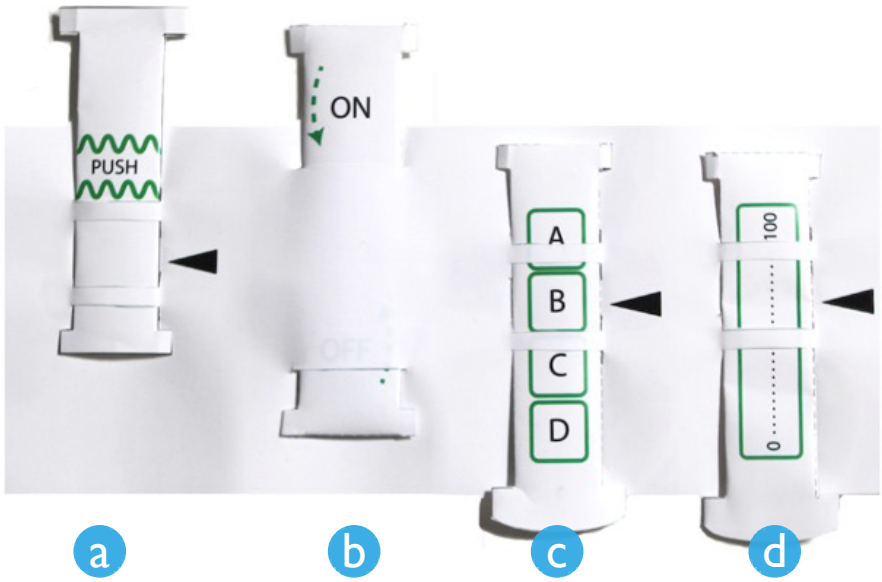


**Figure 3.12:** Pull-chain widgets supported by PaperPulse: (a) Push-button, (b) Switch, (c) Radio button, (d) Slider.

The mechanisms used for pull-chain widgets are optimized for tracking with electronic circuits printed on paper. These conductive traces are often brittle and cannot span across folded structures. As shown in Figure 3.13, the mechanical design of every pull-chain widget consist of the following parts:

a. A folded tube structure with a hollow center to ensure strength and rigidity during pulling and pushing motions.
b. Paper slots to guide the pull-strip.
c. A wing tab to lock the pull-strip in place.
d. A pull-tab that functions as handle.

Since the tube-structure of the pull-strip is interwoven in the top layer, this provides sufficient pressure between the pull-strip and the base layer to ensure electrical connectivity. At the same time this construction provides an acceptable amount of friction to manipulate pull-chain widgets comfortably.

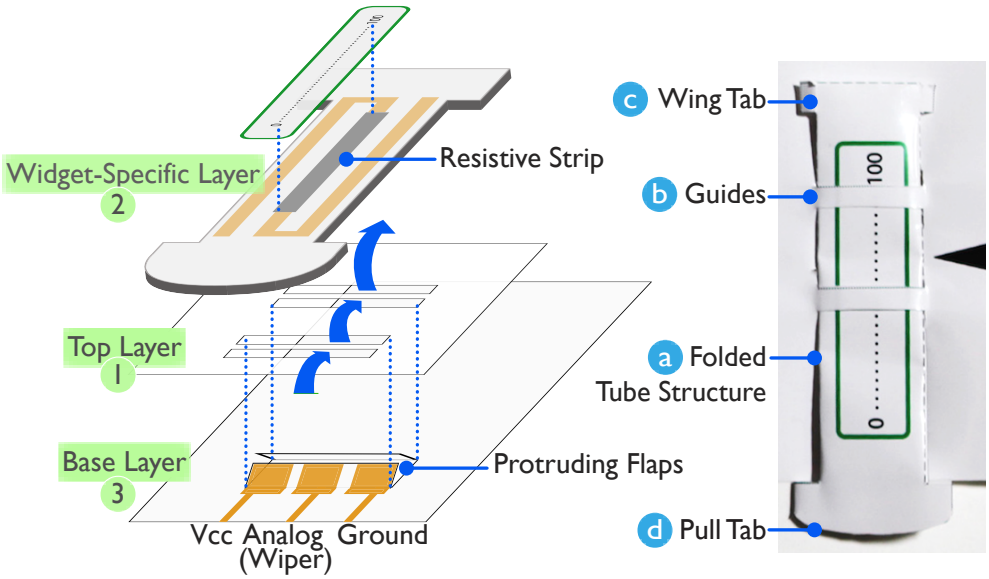


**Figure 3.13:** Design of pull-chain widgets: The widget-specific layer is interwoven into the top layer by passing it through four slots. Protruding flaps on the base layer also pass through these slots to ensure constant contact between the winding circuit traces on the pull-chain and the three pin connections on the base layer.

Figure 3.13 also shows the electrical circuit design specifically for pull-chain sliders. This consists of an analog sensor strip (8 mm VHS resistive tape) and

winded circuit traces on the back of the pull-strip. Pull-chain radio buttons use the same approach but software thresholds are used to realize discrete states. In contrast, pull-chain push buttons and switches consist of conductive patches at specific spots that make an electronic connection when the strips are pushed or pulled. Push buttons, switches and radio-buttons usually employ mechanical detent mechanisms. These techniques however do not transfer to paper since paper is too fragile. To avoid undesired oscillations when widgets are in between states, hysteresis and timeouts are used in software.

### 3.4.4 Summary of PaperPulse Widgets

In order to provide users a wide variety of widgets in PaperPulse, we presented three families of standard widgets. As shown in Table 3.1 each design offers its own strengths and limitations.

| | **Circuit Sticker widgets** | **Paper-Membrane Widgets** | **Pull-Chain Widgets** |
|---|---|---|---|
| **Interaction Style** | Tangible | Touch | Tangible |
| **Minimal Assembly** | ✓ | – | – |
| **Seamless Integration (Non-Protruding)** | – | ✓ | ✓ |
| **Customizable** | – | ✓ | ✓ |

**Table 3.1:** Strengths and limitations of PaperPulse widget families from a user perspective.

## 3.5 Pulsation: Specifying Functional Relationships between Electronic Components

Pulsation allows users to graphically specify logic by demonstrating and recording actions directly in the context of the visual design elements. Demonstrating actions using a graphical user interface, however, is limited to actions that can be defined through the graphical interface of the tool. For example, demonstrating multiple actions that need to occur simultaneously is impractical using a regular mouse and keyboard. Similarly, specifying a set of actions that can be performed in any order, requires demonstrating all different possible orderings one by one. To address these challenges, and provide a higher ceiling than is possible with demonstration alone, Pulsation augments widgets and the demonstrated actions with dialogs that allow fine-tuning of specific properties (Figure 3.5). At the same time, demonstrating actions in the context of visual design elements calibrates the state of the input widget to real world values

that are present in the visual design. This makes it possible, for example, to gauge a slider by demonstration, or choose which state of a switch is high or low.

Pulsation consists of a grammar and interpreter. The interpreter can be deployed on desktop computers as well as microcontrollers supporting the .NET Micro Framework. The Pulsation grammar consists of *if–then* as well as *map–to* rules, as shown in Figure 3.2a. For *if–then* rules, a set of recorded actions (*output set*, or the *then*-part) is executed when a set of recorded conditions (*input set*, or the *if*-part) has been met. For map–to rules, parameters of *input set* (e.g. the number of fulfilled actions in the set) are continuously mapped to parameters of the *output set* (e.g. speed with which the set of actions are executed repeatedly). Both *if–then* and *map–to* rules thus relate an *input* set to an *output* set.

### 3.5.1 Input Sets

Input sets specify conditions (guards) to be fulfilled. Input sets therefore consist of one or more conditions related to input or output widgets. Three types of conditions are supported by Pulsation: (1) *Momentary input conditions*, are true for only a very brief amount of time, such as a pressing or releasing a push button. (2) *Discrete state conditions* are true until the widget switches to another state e.g. the modes of a switch, a discrete brightness value of an LED or the pressed state of a push button. (3) *Continuous range conditions* are true when the current value of a continuous input widget is within a specified range, such as a specific range of a slider or the volume range of a buzzer.

As shown in the walkthrough, Figure 3.6a gives an example of an input set that is fulfilled when a push button is pressed at the same time that an LED lights up. Essential here are the timing options offered by input sets (Figure 3.6b). These options allows one to specify conditions that need to be met *simultaneously*, *sequentially* or in a *random* order. When timing options are different for some conditions in the set, these conditions are grouped in separate layers.

Using the conditions and timing options provided by input sets, simple patterns of conditions can be recorded that need to match with the incoming stream of events. Pulsation supports two matching approaches: (1) The *include matching* approach requires the stream of all incoming events to fulfill the pattern of conditions specified in the input set. Other events which do not fulfil any conditions in the set are also allowed. (2) The *exact* matching approach, does not allow events that do not fulfill any of the conditions in the input


**Figure 3.14:** An invitation card with a code slot designed using PaperPulse: (a) Every time the user enters a correct number of the year of birth of the sender, (b) one more LED lights up. (c) When all four numbers are pressed in the right order, (d) the date of the birthday party appears.

set. Figure 3.14a shows an input set that uses the *exact matching* approach in combination with the *sequential* timing option to enforce end-users to press specific buttons in a certain order without pressing other buttons in the mean time, thus realizing a digits code slot.

### 3.5.2 Output Sets

Output sets consist of one or more output actions. Pulsation supports two types of output actions: (1) *Discrete output actions*, such as lighting up an LED, setting the digit of a seven-segment display or a monotonic tone of a speaker. (2) *Range output actions* specify an output range that has be transitioned. An optional time parameter can be specified by the user. Examples include, fading an LED in or out or realizing a count-down or count-up with a seven-segment display.

As already shown in Figure 3.5e, output sets allow to specify delays between recorded actions. Besides this, the *loop* construct offers the possibility to execute the set of actions multiple times.

### 3.5.3 If–then Rules

One way to relate input to output sets with Pulsation is using if–then rules. These rules allow to *execute* or *stop/reset* an output set when all conditions of an input set are met. An existing output set can also serve as input set for another if–then rule, thus allowing for nested rules. Or-relations are indirectly supported using multiple if–then rules.

Figure 3.14c-d, shows the if–then rule needed for realizing a code slot. When the correct code is entered, in this case the year of birth of the sender

of the invitation card, the date of the birthday party is revealed on a seven-segment display.

When input sets solely consist of stateful conditions, i.e. *Discrete state conditions* and *continuous range actions*, it is often desirable to undo all actions performed in the output set once the conditions in the input set are not fulfilled. Specifying all these "undo" if–then rules manually can become cumbersome, especially when widgets have many modes (e.g. radio buttons). For example, turning the switch, discussed in the walkthrough (Figure 3.5), to the on-state starts the game, and thus the blinking of the LEDs. Turning it to the off-state should turn off the LEDs. Pulsation automatically infers for every if–then rule whether this undo is appropriate and will suggest to automatically undo all state changes, caused by this rule, when the input set is not fulfilled anymore. Pulsation considers the undo operation to be appropriate when the input set only consists of stateful conditions.

### 3.5.4 Map–to Rules

Map–to rules allow for linear mapping of a derived parameter of the input set to another parameter of the output set. For example, mapping the volume of a microphone or speed with which a push button is tapped to the number of LEDs that light up or the frequency with which they blink.

Pulsation supports numerous derived parameters for both input as well as output sets. The mapping parameter can be different for the input and output set, so many combinations are possible.

- *Value* (only for input sets that consist of a single *continuous range condition* and output sets that consist of only *range output actions*): The current value in the range is used as mapping parameter.
- *Progress* (only for input/output sets that consist of at least two actions): As mapping parameter for input sets, the number of fulfilled actions is used. As mapping parameter for output sets, a corresponding number of actions of the set is executed sequentially.
- *Repetition* (only for input sets): The number of times the conditions in the input set are fulfilled is used as mapping parameter.
- *Time* (only for input sets): The duration that all actions in the input set remain fulfilled is used as mapping parameter.
- *Speed*: As mapping parameter for input sets, the speed with which the input set is repeated is used. As mapping parameter for output sets, the actions in the set are repeatedly executed at a certain speed.

Figure 3.14a-b shows how a map–to rule is used to visualize the end-users' progress while entering the code on the birthday invitation card. Here the *progress* through the input set (pressing buttons *sequentially*), is mapped to the *progress* of different LEDs that light up.

## 3.6 Architecture and Implementation

PaperPulse is implemented in .NET/C#. This section describes the architecture and algorithms underlying the PaperPulse system.

### 3.6.1 Generating Electronic Circuits

PaperPulse employs an auto-routing algorithm to generate conductive traces that connect the pins exposed by widgets to the pins of a microcontroller. We implemented a variation of the A* algorithm in which traces can make junctions with other traces that connect to the same pin. Our routing algorithm traces straight horizontal and vertical paths with a thickness of 0.6mm. To guarantee a minimum spacing of 1.5mm between all circuit traces, the routing algorithm uses a grid with cell sizes of 2.1mm (thickness of conductive paths + spacing). The final circuit traces are centered in these cells. Figure 3.15a shows the final circuit design of the diet card example design (Section 3.7). Figure 3.15b shows how this circuit is represented in the circuit routing canvas. The canvas consists of 96x81 cells/pixels as the design measures 170x200cm. Note that the circuit tracing canvas in Figure 3.15b is enlarged for visualization purposes.

To realize a working electronic circuit, the algorithm routes circuit traces one by one and updates the cost of all cells every iteration. Cells with conductive traces or instructions have a high cost and are therefore avoided. When the circuit is non-planar however, the algorithm interrupts one of the intersecting traces and leaves place for an electronic circuit bridge sticker. As the contact pads of bridge sticker components take up space, only circuit trace segments that are surrounded with enough empty space are eligible for bridging. Figure 3.16 shows two invalid locations: (a) bridges close to corners and (b) bridges covering multiple circuit traces. To ensure bridges are only placed at valid locations, we devised an algorithm that extracts invalid bridge locations. Figure 3.15c shows all invalid bridge locations in final circuit design of the diet card example design.

When routing circuit traces, our algorithm inspects which positions on the existing circuit traces are valid bridge locations, by inspecting the surrounding area. In this process, the algorithm uses a series of morphological operations.

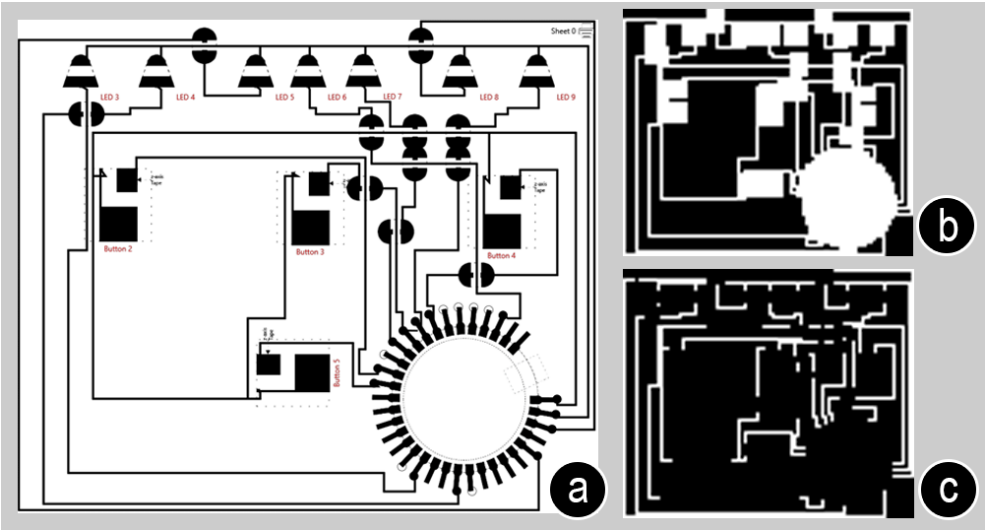


**Figure 3.15:** Detecting invalid bridge locations: (a) Automatically generated circuit diagram for the diet card example. (b) Representation of the circuit in the circuit routing canvas. (c) All invalid bridge locations at circuit traces (white).


**Figure 3.16:** Invalid placement of bridge sticker components: (a) close to corners, (b) bridges covering multiple circuit traces.

Figure 3.17 illustrates this algorithm with a simple example in which one pin of a button has to be connected to one of the pins at the edges of the substrate, in the presence of other traces. The algorithm first creates a binary image with all conductive regions (b) and a filtered representation that includes only the routed traces (c). The former image represents all conductive regions that are potentially in conflicting neighborhoods of a bridge. The latter one highlights all potential bridge locations. As bridge stickers can bridge both horizontal and vertical circuit traces, we consider these cases separately. From representation (c), two images are extracted that contain respectively all vertical and horizontal traces using an erosion morphological operation (e). The length of the structured elements used in these operations matches the length of a bridge sticker component. To verify which locations on the canvas have enough space for a bridge to be placed vertically or horizontally, two dilation morphological operations with the respective structured elements shown in (d) are applied on representation (b). The size of these structured elements match the size of a bridge sticker component. The dark pixels in the final two representations of (d), that correspond with pixels of circuit traces in (e), represent valid bridge locations. To extract these valid bridge locations, we first extract invalid bridge positions in the horizontal and vertical direction by masking the vertical and horizontal representations in (d) with the respective vertical and horizontal line detection masks in (e). This results in respectively (f) and (g) of which the white pixels represent invalid bridge locations. Inverting both representations and applying circuit trace mask (c), results in all valid bridge locations in respectively the vertical (h) and horizontal direction (i). (j) Shows the combined result for both directions. Inverting this representation and applying again circuit trace mask (c), results in all invalid bridge locations at circuit traces.

The highlighted pixels in Figure 3.17k are used as invalid locations in the next iteration of the circuit routing algorithm that connects the last pin of the circuit. Figure 3.18 shows the result after the last circuit tracing iteration, the A* algorithm bridged one of the circuit traces at a valid location and connects the button to pin 4.

In Figure 3.17a, the top two horizontal circuit traces, linking to 5V and ground, are positioned close to each other. As it is not possible to cross multiple circuit traces with bridge sticker components, both circuit segments are correctly marked as invalid bridge locations in Figure 3.17k. Circuit traces that are positioned close together oftentimes make it extremely challenging for the circuit routing algorithm to connect all electronic components. For example, when the first to routed traces are positioned near each other and span

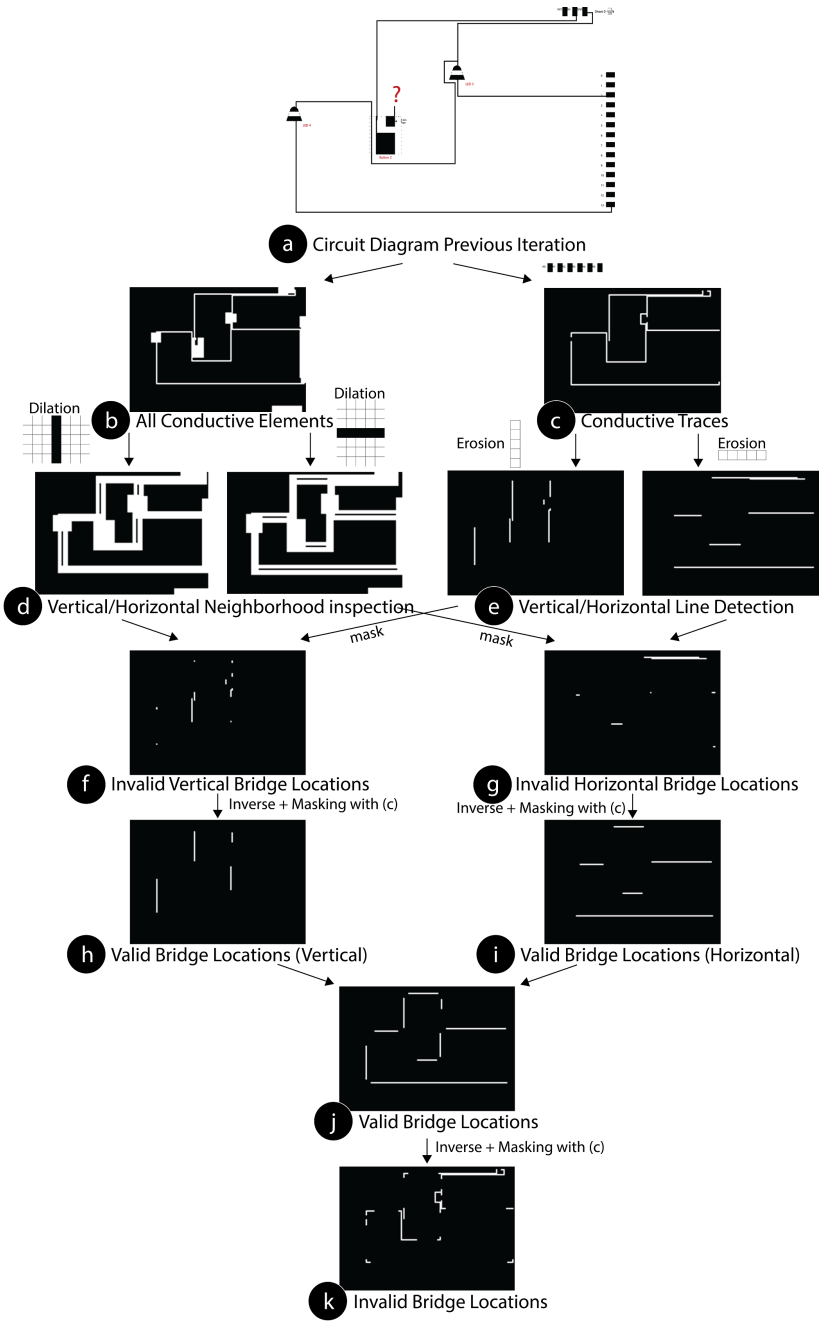


**Figure 3.17:** Morphological operations used to identify valid bridge locations.

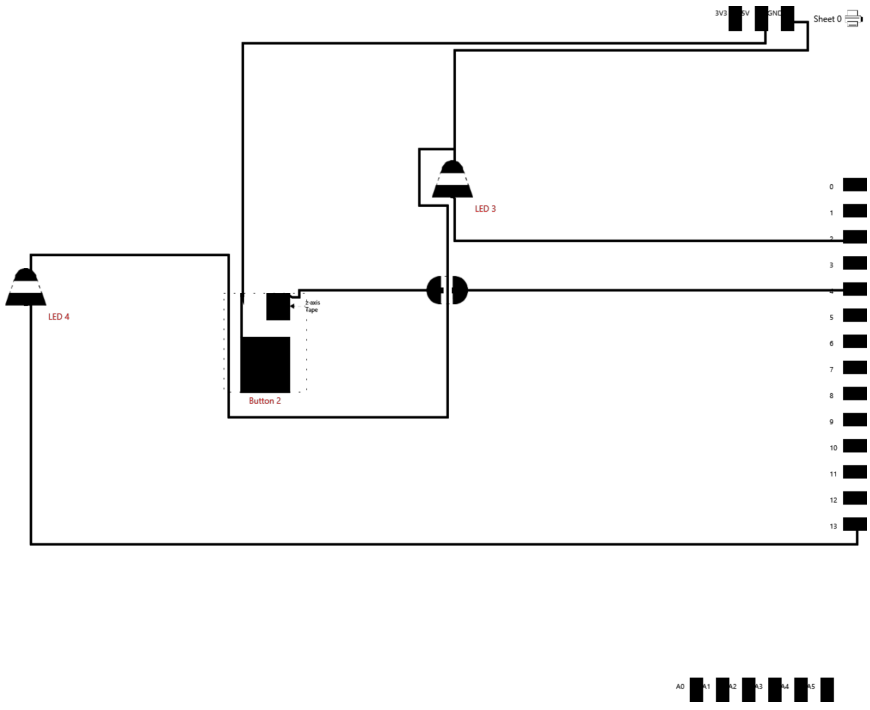


**Figure 3.18:** Electronic circuit after the last iteration of the A* algorithm.

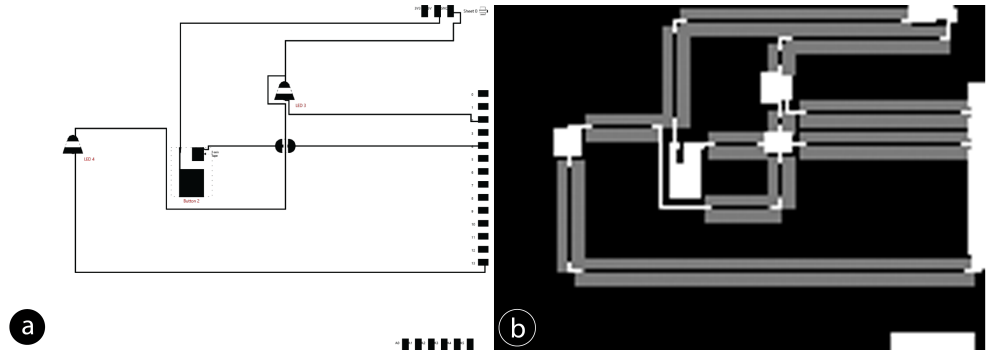


**Figure 3.19:** The circuit routing algorithm favors additional space between circuit traces.

the entire length or width of the substrate, it is not possible anymore to reach the other side of the substrate. Especially when the complexity of electronic circuit increases (e.g. Figure 3.15), this problem emerges. Therefore our algorithm adds additional weights to the surrounding cells of valid bridge locations to force the circuit routing algorithm to first consider cells that are located further from the traces already routed. Figure 3.19a shows the result of this additional step. Compared to the previous result shown in Figure 3.18, the circuit traces at the top connecting to the 5V and ground are spaced further apart and allow for bridges. The 3.3V pin now becomes more convenient to reach from multiple positions on the canvas. Figure 3.19b shows the routing grid after the final iteration of the circuit tracing algorithm, the lighter the pixel the higher the cost of the cell.

Control pins of electronic components can often be connected to multiple pins on a microcontroller. This depends on the input or output signal that is required. For example, the anode of an LED can be connected to any PWM pin. However, if binary output suffices, a digital pin can be used. Our routing algorithm takes this into account and first uses the specified logic to assign a set of valid control pins to every component. The algorithm then selects those pins that maximize the number of components that can be connected given the limited set of pins on the microcontroller. Although this technique allows for realizing arbitrarily advanced circuit designs that utilize all resources of the microcontroller, bridge sticker components take up space on the substrate. This space is also used for components and circuit traces. When the routing algorithm cannot find a valid solution for tracing all electronic components to microcontroller pins, the algorithm will change the order in which pins of

electronic components are traced to the microcontroller. A similar strategy is taken for clusters of intersecting traces. Changing the order in which traces are routed oftentimes allows for routing more circuit traces and can lead to a lower number of intersecting traces. When the algorithm cannot find a valid solution after backtracking all possible solutions, it assumes that there is not enough space on the substrate and leaves some of the pins disconnected.

### 3.6.2 Pulsation Interpreter

The Pulsation interpreter executes recorded if–then and map–to rules in our test and debug environment as well as on microcontrollers. The implementation is consistent with .Net Micro Framework specifications to ensure its portability to microcontrollers, such as Netduino and Threadneedle. As such, the results observed in the test and debug environment of PaperPulse are always consistent with the output from the microcontroller. In contrast to the behavior of widgets inside the design tool, their physical counterparts are subject to noise which might lead to undesired oscillations. PaperPulse mitigates this problem by smoothing analog input signals. When analog signals are discretized (e.g. for pull-chain radio buttons), hysteresis, or double thresholding is used.

To get the recorded logic onto these microcontrollers, we generate code with .NET CodeDOM that re-instantiates all objects needed for the specified Pulsation logic. Once the microcontroller starts, it runs the generated code and thus initializes all logic. Afterwards, the microcontroller runs the Pulsation interpreter every CPU cycle. The Pulsation interpreter keeps track of timing information and states of widgets over different cycles to ensure that the output is always correct and independent of the speed of the microcontroller. The current version of the Pulsation interpreter requires a least 34 kilobytes of memory.

The Pulsation implementation achieves a modular design that is reusable and extensible by abstracting: (1) Widgets according to their input or output type to make the system sensor-agnostic (e.g. whether an electronic circuit sticker slider, paper-membrane slider or pull-chain slider is used, is irrelevant for Pulsation). (2) Connection pins to support different microcontroller platforms, such as Netduino and Threadneedle. (3) Actions and conditions as discussed in sections 3.5.1 and 3.5.2.

### 3.6.3 Generating Printable Pages

For every PaperPulse design, five PDF files are generated using the PDFSharp library[6]: the base layer with the main electronic circuit, the conductive elements on the widget-specific layer, the visual information for the widget-specific layer, the visual elements for the top layer, and the conductive elements on the back of the top layer. These five layers are enventually printed out on 3 sheets of paper as shown in Figure 3.4. Conductive traces are rendered using vector graphics to preserve the quality and maximize its conductivity. When content is printed on the back of a sheet, PaperPulse automatically flips it to ensure correct alignment. Regions of different layers that have to make contact to ensure electrical connectivity are enlarged to compensate for possible misalignments by the printer or user (e.g Z1 and Z2 in Figure 3.11).

## 3.7 Example Designs and Use Cases

To demonstrate the expressive power of PaperPulse, we designed several fully functional interactive paper interfaces for various use cases. Both the visual layout, electronics, as well as the logic is realized using PaperPulse.

#### Interactive Diet Card

The interactive card shown in Figure 3.20 helps tracking the number of portions one consumes of different food categories during the day. Every time a button at the top of the design is pressed, one more LED for that food category lights up. The reset button at the bottom ensures that the same card can be used multiple days.

#### Secret Invitation Card

Since PaperPulse enables end-users to create physical paper interfaces, these interfaces can be very personalized, such as the invitation card shown in Figure 3.21. Only if the year of birth of the sender of the card is known and correctly entered by the receiver, will the date of the birthday party be revealed. Every time a button is pressed, the progress for entering the four digit code slot (year of birth) is reflected by the LEDs at the top. We envision the electronic components and conductive inkjet printing to become cheap enough so that users can send these kind of interactive paper interfaces to people or hand them out.

[6] http://pdfsharp.com

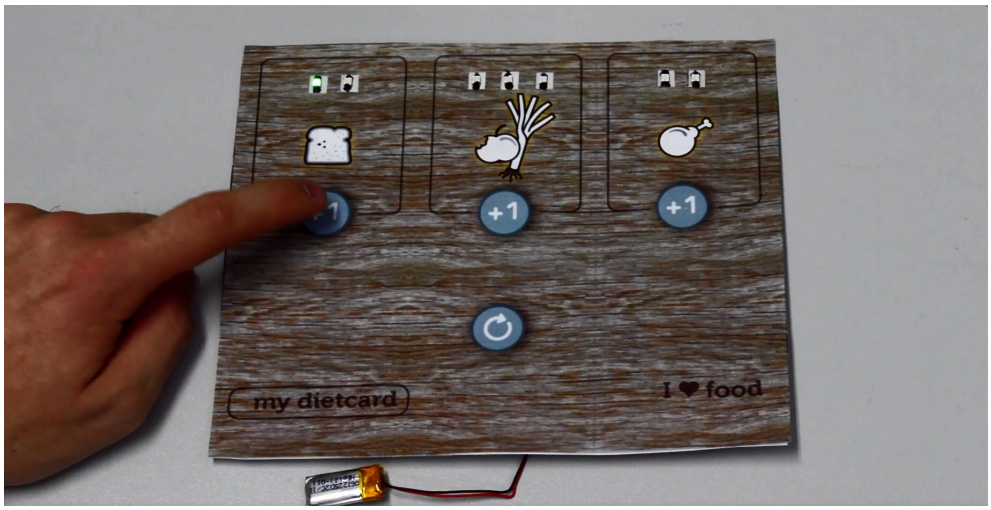


**Figure 3.20:** A diet card to track you food consumption designed with PaperPulse.

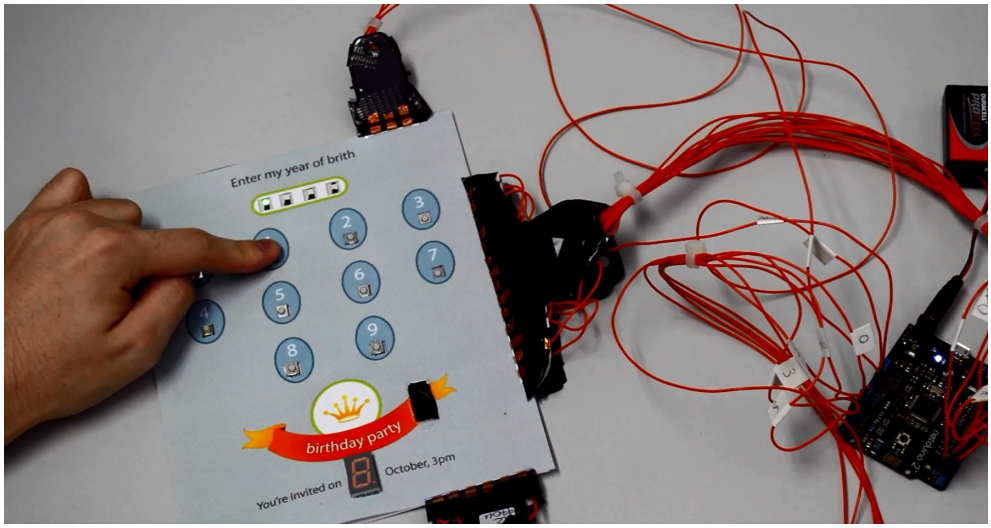


**Figure 3.21:** An invitation card protected with a code slot designed with PaperPulse.


**Figure 3.22:** An interactive restaurant menu to filter through food options designed with PaperPulse.

### Interactive Restaurant Menu

Restaurant menus are sometimes very extensive, including lots of different meals, drinks, combos, special deals and an endless list of wines. Recently, some restaurants replaced traditional paper menus by tablets. These digital alternatives provide many new opportunities, such as filtering through menu options based on the price, or preferences (vegetarian, vegan, etc.). To retain the qualities of paper menus while preserving the benefits of digital interfaces, one can design interactive paper menus using PaperPulse as shown in figure 3.22. When specifying the maximum price using the slider at the bottom of the design, the available food options are highlighted using LEDs.

### Responsive Poster

Figure 3.23 shows a poster interface to attract the attention of people in, for example, a bar. The paper design integrates a microphone which registers the intensity of the music in the environment. The sound level is then reflected to the number of LEDs that light up.


**Figure 3.23:** An interactive poster to attract attention designed with PaperPulse.

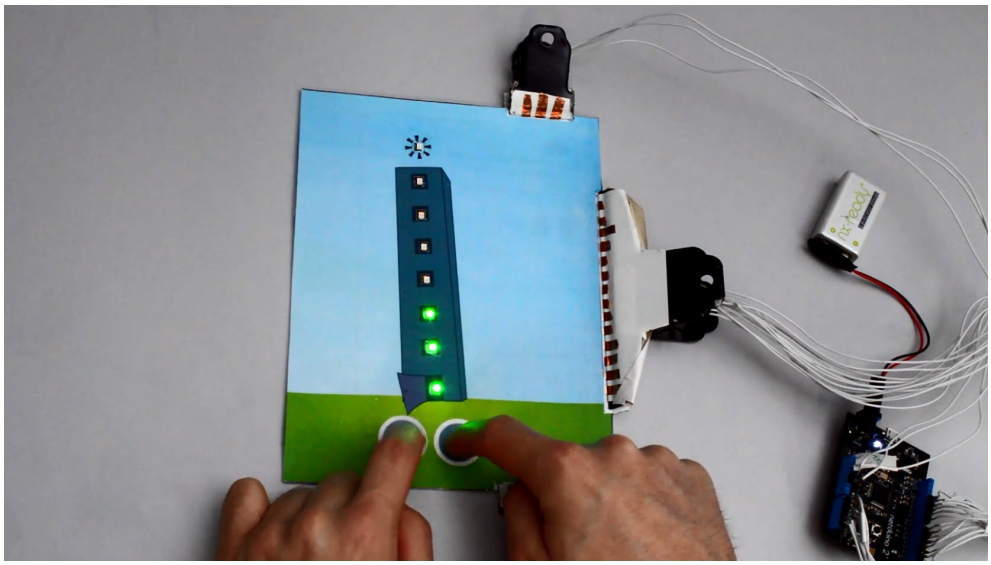

**Figure 3.24:** A children's tapping game designed with PaperPulse.

#### Children's Tapping Game

The tapping game shown in Figure 3.24 entertains children by reflecting the speed with which two buttons are pressed consecutively, in the number of LEDs that light up.

## 3.8 User Study: Making Stand-Alone Interactive Paper Artifacts

### 3.8.1 Preliminary User Evaluation

To gauge the usability and utility of PaperPulse, we conducted a preliminary first-use study with four designers: a multimedia, a graphical, and two product designers. Two participants had no prior experience in programming or electronics. The other two participants had some limited experience with Arduino and programming. Every session lasted for 2.5–3 hours. A video introduced the participants to the basic options of PaperPulse and Pulsation. Next, a video tutorial for designing and fabricating the diet card, shown in Figure 3.20, was provided. For the first task, participants were instructed to replicate this diet card using PaperPulse. For the second task, participants had to design and

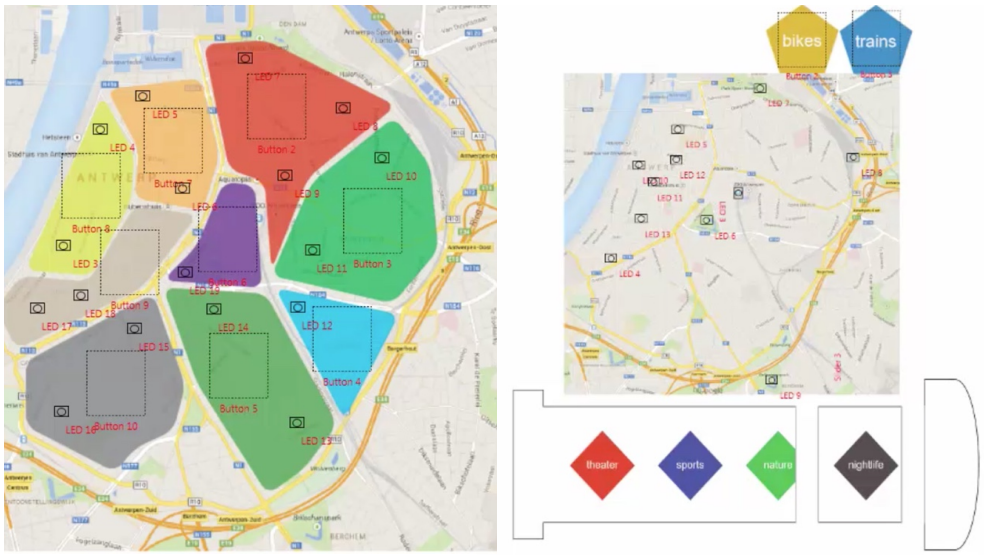


**Figure 3.25:** Designs made by a participant. (a) A voting meter for neighborhoods. (b) A tourist information map.

conceive their own ideas in PaperPulse, and reported on their experience with the system through a questionnaire and interview.

All participants were able to design and assemble the diet card in less than 45 minutes. Participants perceived the process of assembling the design enjoyable and were satisfied with the end result and reported that the outcome met their expectations. One designer said he was "pleasantly surprised and the whole fabrication process was like magic".

After finishing the diet card, all participants were enthusiastic to make their own interactive paper interfaces. Two participants had very concrete ideas: one designed an interactive placemat for restaurants, and the other designed interactive city maps as shown in Figure 3.25: one to filter through points of interest, and another to enable voting for specific neighborhoods (similar to [Vlachokyriakos 14]). The other two participants had more abstract ideas (e.g. pressing multiple buttons to make LEDs blink, and specify beeping patterns played by a buzzer) and explored these using PaperPulse. During logic specification, all participants used the simulator regularly, to check if the rules they added behaved as expected. Since rules used by participants were quite simple, errors were detected immediately. We expect users to take advantage of the 'Debug View' for more complex rules.

The two participants who had experience with the Arduino platform reported that they would be able to make the diet card using other tools, such as breadboards and copper tape. However, they noted that this would require more time and skill and the result would probably not be as visually pleasing as with PaperPulse.

According to the questionnaire and interview, participants felt that PaperPulse supports a wide variety of widgets which could even foster new design ideas. Participants also identified several areas for improvement. Firstly, one participant suggested supporting additional widgets, such as 2D touch pads and stepper motors. Secondly, participants preferred more visual instructions (e.g. images or videos) during the assembly phase. During the limited exposure to Pulsation, participants found *map–to* rules harder to understand compared to *if–then* rules. From the first usage experience, they also found it hard to identify in which scenarios a *if–then* or *map–to* rule would be appropriate. However, everyone recognized that the derived parameters supported by *map–to* rules are very useful and provide a lot of flexibility.

### 3.8.2 PaperPulse Workshop

Following the preliminary study, we conducted two half-day workshops with 9 undergraduate students (bachelor). Although all participants had previous experience with programming, non of them programmed sensor-based systems or worked with electronics before. Therefore, students were encouraged to make designs that included an extended number of logic constructs, to identify the limitations of Pulsation. In groups of 2 (one group with 3 students), participants were first introduced to PaperPulse using video tutorials, at the start of the first afternoon. Afterwards, every group replicated either the interactive diet card (see Section 3.7) or the Hungry Monkey game (see Section 3.3). All four groups successfully completed the replication task within 1–2 hours.

A few days before the start of the second half-day workshop, we gave the same groups of students the task to come up with new PaperPulse design ideas. During the second part of the workshop, they conceived these ideas using PaperPulse. Figure 3.26a shows the design of group A, an interactive paper version of the rock-paper-scissors game. Group B realized a competitive two-player version of the tapping example design discussed in Section 3.7 (see Figure 3.26b). Group C designed an interactive card for learning mathematical sums (see Figure 3.26c). The last group, group D, realized an interactive paper version of the Guitar Hero game.

Groups A and B successfully finished their design within 2–3 hours. Group

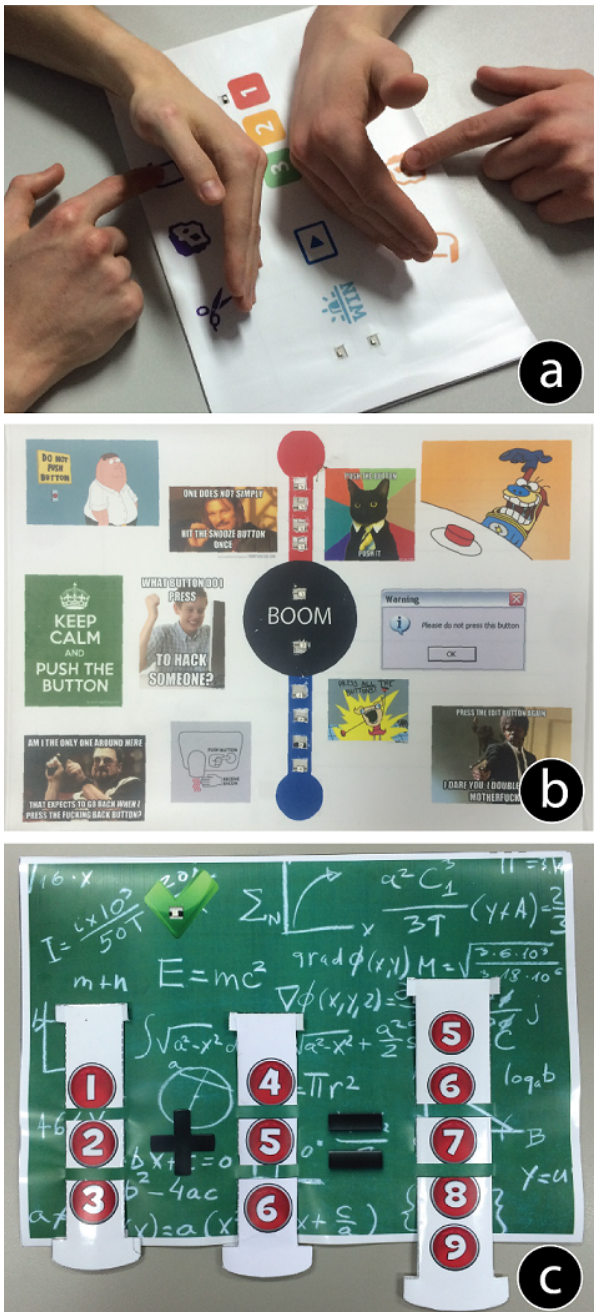


**Figure 3.26:** Three PaperPulse designs realized during the workshop: (a) interactive paper version of the Rock-Paper-Scissors game, (b) a two-player tapping game, (c) an interactive card for learning mathematical sums.

C and D finished their designs but could not immediately deploy the project because the specified behavior exceeded the memory limitations of a single assembly file on the .NET Micro Framework (max 64kb). By manually splitting the project over multiple assemblies, the project could be deployed. We noticed however that these participants specified more than 40 logic rules. After analyzing both projects, we observed that the number of logic rules could be reduced significantly, when intermediate states could be stored in variables. For example, group C used an LED to reflect the correctness of the sum (Figure 3.26c). Here the logic specifies the result of every possible combinations of the three pull-chain sliders. With 45 possible combinations, this results in 45 if–then rules. Using three intermediate variables however, one for representing the visual state of every slider, a single rule would be sufficient for checking the mathematical validity of the sum. However, custom variables are not yet supported in the current version of Pulsation. When the number of logic rules increases, we also noticed that it becomes challenging for users to relate input and output sets to electronic components on the canvas. Participants raised this concern and suggested that visually linking conditions and actions to the respective electronic components might help to get a better overview of the specified behavior.

A questionnaire and interview, revealed that participants enjoyed the experience and reported that they would not be able to realize similar designs, on paper or breadboard, without PaperPulse because of their lack of electronic and sensor-based programming knowledge.

## 3.9 Discussion

Our preliminary study and workshop demonstrates that PaperPulse empowers users with no or very little technical expertise to make physical paper interfaces. For all of our participants, PaperPulse was an enabling technology—without this tool participants would not be able to realize a fully integrated interactive paper design.

### 3.9.1 Pulsation

Pulsation supports specifying functional relationships between electronic components without using complex programming structures, such as loops, variables, and functions. Similar to the IFTTT service[7], Pulsation is developed

[7] https://ifttt.com

around the basic concepts of "If This Then That" conditional triggers. Besides, Pulsation augments conditions and actions with advanced timing parameters and mapping concepts, as discussed in Section 3.5. These advanced constructs allow users to specify complex behavior which traditionally requires using intermediate variables, loops, timers, mathematical calculations, etc. In many ways, Pulsation therefore provides a lower threshold for users to specify behavior between electronic components, as compared to traditional programming languages. However, during the evaluations (Section 3.8) and design sessions (Section 3.7), we noticed the following limitations with respect to Pulsation:

- *Out of context logic specifications*: Pulsation logic specifications are authored using the logic recorder in the upper right corner of PaperPulse (Figure 3.2a). Although this provide users with a visual representation of Pulsation logic, recorded logic behavior is visualized next to the canvas. This makes it hard to relate behavior to electronic components later. Future versions of Pulsation could provide a better overview by visualizing logic on top of electronic components using a graph visualization of the behavior.

- *Control over variables*: Pulsation only allows for specifying conditions related to elementary variables of electronic components, such as the value of a slider, state of a button, or brightness of an LED. The current implementation does not support custom variables, or access to internal variables, managed by the Pulsation interpreter (e.g. speed with which an input set is completed). Therefore every condition has to relate back to one of the elementary variables in the design. Oftentimes, this significantly increases the complexity as well as the total number of conditions.

- *Limited expressive power*: Although participants could all express their design ideas using Pulsation, we noticed several options that were missing while making example designs (Section 3.7). First of all, Pulsation strictly distinguishes if–then from map–to rules. Sometimes, conditional map–to rules (i.e. if–then–map–to) are also desired. Additionally, the user-evaluations revealed that participants found it hard to identify when to use if–then or map–to rules. After several discussions with participants to articulate the complexity they experienced, we realized that map–to constructs are in essence advanced actions. To facilitate users understanding and increase expressive power, future versions of Pulsation could support only conditional statements, and offer map–to constructs as advanced actions. Secondly, we noticed that derived parameters (e.g.

time, speed, progress, etc.), available in map–to constructs, could also be desired for conditional statements. For example, verifying whether all conditions of an input set remain fulfilled for a predefined duration of time. Finally, the current version of Pulsation sometimes lacks precise control over parameters. Examples include: (1) Excluding events that cannot take place while matching input sets. Although the *include* and *exact* matching approaches target these kind of expressions, they only specify whether any other event, besides the event in the input set, is allowed. As such, entering for example, two separate input sequences on a single design in parallel (e.g. two separate code slots) is not possible. Events required for one input sequence would cause the other input sequence to reset and visa versa. (2) Additional control over input sets that have to be completed *sequentially*. The sequential timing parameter is the complex as different conditions have to be completed over time. For example, it is often unclear whether conditions need to remain fulfilled until all other conditions in the input set are completed. Additionally, in some scenarios, it might be allowed to match conditions multiple times or even in a different order, as long as a sequence of events is registered that also match the ordered sequence of conditions. (3) Variable parameters instead of constants. Some of timing and derived parameters, including, the loop and delay construct in output sets, only support constants. Future versions of Pulsation can provide a higher ceiling by supporting variables for these constructs.

Based on these limitations and lessons learned, Chapter 5 presents a novel version of Pulsation.

### 3.9.2 Electronic Circuit Design

Although manufacturing techniques for realizing flexible PCBs with multiple layers exist for a while, PaperPulse's circuit generation technique produces circuits of arbitrary complexity consisting of only one layer. Bridges are placed at crucial location to realize complex non-planar circuit graphs. In contrast to multi-layered PCB manufacturing techniques, single-layered substrates can be manufactured using low-cost technology, such as the off-the-shelf conductive inkjet printer used for our example designs. These low-cost technologies allow for fast on-site design iterations which is essential in many design processes. PaperPulse is the first tool that automatically generates circuits of arbitrary complexity on single-layered substrates. Hence, PaperPulse also speeds up and

lowers the effort for engineers to make these kind of complex single-layered circuits.

The electronic circuit traces generated with PaperPulse are optimized for printing on resin coated substrates using a conductive inkjet printer. However, designs could also be produced using other techniques, such as milling coper traces [Savage 12], chemical etching, or silkscreen methods utilizing traditional inkjet printers [Varun 15].

Future versions of PaperPulse can also optimize usage of electronic components. In the current implementation, every widget needs to be exclusively connected to one GPIO pin on the microcontroller. Supporting multiplexing strategies or sharing pins among output widgets that are in the same state at all times, could reduce the number of pins needed. Another interesting track of future research is to eliminate integrating general-purpose microcontrollers, which for many designs, is the largest non-flexible and most expensive component. One interesting alternative technique is to build the entire circuits using electronic primitives, such as the 7400 series integrated circuits [Barr 99]).

### 3.9.3 Widget Toolkit

We distilled the paper-membrane and pull-chain widget designs to their bare minimum to ensure customizability and reusability. However, we envision more custom designs in the future, such as sliders with non-straight tracks, circular shapes for dial mechanisms (often called wheels or volvelles in paper craft [Carter 99]), or origami constructions for non-flat designs [Olberding 15]. The paper-membrane and pull-chain widgets mainly focus on standard controls, such as push buttons, switches, sliders and radio buttons since these components benefit much from customization. Although the visual design and dimensions of paper-membrane and pull-chain widgets can be customized, their overall shape (e.g. shape of handle) is fixed. We envision a widget editor in the future. Future versions can also integrate paper versions of other input components, such as bend and pressure sensors or output components, such as speakers [Saul 10, Shorter 14] and microphones.

## 3.10 Summary

In this chapter we presented PaperPulse, a design and fabrication approach that enables users to make fully-integrated physical interfaces using novel circuit fabrication techniques, such as conductive inkjet printing (G1). With PaperPulse users are guided through the visual and electrical design aspects,

as well as the programming of physical paper interfaces. Making similar physical interfaces traditionally requires knowledge in various fields and expertise in different tools and systems (C5). To assist users in visually designing a physical paper interface, we contributed a unified three layering approach and three families of interactive widgets that integrate well in paper designs (C2). Additionally, non-programmers specify logic in PaperPulse using the integrated Pulsation logic specification paradigm (C4). Using locations of the widgets on the canvas and the Pulsation logic specifications, our system generates an electronic circuit design that can be produced on flexible substrates using low-cost easy-to-use machinery (C1). A custom-generated tutorial instructs users to attach components to the circuit design (C3). The wide variety of example designs included in this chapter, shows our approach is viable and results in many different physical user interfaces. A preliminary evaluation with laypeople shows the utility and usability of our system and workflow. Participants were pleased with the resulting paper artifact and the workflow of creating interactive paper interfaces themselves, something that was unavailable for them before.

# Chapter 4

## RetroFab: Adapting Existing Physical Interfaces

### 4.1 Introduction

Graphical user interfaces of popular computing devices, such as desktops and smartphones are easy to adapt and interconnect to accommodate for changing user needs. In contrast, devices and appliances such as ovens, thermostats and toasters, are traditionally designed to be static and non-adaptive. The tangibility and rigidity of these legacy devices make it hard for an end user to change the user interface, as one may do with software applications through plug-ins, reverse engineering [Chikofsky 90], or runtime toolkit overloading [Greenberg 01]. For instance, it is not feasible to resolve design mistakes or adapt an interface to users' evolving or custom needs (e.g., impaired users).

To make changes to legacy infrastructures and allow for interconnectivity, one can retrofit the physical user interfaces, for example, to augment light switches[1] and dials[2]. When retrofitting, a redesigned physical component is placed over top of the original component, thus serving as a proxy interface. Mechanical actuators are often used to manipulate the original interface, while sensors detect states of the device (e.g., via LED indicators). Interactions with controls placed over top of the structure are forwarded to the original interface using the sensors and actuators (Figure 4.1). This approach avoids the complications and risks of fully disassembling and rewiring existing electronic

[1] www.switchmate.com
[2] www.locitron.com

**Figure 4.1:** The Switchmate retrofit kit snaps over top of traditional light switches. User's input with the toggle buttons on top are redirected to the legacy light switches using integrated actuators.

components, and is akin to customizing a software user interface without accessing or modifying its source code [Dixon 10].

Although these appliance specific retrofitting kits[12] are easy to install, the redesigned interface that is now exposed is static and cannot be reconfigured by novices to adapt to personal or changing user needs. Davidoff et al. [Davidoff 11] experimented with customizable retrofit interfaces using the LEGO Mindstorms toolkit. However, the mechanisms had to be manually designed and constructed, requiring users to assemble precise structures and brackets that fit over top of the appliances. To enable users to customize and enhance existing physical interfaces (G2), we present RetroFab, a design tool that automates the process of retrofitting a physical interface (Figure 4.3). Using novel Do-It-Yourself fabrication techniques, such as 3D printing (C1), the RetroFab design environment streamlines the process of making retrofit interfaces by automating 3D design modeling, electro-mechanical constructions, as well as programming aspects as follows:

- Designing 3D structures that fit over top of existing physical interfaces requires modeling the existing structure by measuring its dimensions. Physical interfaces, such as appliances often have shapes with complex curvatures and cavities. Designing structure that fit precisely on over

top and connects firmly to the existing infrastructure is challenging even for experienced 3D modeling experts. Furthermore, the retrofit structure holds together actuators and sensors. These electrical-mechanical components need to be positioned precisely for the retrofit interface to work. To allow non-experts to produce retrofit designs, RetroFab automates the entire 3D modeling process starting from an annotated 3D scan of the existing interface (C2).

- Retrofit interfaces integrate electrical sensing components, such as light sensors, as well as electrical-mechanical components, such as DC, servo, and stepper motors. Controlling these components requires advanced technical skills. Although construction kits like LEGO Mindstorms make it easier to connect and control these components, these mechatronic toolkits are not optimized for operating controls in physical interfaces. RetroFab therefore comes with a mechatronic toolkit of component optimized for actuating common controls used in household physical interfaces and appliances. Custom to the generated retrofit interface, our software assist the user in connecting these toolkit components (C3).

- To enable non-programmers to change the behavior of existing physical interfaces, RetroFab allows users to specify the behavior of retrofit interfaces. RetroFab seamlessly integrates the Pulsation visual programming paradigm presented in Section 3.5. With Pulsation, behavior is specified by visually creating functional links between electronic components and online services. Although, we show how Pulsation integrates in RetroFab and demonstrate some example behaviors, the full set of Pulsation features are discussed in Section 3.5.

In the following, we first give an overview of the system. Next, we provide a walkthrough of the system by showing step-by-step how fabricate a retrofit interface for a toaster. We then present and discuss RetroFab's mechatronic widget toolkit and the different retrofit models that are supported in our software. Afterwards, we provide details on the architecture and implementation of the system. We also present various retrofit interfaces designed with RetroFab to show the validity and possible use cases of our approach and report on a user-study which demonstrates the utility and usability of RetroFab. We end with a discussion and summary of the presented work.

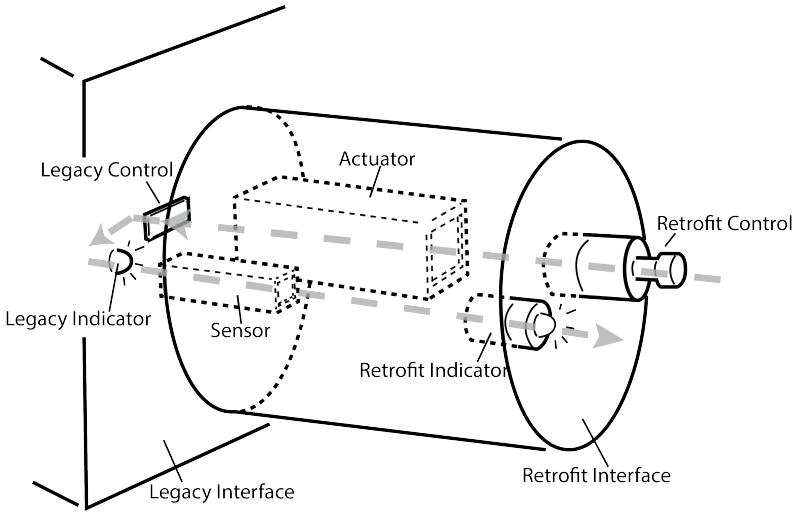


**Figure 4.2:** Overview of the retrofitting process. Sensors and actuators are placed on the legacy interface, with new controls and indicators placed on a new, retrofit interface.

## 4.2 Brief System Overview

The key idea behind this work is to refactor physical interfaces by mounting a redesigned proxy interface over top of the existing form factor that is able to intercept user input and redirect it to the original object using mechanical actuators, while also intercepting device output and redirecting it to the user (Figure 4.2). We define the legacy interface as the target object which the user wishes to modify. The legacy interface consists of one or more components: legacy controls for input (e.g., buttons) and legacy indicators for output (e.g., LEDs).

The RetroFab design and fabrication environment facilitates and partially automates the process of making retrofit interfaces. Hence Retrofab allows laypeople to make retrofit interfaces as follows: From an annotated 3D scan of an existing legacy object (Figure 4.3a-b), RetroFab automatically generates circuitry, firmware and a physical enclosure that precisely fits over top of the legacy interface (Figure 4.3c). These enclosures house mechanical actuators and sensing components to automatically control the device and observe its state (e.g. sensing the state of an LED). The system allows the user to to redesign the new retrofit interface using input and output components (Fig-

ure 4.3d). A layer of actuators and sensors are used to interface between the retrofit and legacy interfaces.

Besides redesigning the physical interface of legacy devices, RetroFab can also change the behavior of devices. By default, retrofit components on the front panel of the attached enclosure structure mirror all actions to RetroFab actuators behind them: controlling a RetroFab push button or dial on the front panel causes similar actions on the original controls using the RetroFab actuators behind these controls. Similarly, output is redirected from the legacy indicators to the enclosure structure using RetroFab sensors: a light sensor, observing the state of an LED on the legacy device redirects this state to an LED on the front panel.

The user can alter this default behavior, add extra logic, or define logic of additional RetroFab components that were added using the Pulsation logic specification paradigm (Chapter 5). In contrast to PaperPulse (Chapter 3), where the Pulsation interpreter runs independently on every microcontroller, here the interpreter is modified to run on a central logic module (i.e. Windows PC or microcontroller supporting .NET MF), making intercommunication an inherent part of RetroFab.

## 4.3 Walkthrough: Refactoring a Toaster

The following walkthrough illustrates the process of retrofitting a legacy interface using RetroFab (Figure 4.3). We use the example of a toaster, and in later sections describe additional functionality and use cases. For the toaster, the legacy interface is composed of the various buttons (cancel, bagel, defrost, and reheat), a dial for temperature control, a lever to push the toast up and down, and a set of indication LEDs.

The following example shows how RetroFab can be used to intercept all interactions and add one extra button on the toaster as a shortcut to a preferred setting - a defrost cycle to thaw the bread, then a mild toasting.

### Step 1: 3D Scanning and Annotating Controls

The user starts by 3D scanning the toaster using the Skanect 3D scanning software and a Microsoft Kinect. Before the scan, the buttons and LEDs are highlighted by covering them with tape to ensure their visibility even in low quality 3D scans by novices (Figure 4.3a). The user loads the 3D model in the RetroFab design tool and annotates the position of the legacy controls and

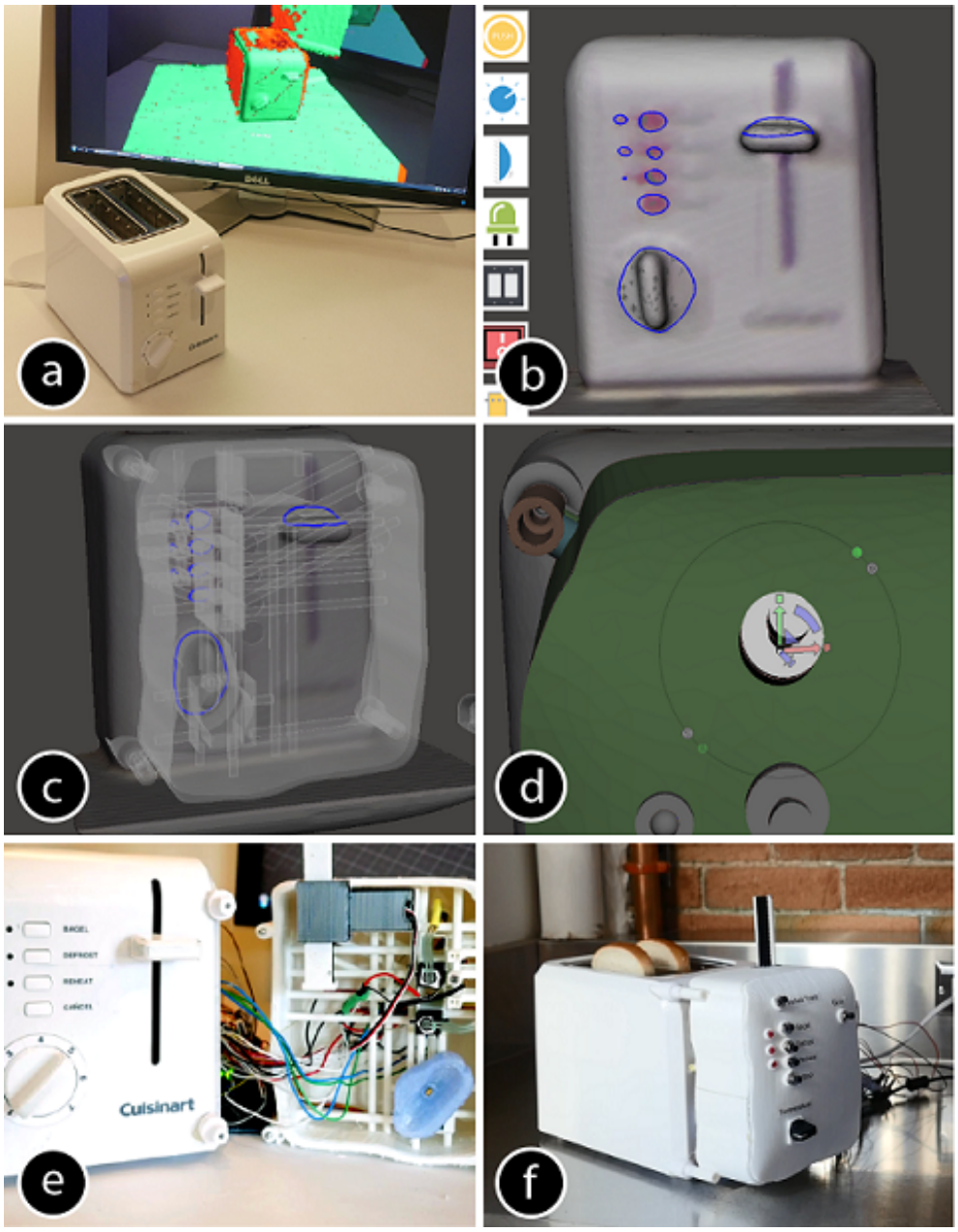


**Figure 4.3:** Retrofitting a legacy toaster with RetroFab. (a) The toaster is scanned, (b) the legacy interface is annotated, (c) the attached enclosure is generated, (d) the physical interface and behavior of the retrofit interface is adapted, (e) the enclosure is fabricated and assembled, (f) the new retrofit toaster is perfectly toasting

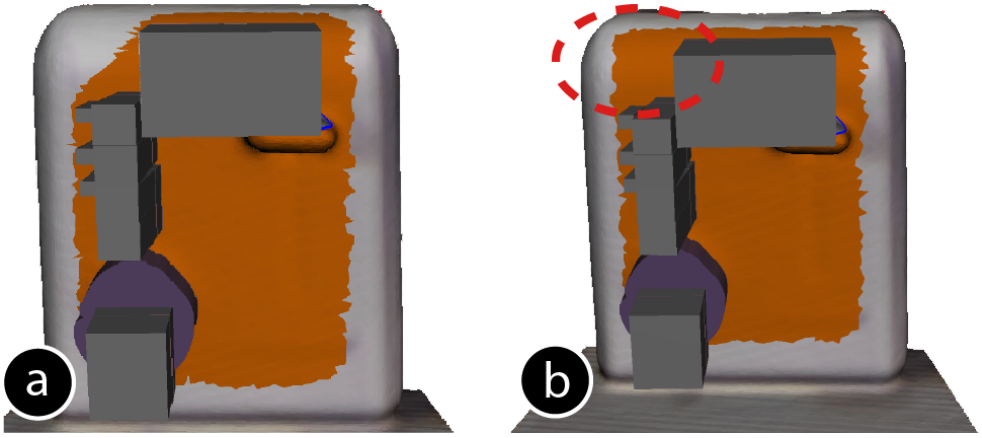


**Figure 4.4:** (a) The orange region depicts the enclosure region that RetroFab derives based on the specified components. (b) The user manually extends the region to have additional space for the new shortcut button.

indicators using the brushes in the toolbar on the left. The system contains one brush for every type of supported legacy control and indicator (Figure 4.3b).

### Step 2: Automated Enclosure Design

Once the annotations are finished, RetroFab positions the housings for all actuators and sensors: linear actuators for the pushbuttons and lever, stepper motors for dials and light sensors for LEDs (Figure 4.4a). RetroFab highlights the region that will be redesigned, and thus covered by the new physical enclosure. The user extends this region to include the area where the new shortcut button will be positioned (Figure 4.4b).

RetroFab responds by generating the 3D model of the enclosure (Figure 4.3c). Finally, the user specifies the preferred location of the mounting brackets (using a brush) that attaches the enclosure structure to the toaster (Figure 4.5).

### Step 3: Redesigning the Interface and Behavior

RetroFab automatically integrates a retrofit interface in the front panel of the enclosure structure that serves as a proxy for the legacy interface. If desired, the user can redesign this front panel by repositioning the retrofit components or by replacing them with alternative components available in the toolbar.

**Figure 4.5:** The mounting brackets allow the enclosure to easily be added to or removed from a legacy object.

Figure 4.3d shows an additional pushbutton being added to the toaster that will serve as a shortcut to the user's favorite toast setting.

By default, retrofit controls are configured to mirror the actions of their associated legacy controls. The user can alter this default behavior or add extra logic for new components using the Pulsation visual programming paradigm (Chapter 5). Users demonstrate actions directly on top of the 3D models of the respective retrofit components and can also record functional relationships between these actions. Figure 4.6 shows the user specifying the logic of the new button on the toaster: If pressed, the lever goes down, and the defrost button is pushed. Additionally, the user can specify that when defrosting is finished (sensed by the LED next to the defrost button), the toast goes back in for a second time at a higher temperature, resulting in perfectly crisp toast!

### Step 4: Fabrication and Assembly

When the design is complete, RetroFab generates: (1) STL files to be printed out using a 3D printer (FDM printer for our prototypes), (2) Microcontroller code that can be directly uploaded to an Arduino platform, and (3) Assembly instructions to guide users in connecting the actuators and sensors to the microcontroller.

Figure 4.3e shows the assembled 3D printed enclosure and its embedded actuators, sensors, and retrofit components. Finally, the enclosure structure

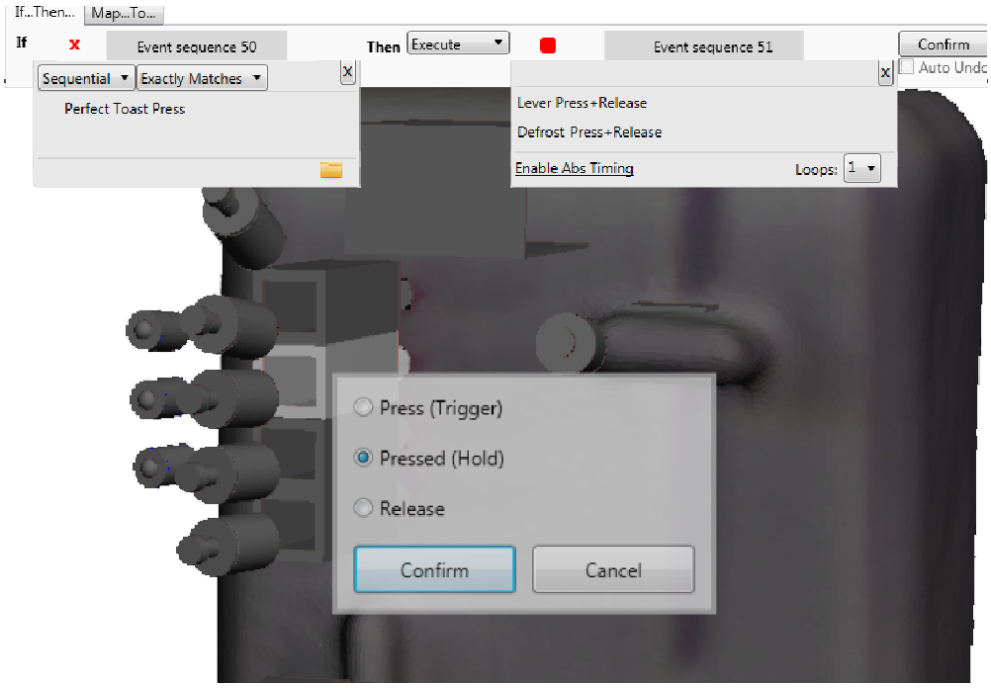


**Figure 4.6:** The user manually specifies a logic rule using the Pulsation engine to program the perfect toast setting.

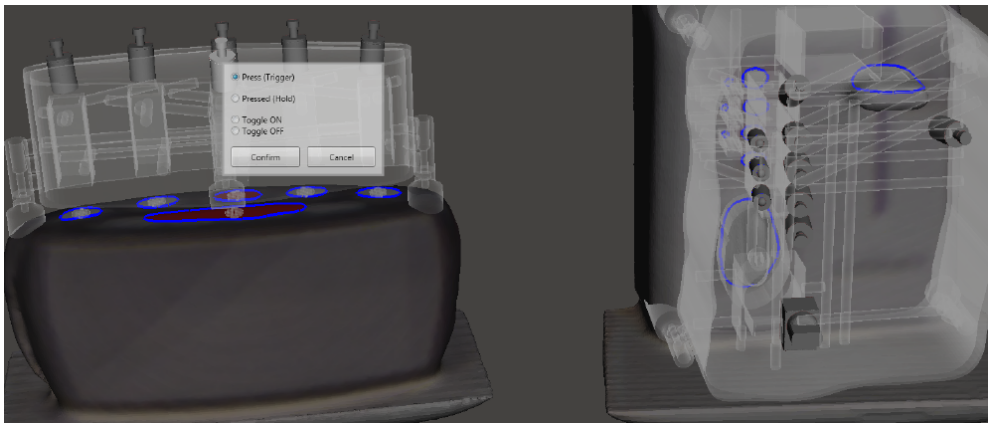


**Figure 4.7:** As all of a user's retrofit objects share the same workspace, they can be interconnected to link their functionality and enable home automation tasks.

is attached to the toaster by gluing the feet of the mounting brackets to the appliance (Figure 4.3f). The mounting bracket is designed so that the enclosure structure is easy to detach using screws, leaving only the feet of the mounting brackets behind (Figure 4.3e).

Once the enclosure structure is attached, RetroFab guides the user through a process to calibrate all the actuators and sensors, such as the range of movement for an actuator to push a button or turn a dial, or the brightness registered by a light sensor to detect the state of an LED. Once the user connects the (temporary) calibration button to the microcontroller of the retrofit interface, the states of each actuator/sensor are calibrated one by one. During this procedure, actuators/sensors are activated and the user presses the calibration button when the actuator/sensor is in the requested state e.g. on/off state for light sensors observing indicator LEDs, min/max state or discrete states for rotary dial actuators. The RetroFab UI supports removing calibration samples and averaging multiple samples.

### Step 5: Deployment

All logic defined using RetroFab runs on a central PC which communicates continuously to the retrofit interface. This makes it possible to reconfigure the behavior of devices at run time and allow for interconnectivity. Figure 4.7 shows the user linking the "snooze" button of his retrofitted alarm clock to his personal "perfect toast" shortcut button on the toaster.

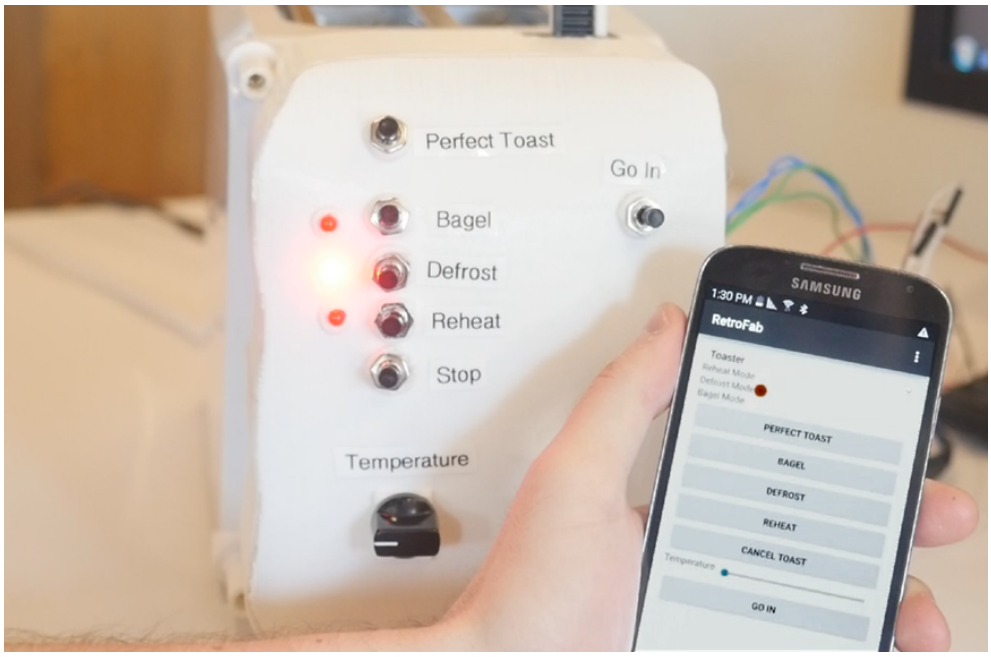


**Figure 4.8:** All of the functionality of a retrofit interface is also available in the companion Android application.

When the user launches the companion RetroFab mobile application, all retrofitted devices are automatically loaded, and display their functionalities to allow for remote control (Figure 4.8).

## 4.4 RetroFab Widget Toolkit

To enable retrofitting a wide variety of devices, RetroFab comes with a set of electrical and mechanical primitives to retrofit common physical interface components, such as pushbuttons, rotary dials, rocker and wall switches, and LEDs.

Figure 4.9 shows the full RetroFab toolkit, consisting of (a) actuators, (b) sensors, (c) controls, and (d) indicators. The actuators and sensors are positioned inside the enclosure and concealed, while the controls and indicators are positioned on the outside of the surface, forming the retrofit interface.

For every component in the toolkit, the RetroFab design tool has a specific component housing that is integrated in the enclosure structure to facilitate

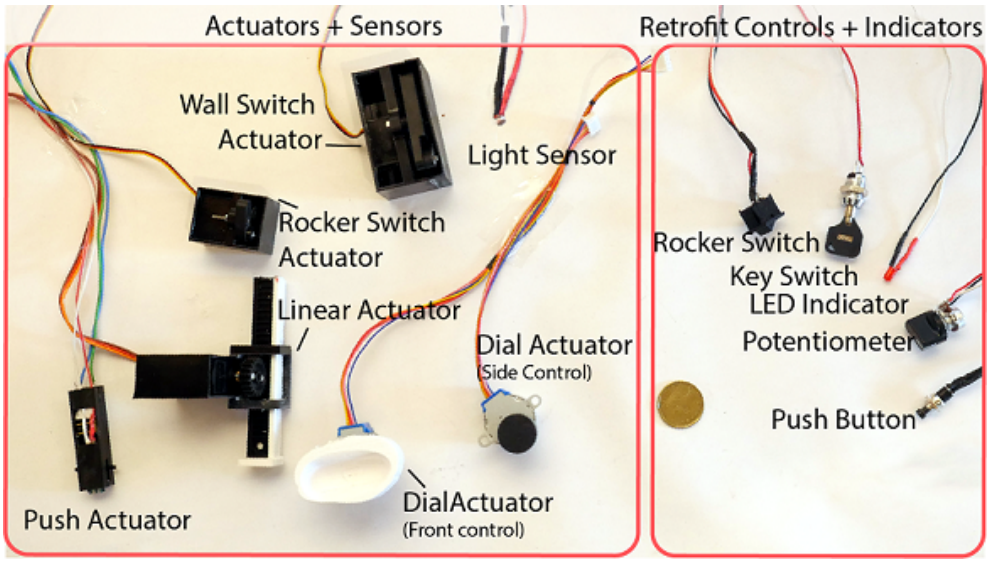


**Figure 4.9:** The RetroFab Toolkit; Left) actuators and sensors; the white material of the dial and linear actuator is the parametric component of the design which conforms to the scanned control. Right) controls and indicators.

assembly and ensure precise positioning. This is particularly important for the actuators, which operate the underlying legacy controls.

To make it possible for users without electronics knowledge to use the RetroFab toolkit, wires have a color coding scheme and integrate the necessary electronic components, such as resistors, in them. Our toolkit is easiest to deploy using the Adafruit Motor Shield, which avoids complex H-bridge electronic constructions. As such, components are connected directly to the microcontroller by following instructions provided in the RetroFab design tool, avoiding the need for complex electronic wiring designs on breadboards.

Our custom designed mechanical actuators use off-the-shelf DC, servo, and stepper motors, in combination with 3D printed transmission mechanisms, to achieve the desired mechanical movement. Figure 4.10a shows how the pushbutton actuator uses a threaded rod to convert rotational movement of a geared DC motor to linear movement. A pressure sensor is attached to the tip of the piston to reverse the motor when the pressure on the push button reaches the calibrated value. In contrast, the rocker switch and wall switch actuators use micro servos and eccentric crank mechanisms (Figure 4.10b-c).

The components that make up legacy interfaces come in different shapes

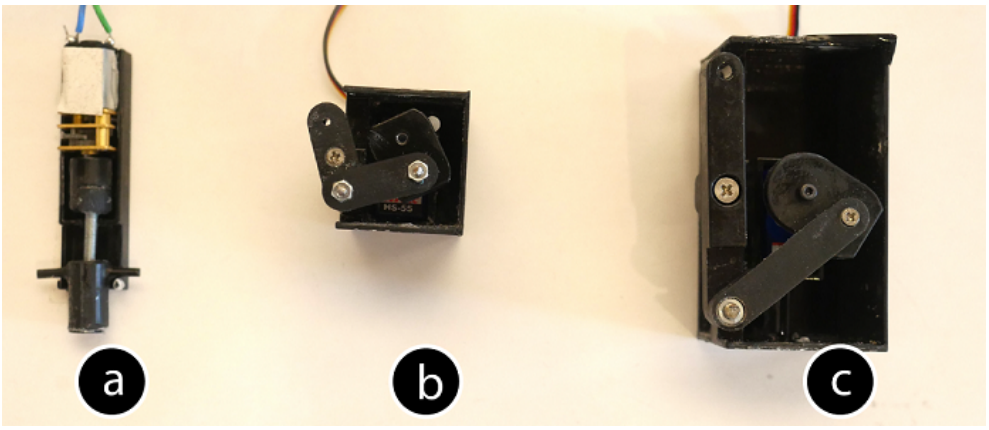


**Figure 4.10:** Inside view of a) push actuator, b) wall switch actuator and c) rocker switch actuator.

and sizes and are often closely packed together on control panels. Our actuators are therefore designed to be as small as possible while still having sufficient force. The width of our pushbutton actuator is 12mm (Figure 4.9), making it even possible to actuate legacy controls that have that same distance between their center points. Additionally, actuators are represented in RetroFab by parametric 3D models, to allow them to scale to different sizes and shapes (white material in Figure 4.9). RetroFab adjusts the size of the rack used in the linear actuator, for example, using information of the 3D scanned model. Similarly, the parameters of the rotary dial actuator allow it to take on the exact inverse shape of the knob of the dial in the 3D scanned model.

A calibration procedure is devised to measure properties of controls that are not visible in the 3D scanned model, such as the range of movement for an actuator to push a button or turn a dial, or the brightness registered by a light sensor to detect the state of an LED.

## 4.5 Enclosure Structure Design

RetroFab supports the design of two types of enclosures: attached enclosures, which attach directly to the legacy interface and remote enclosures, which can optionally house the retrofit interface separately from the attached enclosure. Below the design considerations for these two types of enclosures are outlined. Once designed, the method for specifying their behaviors are equivalent.

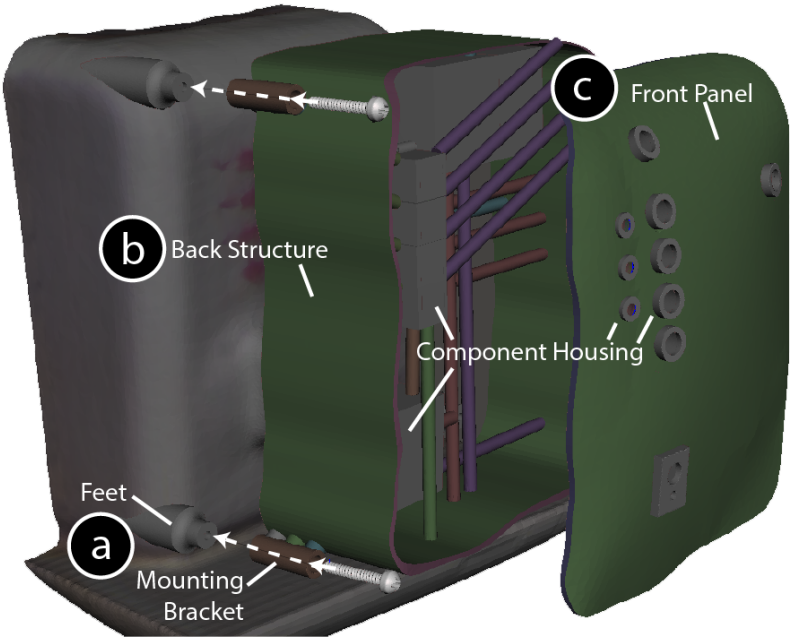


**Figure 4.11:** Exploded view of a RetroFab attached enclosure. The mounting bracket (a) is bolted to the mounting feet, which are glued to the legacy device. The back structure (b) holds the motors and sensors which interact with the new retrofit interface on the the front panel (c).

### 4.5.1 Attached Enclosures

Attached enclosures are computationally designed with RetroFab and always consist of three layers that are printed separately (Figure 4.11): (a) the feet of the mounting brackets, (b) the back structure, and (c) the front panel.

The front panel of the enclosure structure consists of component housings for attaching the retrofit controls and indicators that define the new retrofit interface that is exposed to end-users (Figure 4.11). The back structure holds housings for components in place inside the structure to precisely position the RetroFab actuators and sensors. Figure 4.11 shows how rigid support structures connect component housings inside the enclosure design to the outside structure. Last, the mounting brackets fit the curvature of the legacy interface to ensure a sturdy connection. Figure 4.12 shows how enclosure structures designed with RetroFab fit on devices with different surface curvatures.

This layered approach facilitates the assembly of retrofit components on the

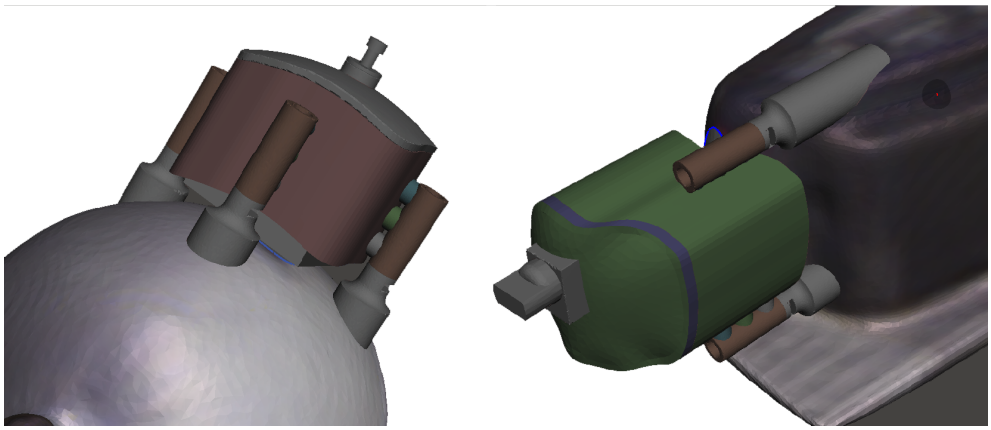

**Figure 4.12:** left) Mounting feet conform to the curved surface of a desk lamp, right) the retrofit dial allows for greater positioning accuracy when tuning the frequency of an alarm clock radio.

front panel and back structure which are later glued together. In contrast, the back structure is mounted on top of the feet of the mounting brackets using screws. This is enabled by the T-slot design inside the mounting brackets (Figure 4.11). As a result, only the feet of the mounting brackets need to be glued to the legacy device. Afterwards, the enclosure structure can be easily removed using the screws, leaving only the feet of the mounting brackets behind (Figure 4.3e).

While adding the mounting brackets to the design, RetroFab leaves a gap (approximately 1 cm) between the legacy interface and the enclosure structure. This gap makes it easier to mount the enclosure structure and allows the end-user to observe the state of legacy components while calibrating the actuators and sensors.

### 4.5.2 Remote Enclosures

RetroFab also allows users to optionally construct a remote enclosure that is not mounted over top of the legacy interface. These types of enclosures only consist of retrofit controls and indicators and communicate wirelessly via a central PC to actuators and sensors that are inside one or multiple attached enclosures. Remote enclosures can be used to design remote controls, reposition an interface to a more convenient location, or introduce new controls to an environment.

To design a remote enclosure, the user loads in any hollow 3D model and adds RetroFab controls and indicators to the front panel. RetroFab responds by integrating component housings in the 3D model that will hold the retrofit controls and indicators in place. Once the design of the remote enclosure structure is finished, the user specifies the behavior between the new retrofit interface and the actuators and sensors in the associated attached enclosures.

## 4.6 Architecture and Implementation

The RetroFab design tool is implemented using .NET/C# and builds on the Meshmixer 3D modeling program [Schmidt 10]. The companion mobile application was developed in Java for the Android platform.

### 4.6.1 Computationally Generated Enclosure Designs

To attach enclosure structures, the automated design process starts with a 3D scanned model that has user annotated regions, specifying the type and position of legacy controls. RetroFab loads and positions component housings related to the annotated controls on top of the scanned model. When components are closely packed together, RetroFab mitigates overlaps between these housings by optimizing their orientation. During this process, the system rotates the intersecting housings one by one, around the normal vector of the annotated region, until all overlaps are resolved. When no solution is found or the process is interrupted by the user, the housings can be manually repositioned

Once the housings of all components inside the enclosure structure are correctly positioned (Figure 4.13a), RetroFab generates the enclosure design. To support legacy interfaces with different surface curvatures (Figure 4.12), an enclosure structure is created by extruding the surface region of the 3D scanned model, thus preserving its curvature. The average orientation of the RetroFab actuators and sensors defines the direction of extrusion. Defining the minimal surface region for the extrusion involves the following steps. First, the bounding box of the housing for every component is projected onto the surface along the extrusion direction (Figure 4.13b). Second, the surface curvature between each component is sampled, resulting in another set of vertices. Together with the vertices calculated in the first step, a mesh of the convex hull is calculated using OpenSCAD . All faces inside this convex hull define the minimal surface region to be extruded to enclose the housings for all RetroFab actuators and sensors (Figure 4.13).

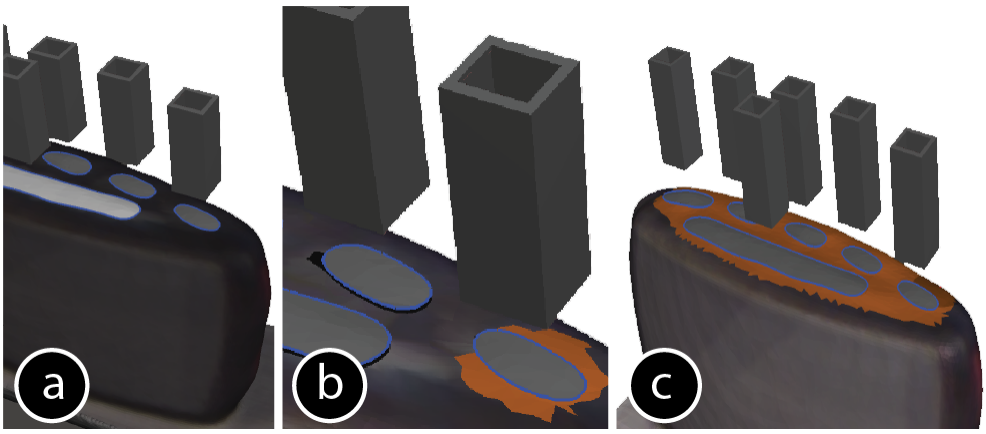


**Figure 4.13:** Computing the minimal surface region for the extrusion of an enclosure model. (a) The component housings, (b) the projected surface region underneath components, (c) the final minimum surface region.

When components are located on different sides of the legacy interface, the minimal surface region required for the extrusion can increase substantially (Figure 4.14a). In these situations, the user can decide to have a separate attached enclosure structure for some components (Figure 4.14b).

Once the final region for extrusion is defined, it is smoothed and enlarged with 3 mm to account for the thickness of the walls. This surface region is then extracted to a new mesh which serves as the front panel of the enclosure structure (thickness of the front panel; Figure 4.11). In another copy of this mesh, only the faces on the outermost 3 mm are preserved, resulting in a ring-like shape that will serve as the side panel of the enclosure structure after the faces are extruded (Figure 4.11). The wall thickness (3mm) was determined through iterative testing. Combined with the support structure that hold actuators in place, these walls provide enough stability during actuation. Thicker walls could cover undesired regions of the legacy interface or make the enclosure heavy and prone to tipping.

To connect the component housings to the enclosure structure, RetroFab casts rays from predefined support locations on the component housings towards the enclosure structure. When a valid intersection is found, a cylinder shaped support structure is created (Figure 4.11).

After the user specifies the locations of the mounting brackets, that connect the retrofit interface to the legacy interface, a predesigned mounting bracket

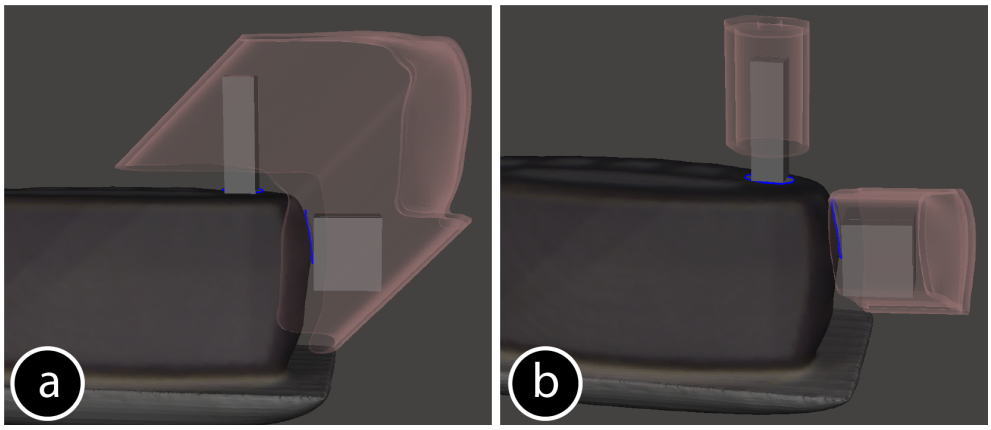


**Figure 4.14:** Depending on the surface region covered by the enclosure structure, users can decide to (a) combine actuators in a single enclosure, or (b) group some of them in a separate enclosure structure.

is put in place. To ensure that the feet of the mounting brackets matches the surface curvatures, the Boolean difference is taken between the faces of the mounting bracket and the 3D model, resulting in the removal of all faces that are in inside the 3D model (Figure 4.15).

In contrast to the housing the components inside the enclosure structure that require a support structure to hold them in place, housings of the components in the front panel are supported by the front panel itself. A Boolean difference operation between the front panel and all the components creates the necessary holes in the front panel (Figure 4.11).

### 4.6.2 Parametric Component Designs

For components that consist of parametric parts (white material parts in Figure 4.9), additional steps are required. RetroFab uses a plane cut to trim the track of the linear actuator to a length that is manually specified by the user (range of movement). For rotatory dial actuators, additional extrusions and plane cuts are applied to create an adaptor that has the inverse shape of the knob of the legacy dial. This approach can also handle dials with an off-centered knob successfully.

**Figure 4.15:** The mounting feet fit the curvature of the legacy object by using a Boolean difference operation with the scanned model.

### 4.6.3 Communication with Microcontroller

The individual Arduino microcontrollers that control the enclosure structures run a generic firmware that handles the GPIO pins as well as the wireless communication. Even for retrofitted devices that do not intercommunicate, user input and sensor data from the retrofitted interface is first transmitted from the Arduino microcontroller to the central PC. This module then decides to turn on specific RetroFab actuators and sensors, controlled by the same or a different Arduino microcontroller. This approach makes it possible to change the behavior and interconnect retrofitted devices even after the design and fabrication is completed. Multiple independent logic modules can be deployed to avoid single points of failure.

The automatic instantiation of the generic Arduino firmware on the microcontrollers requires an automatic assignment of control pins. The control pins of RetroFab components can often be connected to multiple pins on an Arduino. If binary output suffices, a digital pin can sometimes be used in place of an analog pin. The system takes this into account and first uses the specified behavior to assign a set of valid control pins to every component. Next, the algorithm selects those pins that maximize the number of components that can be connected given the limited set of pins on the Arduino microcontroller.

Once pin assignments are finished, the central PC communicates the type of components that are used and the pins they connect to the microcontroller. The microcontroller then responds by instantiating code for controlling these components. Afterwards, updates on components' states are communicated to the central PC over XBee or using a wired serial connection.

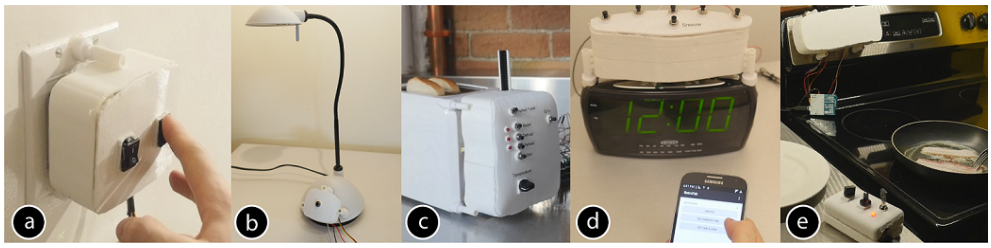


**Figure 4.16:** Example retrofit interfaces created using RetroFab: (a) two wall switches, (b) a desk lamp, (c) a toaster, (d) an alarm clock with companion Android application, (e) an oven with remote enclosure.

### 4.6.4 Communication with the Mobile Application

The mobile application communicates to the central PC using Wi-Fi. Once connected, all retrofit controls and indicators present are transmitted to the mobile device. The companion RetroFab mobile application then automatically instantiates the necessary GUI elements for controlling those components and compiles everything into a single user interface.

## 4.7 Example Designs and Use Cases

Using the RetroFab design tool, 5 legacy interfaces were retrofitted (Figure 4.16): (a) a wall switch, exposing a rocker switch on the retrofitted interface, (b) A lamp, converting a legacy rocker switch into a push button, (c) the toaster discussed in the walkthrough (Section 4.3), (d) an alarm clock with buttons for setting the time (i.e., hours + minutes), setting alarms, and a snooze button, and (e) a stove with a retrofit remote control containing 2 dials and an indicator LED notifying the user when the heating element is warm. Below we discuss a number of use cases that these example design illustrate.

#### Remote Interactions

Every retrofit interface created by the user is available through the RetroFab mobile application. This makes it possible to control devices and appliances remotely, such as the light switch when one forgets to turn off the lights. A retrofit interface can also serve as a remote for another retrofit interface. Turning the lamp off when going to bed, turns off the lighting in the room as well, using the retrofitted wall switches.

### Locking Out Controls

Digital, as well as physical remotes, make it easy to hide potentially hazardous controls for children. The remote control for the stove can be relocated to a more secure area, or protected further using a key lock (Figure 4.16e). At the same time, the attached control on the oven contains no physical interface, making it impossible to operate without the remote control.

### Resolving Design Flaws and Frustrations

RetroFab also facilitates the process of resolving poor design decisions found in physical interfaces. Controlling the dials located on the back panel of the stove requires moving one's arm over a number of elements, which could have pots or frying pans on them. RetroFab allows for repositioning these controls to a more convenient or safe location, such as the side panel of the stove. Setting the time and alarm on an ordinary alarm clock (Figure 4.16d) is often tiring. By retrofitting the interface using RetroFab, a shortcut can be designed for automatically setting the current time after the lock is unplugged or a power outage occurs. This is done by instructing the actuators to press and hold the hour and minute buttons for a calibrated time interval, to increase the time from the known 12:00 start position to the current time.

### Shortcuts for Frequently Used or Personalized Actions

As highlighted in the walkthrough, the retrofit interface for the toaster can integrate a personalized button for automatically toasting bread to one's favorite toast settings. Similarly, by retrofitting different wall switches in the home, one can make new buttons that serve as shortcuts for different lighting settings.

### Facilitating Interactions for Users with Special Needs

People with disabilities are often unable to operate controls that are found on most devices, as they require considerable amounts of force or fall outside the range of motion they are capable of. The retrofitted desk lamp illustrates how a rocker switch can be converted to a lower force control, such as a push button. A similar push button is used in the retrofit interface of the toaster to replace the heavy mechanical lever.

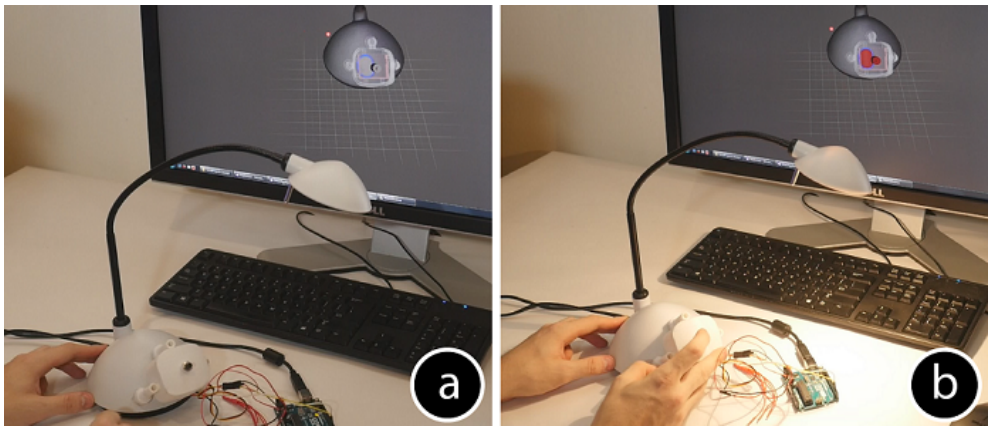


**Figure 4.17:** Real-time monitoring of the RetroFit interfaces.

#### Statistics on Appliance Usage

Since RetroFab intercepts interactions for every retrofitted control, actions can be tracked and visualized in real time on a fine-grained level (Figure 4.17). Using this information, statistics over longer periods of time can be compiled to give, for example, data on how often someone presses the snooze button on their alarm clock.

## 4.8 User Study: Retrofitting a Desk Lamp

To understand the experience of working with RetroFab, an informal guided design session was conducted with four participants. Two participants (P1, P2) were experienced CAD users, while the other two (P3, P4) had only limited experience with 3D modelling. P1 had extensive experience in electronic circuit designs, whereas P2 and P3's knowledge was limited to basic prototyping with Arduino, and P4 had no experience with electronic circuits. Each session lasted for approximately 45 minutes.

Participants were first introduced to the concept of retrofitting legacy devices. Then, the participants were introduced to the RetroFab design tool using the example of the retrofitted wall switch (Figure 4.16a). Once they understood the different concepts, participants were instructed to retrofit the desk lamp using RetroFab (Figure 4.16b). Due of time restrictions, the generated enclosure structure was 3D printed beforehand and was given to the user during the assembly phase, after they successfully designed their own retrofit

enclosure structure. Participants then assembled the 3D printed objects and the electronic circuit by following instructions on the screen. Finally, they deployed the retrofitted desk lamp and controlled it from the RetroFab mobile application. Participants reported their experience with RetroFab through a questionnaire.

All participants were able to retrofit the desk lamp in less than 25 minutes and saw clear benefits in using RetroFab. Participants perceived the entire process as enjoyable and were satisfied with the end result. They reported that the outcome met their expectations. Three participants (P1, P2, P3) felt they could design a working prototype without using RetroFab, however, they all agreed it would involve multiple iterations and span multiple days. All participants appreciated the straightforward, step-by-step process of RetroFab. They indicated RetroFab would be very useful to them for retrofitting legacy devices in the future. P4 highlighted that RetroFab was an enabling technology for him as he would not know how to retrofit devices without the tool.

P1, P2 and P4 mentioned they are looking forward to see how future versions of RetroFab allow for more customization of generated enclosure structures, such as embedding the enclosure structure design inside a 3D model of choice or giving a retrofitted object a cartoon-like appearance. At the same time, these participants noted that precise placement of RetroFab components inherently allow for anthropomorphism, e.g. making a smiley face with RetroFab components.

Participants recognized that this approach would be useful in different situations, such as controlling the heating at home remotely and saving energy, or for adapting interfaces for impaired users. They all indicated that they would consider deploying this technology at home.

## 4.9 Discussion

Our preliminary study demonstrates that for users without a technical background, RetroFab is an enabling technology to make retrofit structures in order to adapt physical user interfaces. For users with some experience in 3D modeling, programming, or electronics, RetroFab significantly speeds up the design process as many complex and time consuming tasks are automated. As such, a working retrofit interface is produced in the first design iteration. On the flipside, the RetroFab environment can be extended in several ways:

First, RetroFab only generates attached enclosure structures when there is space on the 3D scanned model for attaching the structure. For instance, very small controls are not supported, such as lamps that have small rocker switches

integrated in the power cable. To retrofit these devices, future versions could support wraparound enclosure structures that entirely enclose these kind of controls.

Second, the RetroFab toolkit currently supports actuators optimized for operating basic controls used in appliances. In the future, multiple actuators could be developed of various shapes and sizes to reduce size and cost and provide an optimal actuator for each use case. Bigger and more powerful actuators would, for example, allow for retrofitting heavy duty mechanical controls, such as the linear actuator already supported, controls used in industrial machines, and handles to adjust car seats. Another interesting direction for future research is the support of more high-fidelity sensors, such as microphones and cameras, besides the light sensor that is already supported. Cameras and image processing techniques, could allow retrofitting more complex legacy interfaces that communicate states using displays.

Last, the current implementation of RetroFab requires actuators and sensors to be positioned directly in front of legacy controls. In the future, advanced transmission mechanisms could be supported to relocate the actuators out of sight, behind the legacy device, in order to improve the aesthetic appearance of enclosure structures. One could imagine using a single actuator to activate multiple controls to make the retrofit interface smaller. Besides this, the aesthetic appearance could be improved by allowing the user to remodel the enclosure design. For example, designing retrofit toaster with the "look and feel" of a slot machine by merging the generated enclosure structure with a 3d model provided by the user.

## 4.10 Summary

In this chapter we presented RetroFab, a design and fabrication environment that enables laypeople to change the behavior and layout of existing physical interfaces and appliances by retrofitting them (G2). RetroFab automates parts of the 3D modeling, programming and circuit wiring process into a streamlined workflow (C5). More specifically, our software assist in the 3D modeling process by computationally generating a retrofit structure from an annotated 3D scan of the existing physical interface (C2). Besides remodeling the layout of the interface, non-programmers change the behavior and interconnect existing physical interfaces using the integrated Pulsation logic specification paradigm (Section 3.5) (C4). The retrofit interface can be produced with any 3D printing technology, including low-cost easy-to-use FDM printers (C1). A custom-generated tutorial assists users in assembling the retrofit interface and the com-

ponents of our easy-to-use mechatronic tookit (C3). The example designs included in this chapter show that many people could benefit from retrofitting interfaces, most prominently members of the maker community, IoT-developers, and researchers. One particularly interesting target audience for retrofit devices are caregivers for disabled or elderly individuals. Retrofitting could allow people with disabilities regain independence and operate legacy interfaces they would otherwise be unable to. A preliminary evaluation demonstrates the utility and usability of our system and workflow for both laypeople and users with a background in electronics or 3D modeling.

# Chapter 5

## Pulsation 2.0: Visual Programming for Physical Interfaces

Expressing the behavior of physical interfaces traditionally requires non-programmers to follow extensive tutorials, workshops, or courses to learn the necessary programming skills [Qi 14, Mellis 13]. Researchers investigated several techniques to make it convenient for non-programmers to specify logic behavior [Kelleher 05]: (1) Syntax abstraction techniques, such as Scratch [Resnick 09] abstract syntax details using building blocks. However even for simple programs, some exposure to different programming constructs, such as loops, variables, functions, etc. is hard to avoid. (2) Simplifying the language to only very basic constructs, such as the IFTTT[1] service. Although suitable for simple condition-action rules, this approach is often too limited in design environments that allow for making a wide variety of designs. (3) Tailoring the language for a specific domain of programming problems. This approach has been taken in a many of domains, including 2D (e.g. NodeBox[2]) and 3D (e.g. Grasshopper Rhino[3]) renderings as well as audio processing (e.g. PureData [Puckette 96]).

The Pulsation visual programming paradigm, presented in Section 3.5, takes this last approach and is optimized for specifying functional relationships between electronic components in sensor-based systems (C4). In many sensor-based systems, the majority of behavior is influenced by some sort of temporal

[1] https://ifttt.com
[2] https://www.nodebox.net
[3] http://www.grasshopper3d.com

behavior. Examples include, actions triggered after a temperature change over time, a button that has to be pressed fast or long enough, an LED or audio signal triggered at a certain frequency, conditions that need to be completed in a certain order or at the same time, etc. Specifying these temporal conditions using traditional programming languages involves constructs, such as variables, loops, functions, timers, etc. Learning and combining these different constructs has a steep learning curve and is often cumbersome for non-experts. The Pulsation logic specification technique is optimized for expressing temporal behavior of sensor-based systems and allows non-programmers to specify behavior without following extensive tutorials. In chapters 3 and 4, we showed how Pulsation is embedded in respectively the PaperPulse and RetroFab design environments. The user evaluations and design sessions with PaperPulse revealed several limitations of Pulsation 1.0:

1. *Out of context logic specifications*: Pulsation 1.0 visualizes logic in a separate visual representation. Linking logic behavior back to electronic components, once specified, is often hard and requires careful inspection.

2. *Control over variables*: Pulsation 1.0 only provides access to elementary variables of electronic components (e.g. value of a slider, state of a button, brightness of an LED). As custom variables cannot be created, every condition relates back to one of the elementary variables in the design. This significantly increases the complexity as well as the total number of conditions.

3. *Several constructs limit the expressive power of Pulsation 1.0*: (1) No support for combining conditional (if–then) and mapping (map–to) constructs. (2) No support for derived parameters (e.g. speed, progress, etc.) in conditional statements. (3) Lack of precise control over parameters, such as excluding events while matching conditions, fine grained control over conditions that have to be completed sequentially over time, and variable instead of constant timing parameters.

The first limitation impedes the ease-of-use for first time users (low threshold). The other limitations interfere with supporting a wide variety of design variations (high ceiling). To resolve these limitations and provide a higher ceiling as well as a lower threshold, this chapter revises the logic specification approach in a new version, we call Pulsation 2.0.

## 5.1 Brief System Overview

Pulsation 2.0 language is based on a well-defined grammar and is supported by a dedicated interpreter. Similar to the first version of Pulsation, Pulsation 2.0 is optimized for running on microcontrollers but also runs on desktop computers for simulation purposes. The language consists of visual elements in which the program logic is specified. The limitations of the first version, discussed in the previous section, are addressed as follows:

1. In Pulsation 2.0, users specify logic by drawing visual links between electronic components and their associated attributes on the canvas. The logic behavior is thus presented as an interactive graph in which the nodes represent conditions and actions, while the edges link their respective event handlers. This lowers the threshold for users to get started with Pulsation as users rely on a visual programming paradigm instead of referring to attributes using abstract names. Furthermore, the visual design and logic behavior is presented in a single integrated representation to ease interpretation.

2. Pulsation 2.0 presents novice users with elementary parameters of electronic components present in the design (e.g. value of a slider, state of a button, brightness of an LED). Experienced users, however, can create custom variables and access internal variables managed by the Pulsation interpreter to realize advanced constructs.

3. All logic behavior is integrated in a single logic rule: conditional behavior. Mapping constructs (map–to rules) are available as advanced assignment actions that execute a transformation function on a variable. Offering a single logic construct lowers the threshold for novices. At the same time, it increases the ceiling for experienced users as mapping behavior is supported within conditional statements, which was not supported before. To increase the ceiling further, derived parameters (e.g. speed, progress, etc.) are available as functions for any expression. Additionally, constants and variables are interchangeable throughout Pulsation 2.0.

   Besides these advancements over the first version, Pulsation 2.0 also has several strengths compared to existing visual programming methodologies introduced in Section 2.3. Unlike visual programming approaches to avoid syntax errors, such as Scratch [Resnick 09], Pulsation 2.0 facilitates authoring the behavior of electronic components as every logic construct offers precise control over timing properties. In contrast,

Scratch [Resnick 09] or visual programming approaches supported by LEGO Mindstorms [Barnes 02, Erwin 00, Kim 07] require users to construct complex algorithms using timers in order to match or produce patterns over time. Amongst the visual programming approaches that target specific domains, such as PureData [Puckette 96] for audio processing or Nodebox[4] graphical renderings, LabView [Wells 96] and programming approaches in Loxone[5] are optimized for sensor-based systems, similar to Pulsation. However, both systems target experts and thus require users to follow extensive tutorials. The IFTTT system[6] on the other hand, simplifies programming to elementary conditional behavior involving only a single condition stimulus and action. IFTTT therefore does not offer the expressive power supported by Pulsation 2.0.

In Pulsation 2.0, we replaced the *input set* and *output set* terminology, used in Pulsation 1.0, with *conditions* and *actions* as all logic constructs are part of conditional behavior.

## 5.2 Pulsation 2.0 Grammar

The Pulsation 2.0 grammar consists of five constructs: variables, conditions, actions, and events. Conditions are boolean expressions that evaluate the state of variables, actions change variables, and events of conditions link to events of actions. Pulsation has constructs that are specific to conditions or actions to support loops and encapsulations (i.e. condition/action repeaters and condition/action encapsulations). By only making these constructs available when composing either a condition or action, Pulsation eliminates the complexity that comes when users compose functions or loops consisting of both conditions and actions. Additionally, this separation significantly simplifies the concept of logic connections. In this section, we elaborate on the different logic constructs available.

### 5.2.1 Variables

In Pulsation 2.0, elementary variables, inherent to specific components (e.g. brightness variable of an LED or state variable of a switch), are automatically available and appear to the user as a simple property of these components. For

[4] https://www.nodebox.net
[5] http://www.loxone.com
[6] https://ifttt.com

complex designs, requiring additional variables to store intermediate states, custom variables are supported in Pulsation.

### 5.2.2 Conditions

Pulsation 2.0 supports a wide range of basic conditional grammar for evaluating the state of variables, including variable in high/low state, variable equals, variable in between range, variable greater/smaller than, variable changed value. Conditions related to Pulsation actions are also supported, such as evaluating whether an action starts executing, is paused, or resets.

Condition aggregates consist of multiple conditions and a temporal relationship. Pulsation supports the following temporal relationships for aggregates:

- Conjunction: True when all conditions are completed.
- Disjunction: True when at least one of the conditions is completed.
- Simultaneous: True when all conditions are completed at the exact same time. Child of child conditions are not subject by the grandparent condition node with a simultaneous temporal constraint.
- Ordered: True when all conditions are completed in the specified order. Child conditions are allowed to match multiple times. Child of child conditions are subject by the grandparent condition node with an ordered temporal constraint to enforce the order of matches.
- Ordered-Strict: Same semantics as the ordered temporal constraint. Additionally, the ordered-strict constraint enforces child conditions to match exactly once, except for conflicting conditions.

Condition aggregates also support a modal operator similar to Linear Temporal Logic (LTL) specifications [Wolper 83]. The modal operator in Pulsation works as an electronic latch and ensures that a condition remains completed for its parent condition even when it is not satisfied anymore. This operation is oftentimes required when a condition consists of conflicting child conditions. For example, a condition specifying that a button has to be pressed and then released. In this example, at least the first child condition requires using the latch operator. Pulsation also supports an advanced modal operator to specify that a condition needs to remain satisfied until a certain moment (external condition) before the latch takes over. Every aggregate condition also list conditions that cannot be satisfied while completing the aggregate condition. When one of these conditions do take place, the aggregate construct resets.

Derived condition operators calculate a specific derived parameter of the child condition, examples include:

- Repeater: The number of times the child condition is completed.
- Inverse: The inverse of the condition.
- Time to complete: Duration of time to complete the first and last condition of a condition aggregate.
- Time completed: Duration of time the condition is already satisfied.
- Progress: Percentage of conditions completed.
- Speed: Speed with which child conditions are completed.

These derived parameters are also available as variables and can be used in other conditions or as the "stop-condition" of the derived condition e.g. repeater is satisfied when child condition is completed two to five times. Arbitrarily complex constructs are realized by nesting condition aggregates and derived condition operators.

### 5.2.3 Actions

Pulsation supports a wide range of actions including, assignments, increments, decrements, and other basic mathematical operations on variables. Besides, assignments that are subject to more advanced mathematical functions are supported (mapping constructs). For example, transitioning the value of a variable (e.g. brightness of an LED) over a period of time.

Similar to conditions, Pulsation also supports aggregates and derived actions. While basic action aggregates encapsulate actions that execute all at once, action aggregates with time support allow for specifying delays between actions. We also propose two derived action operators: repeaters and progress regulators. Repeaters loop over child conditions until the stop condition is completed. Progress regulators execute a variable number of child actions.

### 5.2.4 Events

To associate conditions and actions, Pulsation supports a set of events for conditions, including activated, deactivated, tick active (raised every CPU cycle during which the condition is satisfied), tick not active (raised every CPU cycle during which the condition is not satisfied). Similarly several events/subactions are supported for actions, including execute, pause, resume, and reset.

In contrast to all other logic constructs, interconnections between these events are established outside the interpreter and thus interpreted directly by the microcontroller. This significantly reduces the memory consumption as projects oftentimes consist of a high number of logic connections. Besides, these constructions allow advanced users and programmers to link custom-written code to Pulsation actions or conditions.

### 5.2.5 Grammar Instance

Appendix A shows how to initialize the grammar of a more complex interactive system. The grammar instance specifies the conditions, actions, and events for an advanced code slot mechanism on a numpad. In this example, pressing button 1 needs to be followed by simultaneously pressing buttons 8 and 9. This sequence has to be repeated twice within 10 seconds to complete the code slot. The condition sequence specifies that when invalid buttons are pressed, the sequence resets. When the input sequence is completed in time, an LED turns on to notify the user that he is logged in. When the user starts interacting with the numpad but does not enter the correct sequence in time, a buzzer repeatedly beeps. To help users observe their progress, the progress through the event sequence is represented by 6 LEDs, one for every button that has to be pressed.

## 5.3 Pulsation 2.0 Visual Logic Specifications

We simplify composing Pulsation grammar by allowing users to draw visual links between electronic components. Although visual links improve the visibility and overview for non-experts, it is oftentimes unclear how simple links translate Pulsation grammar. Even for simple components, such as linking a pushbutton to an LED, dozens relationships are possible including, turning the LED on when pushed and turning it off when pushed again (toggle), turning the LED immediately off when the button is released. Alternatively, the LED can fade in over time or blink. Therefore we devise an interaction paradigm for constructing a detailed logic graph to precisely link conditions to actions. As our Pulsation grammar offers unlimited possibilities for composing condition and action, our technique is scalable while at the same time simple to grasp for first-time users.

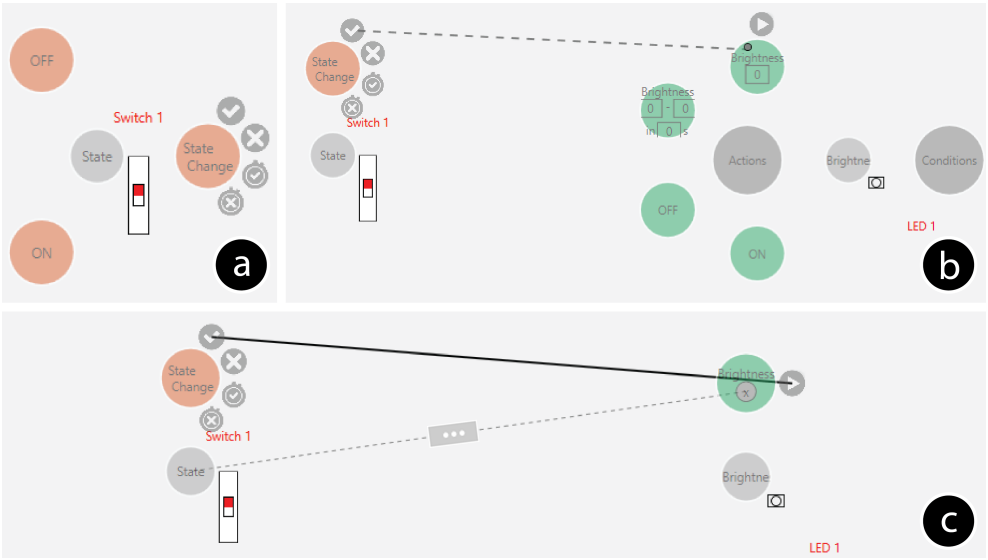


**Figure 5.1:** Linking the state of a switch to the brightness of an LED.

### 5.3.1 Conditions and Actions with Individual Variables

Figure 5.1 demonstrates the basics of the Pulsation visual logic specification approach: (a) Clicking on an electronic component reveals a hierarchical radial menu consisting of conditions and actions that relate to the variable associated with that component (e.g. brightness variable of an LED). This includes basic conditions and actions (e.g. brightness equals x, turn LED on), as well as derived conditions and actions (e.g. LED on for x seconds, fade LED from x to y in t seconds). While hovering over these conditions and actions, associated events appear. (b) Events of conditions are linked to events of actions by drawing visual links using a drag-and-drop interaction style. (c) We specify in this example that every time the switch changes its state, the brightness of the LED is assigned to the state of the switch.

To facilitate the readability and comprehension of conditions and actions, variables, have names that reflect their meaning (e.g. brightness for LEDS, state for switches, and value for sliders). However, concepts, such as assignments of variables and constants might still be hard to grasp by first-time users. Therefore, redundant conditions and actions are available which are automatically tuned with frequently selected constants. Examples include conditions, such as button pressed, button released, switch on, switch off; or actions such


**Figure 5.2:** Alternative solution for linking the sate of a switch to an LED.

as LED on and LED off. Figure 5.2 shows an alternative solution for linking a switch to an LED. In this solution, the on and off-state are individually linked to the minimum and maximum brightness of the LED.

The Pulsation logic specification approach includes context aware strategies. When connecting to/from a central condition or action node in the radial menu, the system automatically selects the most appropriate leaf node in that context. For example, linking a switch to an LED automatically links the most appropriate condition i.e."switch on" to the most appropriate action i.e."LED on". Additionally, the system suggests to automatically add an undo operation to link the "switch off" condition to the "LED off" action. This context-aware decision support mechanism ensures that interactive systems are always operational and users gradually refine and discover more options.

In contrast to the first version of Pulsation (Section 3.5), Pulsation 2.0 solely consists of conditional rules. Mapping constructs in which the value of a variable (e.g. slider) is continuously mapped to another variable (e.g. brightness of an LED) are realized using the "tick" events supported by conditions (Section 5.2.4). Figure 5.3 shows how the value of a slider is continuously assigned to the brightness of an LED as long as the switch is in the on-state. To perform this mapping independent of the state of the switch, a special application event exists that is triggered every CPU cycle (Figure 5.4). Mapping constructs are thus considered as assignment actions. By expanding the options associated with a reference link to a variable, a transformation function

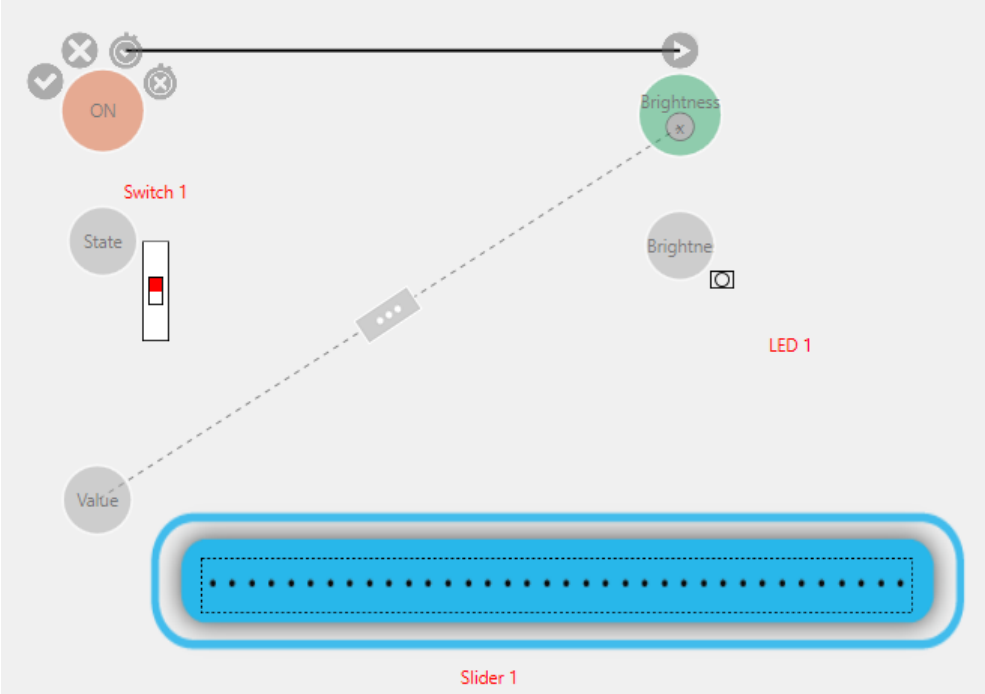


**Figure 5.3:** Continuously assigning the value of a slider to the brightness of an LED when a switch is in the on-state.

can be specified. Figure 5.4 shows how to invert the value of the slider before assigning it to the brightness of the LED.

By default, only the parameters of widgets added to the canvas are shown on the screen. More advanced users can also add custom variables to the canvas to save intermediate states. For example, when assigning the inverted signal of a slider to the brightness of multiple LEDs, the inverted value can first be stored in an intermediate state and then be assigned to the brightness of all LEDs. Without custom variables, an additional transformation function would be needed between the slider and every single LED. This would increase the complexity of the visual code and have a negative impact on the readability and evolvability of the code.

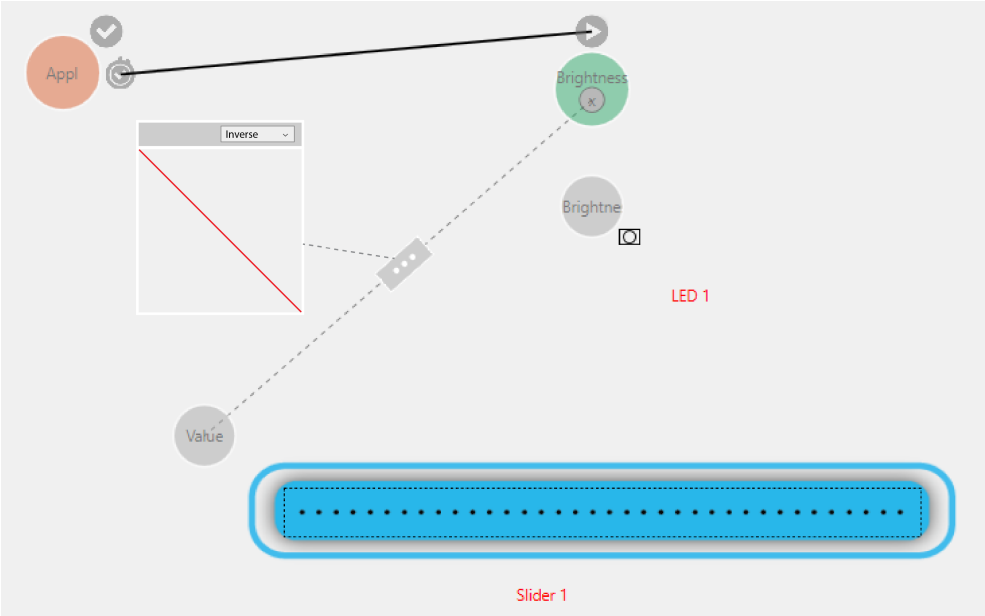


**Figure 5.4:** Continuously assigning the inverse value of a slider to the brightness of an LED.

### 5.3.2 Composite Conditions and Actions

The radial menus presented in the previous section offer conditions and actions in the context of a single variable. Advanced conditions and actions consisting of multiple variables, are authored using a condition and action composer. Using these advanced composers, aggregate conditions/actions are nested to support precise control of timing parameters and relations between variables.

Figure 5.5 shows a condition composer consisting of a nested aggregate and derived condition to specify that three switches have to be turned on, in a sequential order, within 5 seconds to make the LED fade in. More aggregate conditions, derived conditions, or the modal operator (Section 5.2) are added using the "add operation" drop-down menu. The five derived conditions presented in the grammar (Section 5.2), are available through this menu, including *time to complete*, *time completed*, *speed to complete*, *inverse*, *repeater*, and *progress*. Some of these derived constructs automatically add derived variables to the system, such as time, speed, number of repetitions, etc. These variables are used similarly to regular variables, although conditions on these variables oftentimes link back to the same derived construct to realize stop-conditions (e.g. minimum number of repetitions or maximum time to start and complete the condition).

By clicking on the header of aggregate conditions, one can loop through the different temporal relationship (i.e. conjunction, disjunction, simultaneous, ordered, or ordered-strict). Conditions are dragged in the "exclude" section of an aggregate construct to specify states of variables that may not be registered while completing the aggregate condition. Using the black arrow shown in the right-bottom corner of every aggregate or derived condition, one links events of sub-conditions in the nested hierarchy.

Figure 5.6 shows an action composition that is triggered by a condition composition. In this example, the push button needs to be pressed for four seconds and released to complete the condition and make the LED blink twice. The action composer works similar to the condition composer and offers derived action constructs, including a repeater and progress regulator. A time-line is featured to precisely control the delay between actions. Actions that take place over time support additional events/sub-actions, as presented in the grammar in Section 5.2.4 (i.e. restart, pause, resume).

We now reconsider the interactive card for learning mathematical sums that was designed during the workshop with PaperPulse (Figure 3.26c). In the first version of Pulsation, this design required 45 rules, one to check the validity of every possible combination of the three slider values. In Pulsation

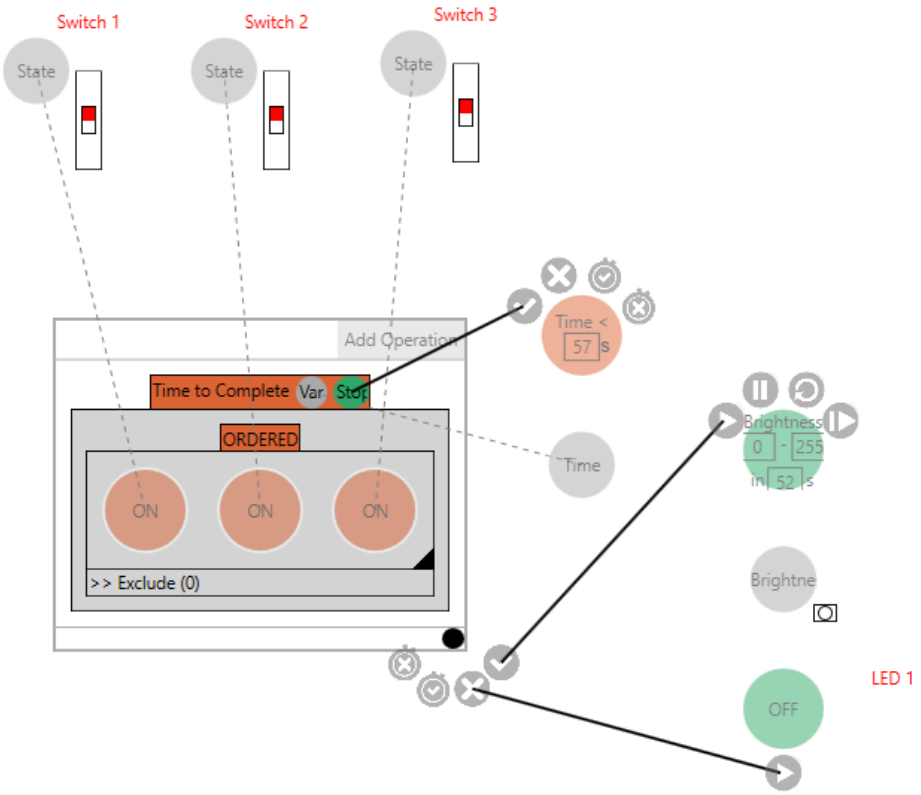


**Figure 5.5:** Turning all switches to on within 5 seconds will start fading the LED.

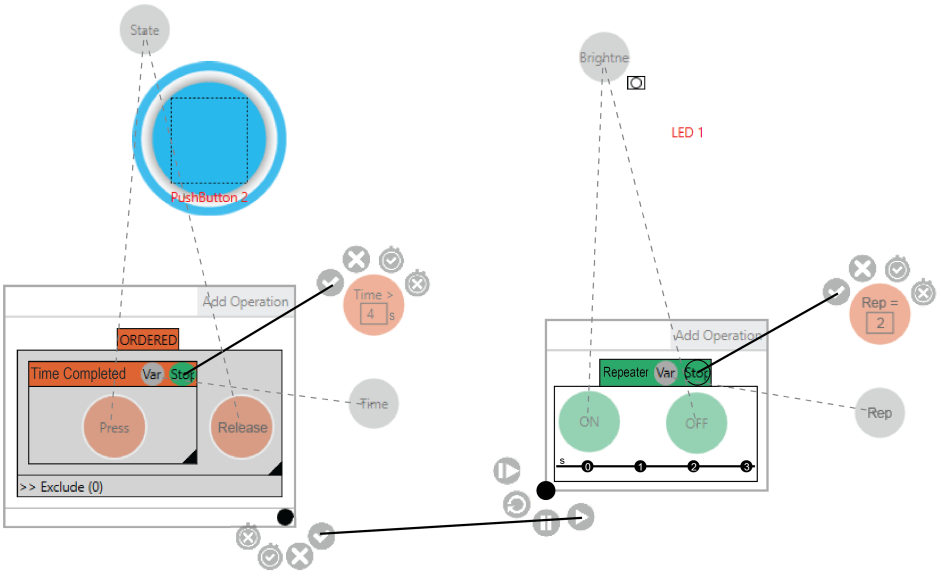


**Figure 5.6:** Pressing the push button for four seconds and releasing it afterwards makes the LED blink twice.

2.0, this behavior can be specified with only 12 logic rules. The solution consists of three custom variables, one for every pull-chain slider. Eleven logic rules, one for every value in the pull-chain sliders, ensure that the currently selected value (displayed value on the slider) is represented in the corresponding custom variable. The last logic rule verifies the validity of the sum. Specifying the behavior of this solution requires significantly less effort, is less error prone, and is easier to maintain as compared to the first solution.

## 5.4 Architecture and Implementation

Similar to the first version of Pulsation (Chapter 3), Pulsation 2.0 is implemented in C# and uses the .NET Micro Framework specifications to ensure its portability to microcontrollers supporting the .NET MF. We developed a Pulsation interpreter that evaluates condition grammar and executes action grammar. Pulsation grammar is uploaded to the microcontroller by generating code that re-instantiate the entire Pulsation grammar hierarchy that was composed by the user (Appendix A). In contrast, events between conditions and actions are established by generating event handling mechanisms using the .NET CodeDom, directly in C#. On the one hand, this reduces the amount of working memory that would be required to model all events in the Pulsation interpreter. On the other hand, more experienced users, with programming experience, can write custom code that links to the generated project using these traditional event handlers.

Every Pulsation project consist of 4 assemblies that are uploaded to the microcontroller: the core interpreter, condition interpreter, action interpreter, and generated code. The core interpreter assembly requires 8kb of program memory and consists of the main interpreter loop, utility functions, and code to manage Pulsation variables. The condition interpreter assembly requires 32kb of program memory and consists of all condition constructs supported by Pulsation. The action interpreter assembly requires 20kb of program memory and consists of all action constructs supported by Pulsation. The fourth assembly includes the generated code for instantiating the Pulsation grammar and event handlers. The size of this assembly depends on the complexity of the project and on average requires less than 10kb of program memory. The three interpreter assemblies are compiled upfront. The assembly containing the generated code is compiled at run-time within the design tool, every time the user updates the Pulsation logic.

Similar to the first version of Pulsation, Pulsation 2.0 achieves a modular design. Uploading all supported module requires a minimum of 60kb of

program memory on microcontrollers. However, future versions could optimize these memory requirements by compiling only the logic constructs that are used in a specific project. This significantly reduces the size of Pulsation projects as only a very small subset of the Pulsation logic constructs are used in the majority of projects.

Every logic construct requires one byte of working memory memory to store its internal state. Pulsation optimizes memory consumption by dynamically selecting the type of Pulsation variables based on the value resolution that is required. By default, variables representing time consume only a single byte but are limited in resolution to a tenth of a second. Additionally when the number of Pulsation variables does not exceed 255, Pulsation variables are stored in a custom bank of memory that is addressed with a single byte.

## 5.5 Pulsation 2.0 Effectiveness Attributes

To demonstrate the effectiveness of our approach, we show how Pulsation 2.0 reduces solution viscosity and supports power in combination. These are two important attributes that contribute to the effectiveness of visual programming paradigms and authoring tools in general, as stated by Olsen [Olsen 07].

### 5.5.1 Reducing Solution Viscosity

Programming approaches with a low solution viscosity minimize the effort required to iterate over different solutions. Pulsation 2.0 reduces the solution viscosity in three ways:

1. *Flexibility*: Composing and altering logic is convenient with Pulsation. By simply linking basic logic constructs, the behavior of electronic components is changed. These changes are immediately compiled and interpreted. As such, the user can verify changes quickly using the Pulsation simulator.

2. *Expressive Leverage*: is achieved when a tool reduces the total number of choices that a designer must make to express a desired solution [Olsen 07]. Pulsation 2.0 reduces the number of design choices by automatically instantiating options that are implied by the user's design choices. First of all, Pulsation reasons that when designing physical interfaces with electronic components, there is a fundamental need for precise control of timing properties. Hence, action and condition primitives in Pulsation are mainly oriented towards control over time.

Secondly, adding an electronic component automatically declares new variables in the system to represent internal states. Depending on the type of the electronic component, the variable has a specific type e.g. binary, continuous, discrete. Additionally, Pulsation makes it convenient for users to use frequently selected values depending on the type of variable, such a LED in the on or off-state. Similarly, some condition and action constructs automatically declare variables, including the number of repetitions for repeaters, or the time to complete a condition. Finally, when composing basic conditions, such as an LED turning on when the switch is in the on-state, Pulsation automatically suggests adding the inverse/undo operation.

3. *Expressive Match*: refers to how close the means for expressing design choices is to the problem being solved. Pulsation 2.0 adheres to this principle by allowing users to simply draw links between logic constructs. Using these visual links, abstract names for referring to electronic components and logic constructs are avoided. Although some advanced Pulsation constructs introduce additional layers of abstraction on top of the problem being solved, first-time users interconnect basic states by simply drawing links between electronic components. Additionally, the conditions and actions are context specific e.g. LED on/off or transitions in brightness over time, push button pressed/released. In contrast, traditional programming approaches offer constructs which are independent of the variables used in the context. Similarly, Pulsation facilitates using time constructs without relying on abstract timers or timestamps as in traditional programming approaches.

### 5.5.2 Power in Combination

Olson [Olsen 07] defines *Power in Combination* as the way user-interface tools can support new components to create new solutions. First of all, Pulsation 2.0 offers unlimited design variations using a limited set of primitives consisting of variables, conditions, actions, and events. These primitives are combined and nested to realize advanced behavior. Similar to primitives, advanced condition and action constructs can be used in other primitives, thus realizing unlimited design variations. Secondly, all electronic components and logic constructs realize their behavior by manipulating Pulsation variables. This makes it convenient to interchange or introduce new components to Pulsation.

## 5.6 Discussion

When designing a visual programming language, it is always challenging to select which parameters should be concealed and which should be revealed to the user. Revealing too many parameters increases the threshold for first-time users while concealing many parameters will not allow for many design variations (low ceiling). Pulsation reveals high-level parameters that are especially relevant when specifying logic for sensor-based system (e.g. timing properties). Pulsation 2.0 advances the first version of Pulsation (Section 3.5) and offers a higher ceiling by supporting a more comprehensive logic grammar and custom variables. At the same time, Pulsation 2.0 lowers the threshold for first-time users by representing logic behavior as a visual graph. As such, users do not have to write textual code statements that are prone to syntax errors, or refer to constructs using abstract names. Additionally, the system assist users in composing this logic graph. For example, Pulsation 2.0 automatically interconnects the most appropriate conditions and actions when two electronic components are linked.

On a high level, Pulsation 2.0 grammar consists of variables, conditions, actions, and events. Basic constructs available in traditional programming languages, such as functions (encapsulation) and loops are available within condition and action constructs. This lowers the threshold for first-time users as Pulsation does not require learning many different programming concepts. Pulsation 2.0 allows users to interconnect conditions and actions and gradually advance to more complex constructs. Our programming approach therefore targets users who want to "use programming [Kelleher 05]". Transitioning from programming in Pulsation to traditional programming languages is less trivial, therefore the current version of our programming approach is less suited for students to learn basic programming concepts.

As many embedded systems are influenced by some sort of temporal behavior, the majority of logic constructs in Pulsation 2.0 offer control over timing parameters. Although Pulsation 2.0 is demonstrated in this chapter in the context of the PaperPulse design environment (Chapter 3), the approach can be integrated in other systems to facilitate programming sensor-based systems.

Despite the flexibility of Pulsation 2.0, it can be extended in the future in several ways: First, Pulsation 2.0 allows users to encapsulate and thus reuse composed constructs in other conditions or actions. However, the current implementation does not support parameterizing logic constructs to reuse them within a different context. For example, a condition that verifies whether a switch is in the on-state can be encapsulated in another condition to verify

whether a button is pressed at the same time. However, that same condition cannot be used to verify whether another switch is in the on-state. Future versions could support parametrization options for advanced users.

Second, the number of links between logic constructs can expand quickly when specifying complex behavior. This clutter can be reduced in the future, by exploring visualization and interaction techniques for graphs. For example, the edges of the graph can be routed to avoid intersections. Additionally, only edges that the user is currently investigating can be revealed by hovering over the associated nodes.

Third, the current version supports basic electronic components, such as LEDs, buttons, switches, and sliders. Future versions can extend the set of electronic components and support sensors, such as accelerometers, graphical displays, sensors supporting touch or gestural interaction, or wireless communication. Similar to basic electronic components, these advanced components can be supported as widgets on the canvas. Although advanced sensors could be programmed using elementary logic constructs supported by Pulsation 2.0, such as thresholding, average, and basic mathematical operations, specifying more complex behavior could become cumbersome. Future versions could support high-level parametric actions and conditions, such as movement patterns for accelerometers or visual primitives for graphical displays. For some sensors it might also be appropriate use a programming-by-demonstration approach [Hartmann 07] and allow users to specify input or output patterns by demonstrating patterns directly using the sensors. Alternatively, advanced users and programmers can link custom code to events in Pulsation.

Last, to lower the threshold further for realizing advanced constructs with Pulsation, more abstraction layers can be added in the future to automatically fill-in appropriate logic details. However, automatically adding the correct logic behavior is non-trivial and oftentimes depends on the context in which it it used. Therefore, advanced algorithms and machine learning techniques could be considered to automatically suggest logic details based on the behavior already specified.

## 5.7 Summary

In this chapter we presented Pulsation 2.0, a visual logic specification paradigm that resolves the limitations of the first version (Section 3.5) by offering in-context logic specifications, custom variables, and additional expressive power. As such, Pulsation 2.0 provides a higher ceiling and lower threshold to get users started (C4). Pulsation is optimized for specifying the behavior of sensor-based

systems for users without a programming background. In contrast to the first version, Pulsation 2.0 only consists of variables, conditions, actions, and events. This is essential to lower the threshold for first-time users as they can simply draw relations between electronic components. More advanced logic behavior is realized by adjusting additional parameters or using advanced compositions. Similar to the first version, Pulsation is optimized for microcontrollers but also runs on desktop computers for simulation purposes.

# Chapter 6

## Paddle: Real-Time Physical Transformations

### 6.1 Introduction

The systems outlined in Chapters 3 and 4 offer an enabling technology for laypeople to fabricate new and adapt existing physical interfaces. However, the workflow to make these interfaces, including the production time of DIY machinery, ranges from a few hours until a full day. Although this is acceptable for designing systems that only require changes once in a while, these fabrication approaches are not suitable to create interfaces that require frequent changes in their physical layout. Examples include tactile keyboards on mobile devices, which ideally would support different physical configurations for different modes, settings, or applications, depending on the context (e.g. application) that is currently being used. Hence, these tactile keyboards on mobile devices have been replaced by touch screens at the expense of tangible interactions.

To preserve the richness and qualities of tangible controls, such as physical affordances [Ishii 97] and eyes-free use [Fitzmaurice 95], in this chapter, we investigate technologies that enable laypeople to adapt the physical configuration of interfaces in real-time (G3). More specifically, we explore techniques to embed materials in interfaces that allow for fast transitions in their physical state (C1). As such, the physical interface itself transforms to various physical controls depending on the context of use. We demonstrate this concept in the context of mobile devices. Especially in these settings, supporting a diverse set of physical controls is challenging, as outlined in the previous paragraph.

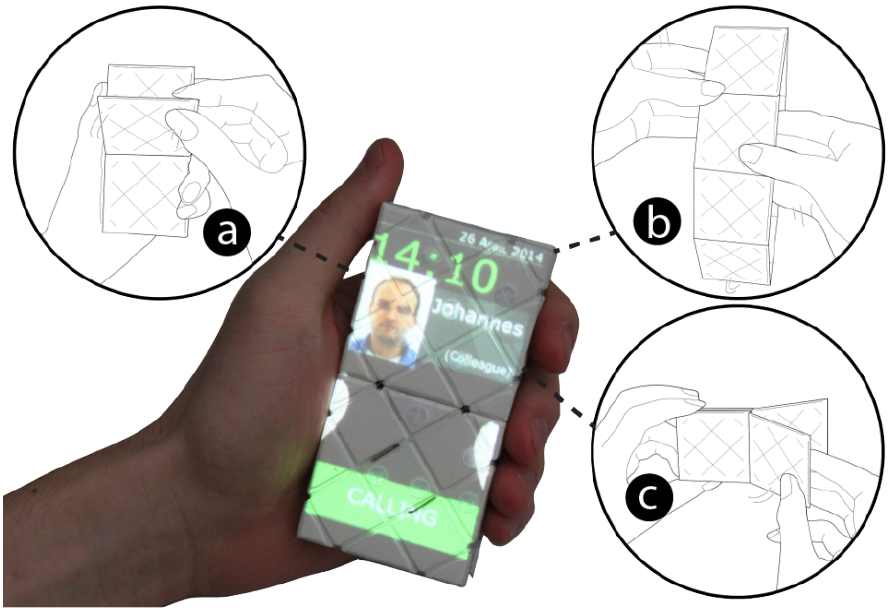


**Figure 6.1:** Paddle supports several physical controls, including (a) peeking, (b) scrolling, (c) leafing.

Figure 6.1 shows our prototyped mobile device, called Paddle, that can be transformed to various physical controls, including a window to peek at content (Figure 6.1a), a ring to scroll through lists (Figure 6.1b), and a book-like form factor to leaf through pages (Figure 6.1c).

Previously, researchers explored how bending [Kildal 13, Lahey 11] and folding [Gallant 08, Khalilbeigi 12] interactions, supported by deformable mobile devices, can be used as physical controls. Compared to traditional physical controls, however, these controls lack affordances in that it is often unclear how folds and bends map to actions [Lahey 11]. In contrast, Paddle transforms to various physical controls and thus combines the flexibility of touch screens with the physical qualities of real world controls. As a side effect, Paddle also bridges the gap between differently shaped mobile devices, such as phones, tablets and wristbands.

In the following, we first give an overview of the system. Next, our interaction design space elaborates on the design choices when designing for transformable devices. We then provide details on the mechanical engineering and implementation of the system. Afterwards, two controlled experiments

report on the differences between interacting using the physical controls supported by Paddle and traditional direct touch input. We end with a discussion and summary of the presented work.

## 6.2 Brief System Overview

The mechanical construction of our prototype, Paddle, is based on principles used in the design of the Rubik's magic puzzle, a folding plate puzzle (Figure 6.7). Tiny infrared reflective markers are attached to this device and tracked with an optical tracking system. This enables us to project on the device to provide visual output from a ceiling mounted projector. Similarly, touch input is enabled by tracking the fingers of the user. In the future however, we envision Paddle devices to be self-contained using electronic connectors to track the topology of the device [Gorbet 98] and tiny integrated displays [Merrill 07], as those used in Tilt displays [Alexander 12] and Facet [Lyons 12], for visual output. In contrast to Paddle, earlier prototypes of transformable mobile devices [Gallant 08, Khalilbeigi 12, Lahey 11] are paper-like and thus support an origami transformation model that often requires numerous transformations. As Paddle is based on engineering principles used in the design of 3D puzzles, the transformation model is superior to origami and allows switching between different shapes in only a few steps (e.g. two steps to change a small flat form factor (Figure 6.1a) into a ring (Figure 6.1b)).

Similar to the system developed by Darliri et al. [Daliri 16], Paddle provides visual help cues to communicate transformation possibilities and guide the user (Figure 6.4a,e,g). These cues are only displayed when the user places his fingers at predefined spots, visualized by virtual fingers (Figure 6.4b,d). This approach (inspired by TouchGhosts [Vanacken 08]) reduces visual clutter and shows how to hold Paddle to perform different transformations. On the other hand, users can always choose to perform transformations that are not supported by the current form factor (Figure 6.3). In these situations, help cues are provided to backtrack to a supported configuration.

Figure 6.4 shows an example interaction that illustrates how Paddle can be used. (a) Adam gets a call from his close friend John on his Paddle, to see if he is in for a hike this week. (b-c) Adam answers the call by unfolding the compact device to a flip phone, which is more comfortable to hold while calling. (d) After a short chat Adam quickly transforms the device into a ring shape to scroll through the weather forecast of the next days. (e-f) On Wednesday and Thursday the weather looks perfect for a hike and Adam transforms the device into an agenda book through which he can leaf to see when he is available.

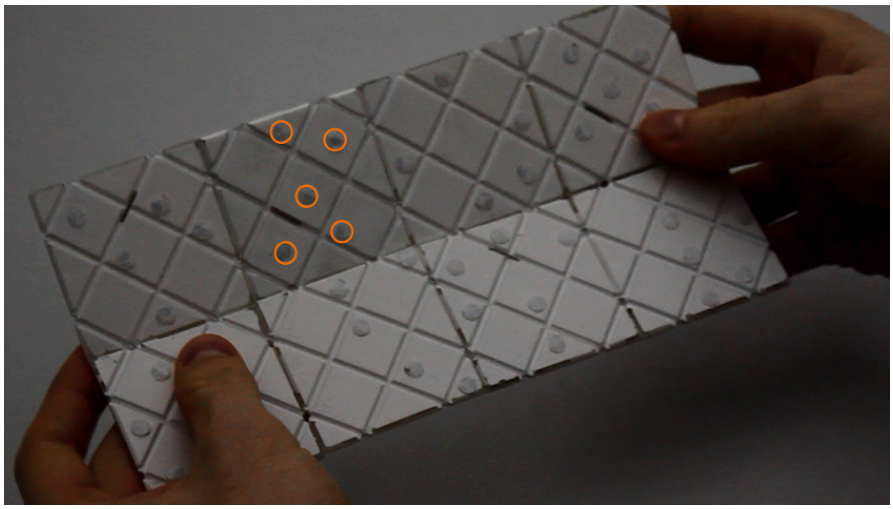

**Figure 6.2:** Tiny infrared reflective markers (highlighted in orange) are attached to the device.

(g) Both agree to hike on Tuesday and Adam unfolds his Paddle to see his schedule for that day. (h) Meanwhile, Adam can peek to check the status of the call or look for incoming emails. (i) Adam notices that he is only available in the afternoon and adds the appointment. (k) He unfolds his Paddle to a map on which he can plan a hike through the woods.(j) At the end of the call, John and Adam say goodbye and Adam folds his Paddle back in a compact form to fit inside his pocket.

## 6.3 Interaction Design Space of Transformable Devices

Transformable mobile devices provide various new opportunities for interaction design. We categorized these novel opportunities for interaction design in a design space. Transformable devices can also be considered in design spaces other than interaction design, such as design spaces elaborating on engineering techniques and principles (e.g. shape resolution [Roudaut 13]). In our interaction design space, we identified two dimensions: the *initiative* to transform and the *intent* of the transformation.

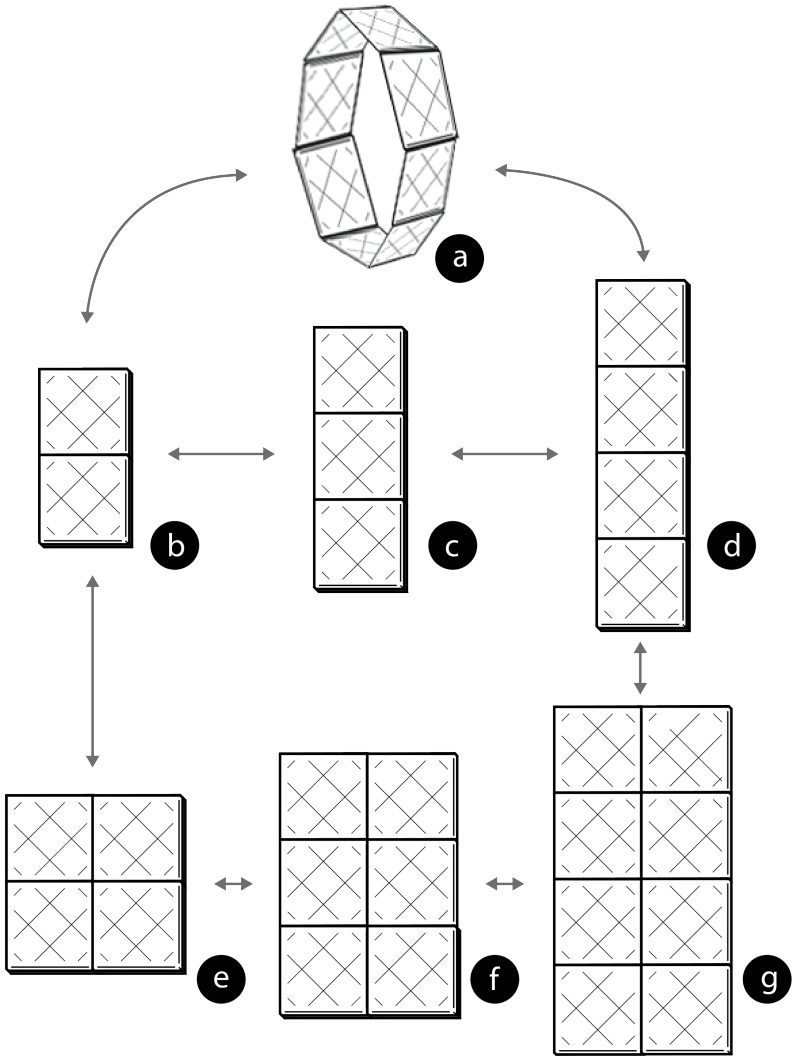


**Figure 6.3:** The transformations that are supported by Paddle. This is only a subset of the transformation model of the Rubik's magic puzzle, which is at the basis of our prototype.

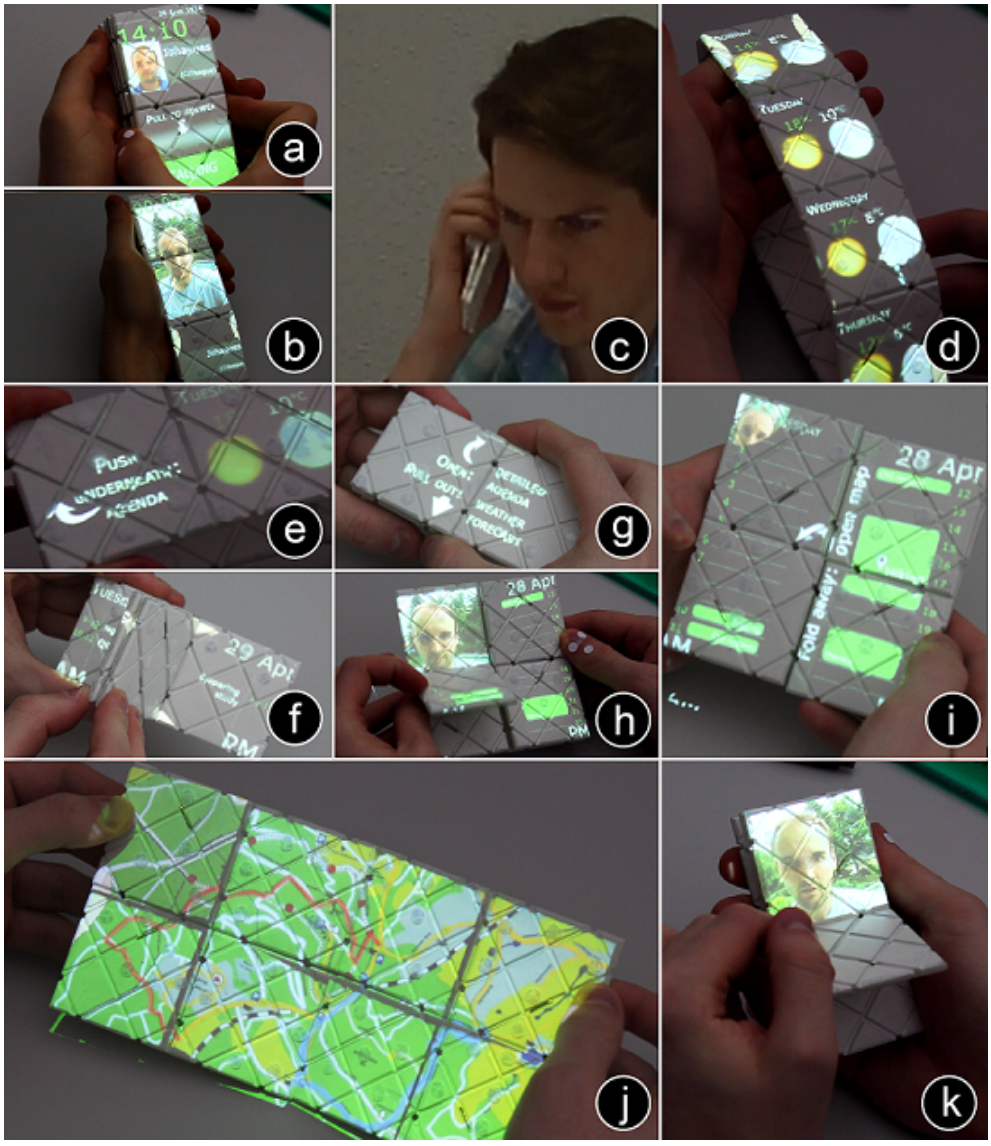


**Figure 6.4:** An example interaction that illustrates how Paddle could be used during a call.

### 6.3.1 Initiative to Transform

Either the user or system might take initiative to perform a transformation. For transformable devices that only transform manually, such as Paddle, users always initiate the transformation, even though the system might guide users by suggest transformation possibilities. The system interprets these types of user initiated transformations as input. We therefore consider this type of transformation as a novel input modality. In contrast, actuated transformable devices, such as Morphees [Roudaut 13] or MorePhone [Gomes 13] are initiated by the system to reveal output, such as incoming messages. Some systems (e.g. inFORM [Follmer 13]) support transformations as input (user initiated transformation) as well as output (system initiated transformations) modalities. Here it is important to note that although a transformation initiated by the system might be a direct response to an action of a user, we still consider the physical transformation to be initiated by the system. For example, the user presses a button after which the system initiates a transformation of the device.

### 6.3.2 Intent of the Transformation

The wide variety of transformations (both user and system initiated), supported by transformable devices, can serve different purposes. We identified three main intents for the shape of a device to transform:

1. *Shape fits digital content:* As demonstrated in earlier prototypes of transformable devices [Khalilbeigi 12, Khalilbeigi 11], enlarging a device can reveal additional data or tools (Figure 6.5b). Alternatively, the same content can take up a larger space as a way to zoom-in on data (e.g. text or maps). Oftentimes, it is also desirable to have a semantical zoom, providing a more detailed or concise version of the same content when a device changes in size. Finally, the physical form or shape can reflect the digital content [Lee 08]. For example, in Figure 6.3 the tiles of configuration (a) can reflect different elements of a list, (c) has the size of a traditional phone, (e) fits a traditional portable game console (Figure 6.5a) and (g) matches the size of a sheet of paper.

2. *Shape fits interaction style (ergonomics):* Interacting with a physical shape also relates to the comfort that it provides. For example, it is often more comfortable to read text on a large screen (e.g. Figure 6.3g). Likewise, shapes that are easy to support by the palm of the hand provide

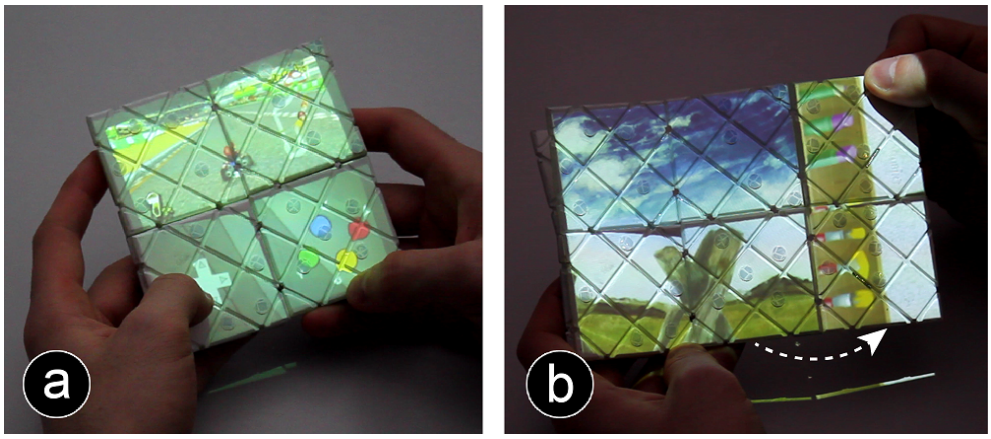


**Figure 6.5:** Transformations supported by Paddle can be used for many purposes, for example, (a) a game controller, (b) opening a toolpalet.

more comfort during touch interactions (more normal force [Dijkstra 11]). Other shapes make a larger region accessible for thumb interactions or position the microphone and speaker closer to the mouth and ear to improve voice and sound quality during calls[1].

Some shapes might also offer affordances that better suit certain interaction styles. Manipulating the angle of the extra panel in Figure 6.5b could control the value of a slider, as demonstrated in the FoldMe system [Khalilbeigi 12]. Figure 6.6 shows some configurations of the Paddle prototype that offer affordances and interaction styles that map particularly well to the task at hand, including peeking, scrolling, and leafing. In this last configuration, pages continuously flip from one side to the other to simulate an unlimited number of pages. Figure 6.3 shows these three interactions styles within the Paddle transformation model: configurations (b) and (e) support peeking (Figure 6.6a1), configuration (a) supports scrolling (Figure 6.6b1) and configuration (b) also supports leafing (Figure 6.6c1).

3. *Configuration fits social setting:* A device can also transform to accommodate the social setting that it is used in. MorePhone [Gomes 13] shows how lifting the corner of a device provides a subtle notification for incoming messages without disturbing others. Alternatively, peeking at

[1] http://www.lgnewsroom.com/newsroom/contents/63988

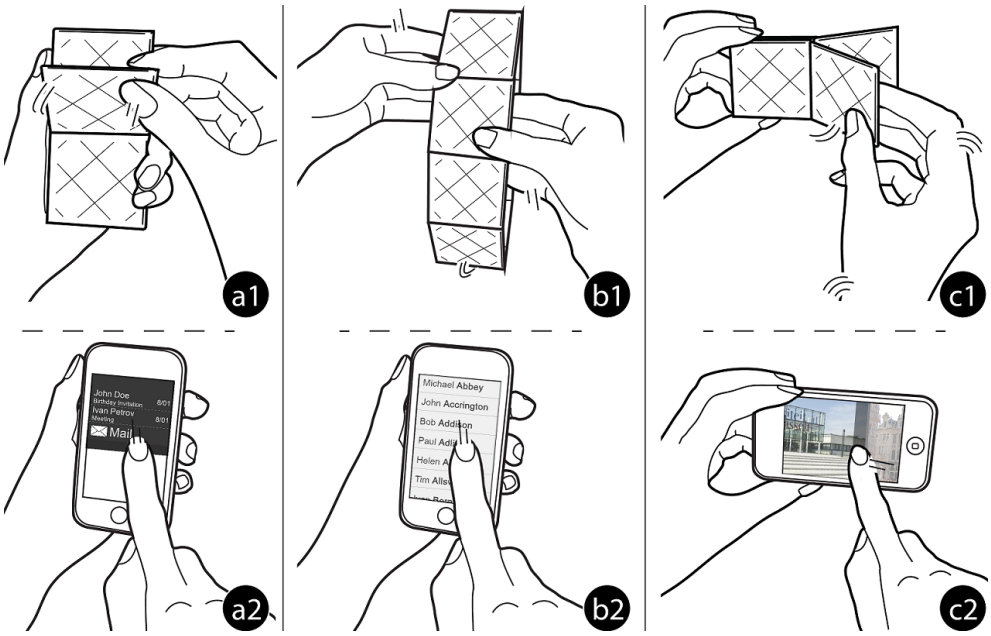


**Figure 6.6:** Paddle shows how physical controls for (a1) peeking, (b1) scrolling and (c1) leafing can be brought to mobile devices as alternatives for traditional touch interactions (a2,b2,c2).

content with Paddle, using the configuration shown in Figure 6.6a1, can reveal private content to mitigate shoulder surfing. Likewise, making calls using Paddle's phone configuration (Figure 6.4c), brings the microphone and speaker closer to the mouth and ear to have a more private conversation.

In many situations, the intent of a transformation is not mutually exclusive. For example, one might transform Paddle to the flat shape shown in Figure 6.3g to display a full page of text (shape fits digital content) and enjoy better reading comfort [Jabr 13] (shape fits interaction style).

## 6.4 Engineering and Implementation

### 6.4.1 Mechanical Construction

While transformable devices are not commercially available as yet, we aim at prototyping a mobile device that provide similar interaction styles. Our prototype, Paddle, is based on engineering principles used in creating 3D puzzles. 3D puzzles have already existed for centuries[2], thus capturing a vast amount of design knowledge in constructing compact mechanisms to enable complex transformations.

Our prototype leverages engineering principles of the Rubik's Magic design (Figure 6.7), a folding plate puzzle. The original goal of the puzzle is to transform the tiles until the pictures on the different tiles all line up and form an interconnection of rings (Figure 6.7a,b).

The design of the puzzle consists of a loop of square sized tiles held together by wires. A special wiring technique, where every wire runs diagonally across the tiles, ensures that all tiles can hinge along two adjacent sides. The location of these hinges is different when a tile is on top or underneath another tile. This technique is an extension of the Jacobs Ladder folk toy principle. Although the puzzle only transforms piece-wise, the special engineering principles used in the hinges allow for transforming the puzzle to various shapes (Figure 6.7). Figure 6.8 shows how the wires flip between the tiles to move a hinge from one side of a tile to another adjacent side, when going from a flat form factor (Figure 6.3g) to a ring (Figure 6.3a).

[2]Scientific American, Volume 16 Nr. 15, 1889, page 227

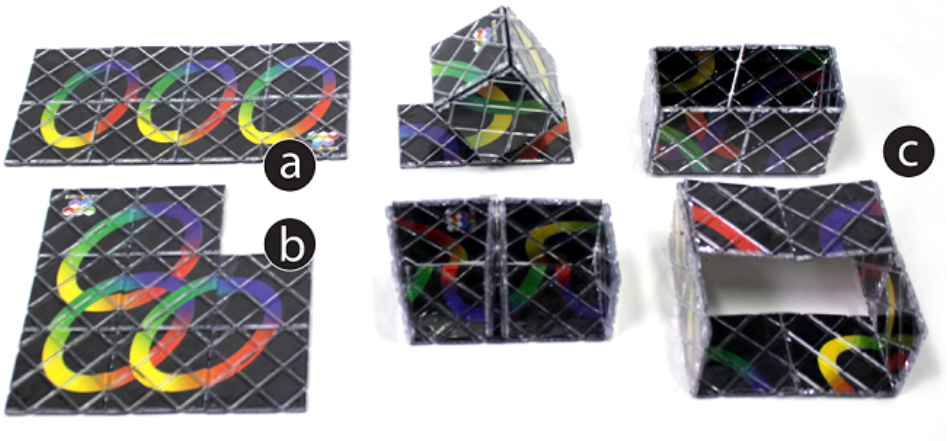


**Figure 6.7:** Paddle leverages engineering principles of the Rubik's Magic puzzle: (a) start configuration and (b) end configuration of Rubik's Magic, (c) some of the other supported shapes.

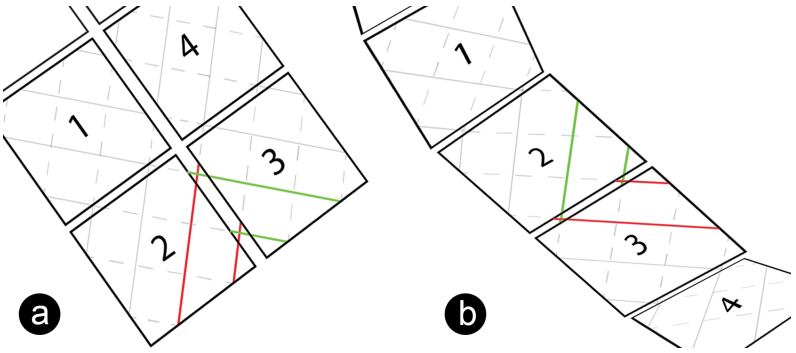


**Figure 6.8:** The wiring pattern used for the hinges in Paddle: (a) flat form factor, (b) ring form factor.

**Figure 6.9:** Paddle uses an eight-camera OptiTrack system to track the topology of the device and a ceiling projector to provide visual output.

### 6.4.2 Software Implementation

Paddle uses an eight-camera OptiTrack system to track the topology of the device and a ceiling projector to provide visual output (Figure 6.9). This makes the current version of the mobile device entirely passive. By attaching five tiny infrared reflective markers on both sides of every tile, we can track the position and rotation of every tile and distort every region of the projected user interface precisely (Figure 6.2). The Rubik's magic puzzle is painted white to make the projection clearly visible.

To enable touch interactions, markers are attached to the users' fingers. We calibrate every finger to get a precise estimate of the location of the markers on the fingers. When a finger touches our device, TUIO events are generated and translated into touch events using Multi-Touch Vista [3]. Our entire system is implemented in C#/WPF, making it possible to use the Microsoft Surface SDK[4] to have the same look-and-feel as traditional touch screens.

In the future, however, we envision devices similar to Paddle that are en-

[3] http://multitouchvista.codeplex.com/
[4] http://msdn.microsoft.com/en-us/library/ff727815.aspx

tirely self-contained, using electronic connectors to track the topology of the device [Gorbet 98] and tiny integrated displays [Alexander 12, Lyons 12] for output. The wires necessary for interconnecting these displays and sensors can replace the wires used in the hinges to not interfere with transformation possibilities.

## 6.5 User Study 1: Physical Controls vs Direct Touch

Although there are different intents to transform Paddle to various configurations (Section 6.3.2), Paddle's ability to transform in real-time to different physical configurations (shape fits interaction style) captures the main motivation of the work in this thesis (G2).

The physical control supported by the Paddle prototype are different from traditional tangible controls [Ishii 97, Fitzmaurice 95]. Similar to direct touch, the Paddle combines input and output in a single space, making input indistinguishable from its graphical output. In contrast, manipulating traditional tangible controls are perceptually coupled to their digital counterpart [Ullmer 00] which often reduces the feeling of engagement with digital content [Hancock 09, Terrenghi 08]. Although direct touch and physical controls supported by Paddle both offer clear affordances, the interaction styles they adopt are very different: direct touch relies on pointing as sole interaction technique whereas various grasps are used to interact with the physical controls supported by Paddle.

In a first experiment, we investigate the differences between using physical controls supported by Paddle, and traditional direct touch input typically used on mobile devices. We picked *peeking*, *scrolling* and *leafing*, as they are good representatives of physical controls supported by our prototype and equivalent interactions are supported on many touch screens (Figure 6.6).

### 6.5.1 Task Designs

Similar to earlier studies [Buxton 86, Hancock 09], we compare the interaction styles by considering both physical skills as well as the effects on the human congnition. Our task designs are based on best practices for measuring differences in skill and motor behavior [Schmidt 88]. Various pilot studies were conducted to calibrate the difficulty of the task sets.

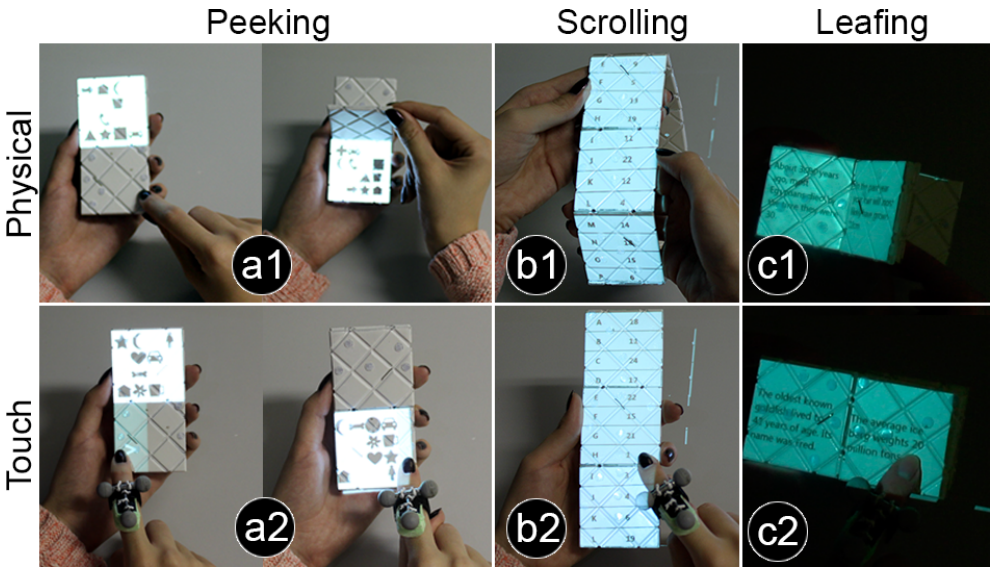


**Figure 6.10:** Our first study compares (1) physical to (2) touch interactions for (a) peeking, (b) scrolling and (c) leafing tasks.

### Peeking

For this task, 9 different symbols (out of a set of 17 different symbols e.g. a circle, triangle, star) are displayed at random positions in a grid (Figure 6.10a). When performing a peeking interaction, all 9 symbols plus an additional symbol become visible at random locations on the window that is revealed (Figure 6.10a). The original window then becomes invisible to encourage more peeking interactions. The task is completed when the user identifies the additional symbol. In the physical condition, the peeking window is located on the upper side of the device, while in the touch condition, the peeking window can be opened by dragging the window from left to right which reduces the effect of occlusions caused by the hand. While the participant finds the missing symbol, we measure *task completion time* and *number of peeks* as measures of performance.

### Scrolling

The main goal of the scrolling task is to encode a given word into a series of digits, as fast as possible, using the encoding list displayed on the device (Figure 6.10b). The user can scroll through a look-up table that has 26 entries,

one for every letter in the alphabet, using Paddle. Four entries of the encoding table are displayed on every tile of Paddle, leaving 1.5 tiles empty between the end and the starting point of the loop (Figure 6.10b1). In the touch condition (Figure 6.10b2), the loop is simulated, making it possible to scroll infinitely, similar to the physical condition. Different encoding tables are used for every word. Although words with the same length are used, different letter combinations require different amounts of scrolling. We compensate for this by generating encoding tables that requires the same amount of scrolling for every condition. For this task, the *task completion time* is used as the main measure of performance.

#### Leafing

For the leafing interaction, participants leaf through a book of 10 pages in which facts about various topics are shown on every page (Figure 6.10c). Participants leaf through the book and read the facts for 70 seconds. Afterwards, 4 facts are presented, but some of the facts have their details changed. Participant are asked if the fact is correct (distractor task) and have to estimate the page number of the fact in the book (main task). For this task, the *error rate* for estimating page numbers and the *success rate* (percentage) for detecting correct facts was measured.

### 6.5.2 Study Procedure

We recruited 16 participants (2 female, mean age 25) from our university campus. All were right-handed. The study used a within subjects design. For each of the three tasks, the physical and touch condition, were presented in a counterbalanced order. Additionally, the task sets were switched over the conditions to balance potential differences in task difficulty.

For every condition (physical, touch), the participant received instructions about how to perform the task. For the peeking and scrolling task, participants were instructed to complete the task as fast as possible. Since both tasks are feasible with a very low error rate (confirmed by the pilot studies) under all conditions, a new task was given in the rare occasions when participants failed to give the right answer. For the leafing task, we instructed participants to focus on the distractor task (correctly identifying wrong facts). Participants controlled a footswitch to start and end every task. Every condition started with a series of practice trials until the participant understood the interaction style and felt comfortable with the system. For the touch conditions, the

participant's pointing finger was instrumented with markers to enable touch interaction on Paddle.

After every condition, a modified version of the ISO 9241-9 questionnaire with dependent rating scale for testing input devices was filled in, with only a single fatigue category and an additional "intuitiveness" category. All interactions were recorded on video. The average experiment completion time was approximately one hour. After a participant completed the experiment, there was an informal discussion during which the participant was asked to explain his interaction preferences.

### 6.5.3 Hypotheses

We expect that peeking using touch requires participants to focus on both the content in the peeking window and how the entire peeking window is moving when it is opened or closed. When using physical peeking, on the other hand, participants rely on tactile sensory feedback to interact with the peeking window and thus better focus on the visual information that is presented (i.e. our symbol comparison task). We therefore hypothesize:

**H1** : Physical peeking is faster than peeking using touch.

We expect that physical scrolling helps users to build up spatial memory and better uses sensory motor skills compared to touch. We therefore think that for physical scrolling, participants will immediately grasp at elements along the ring instead of scrolling through all elements, as is necessary with touch. We therefore hypothesize:

**H2** : Physical scrolling is faster than scrolling using touch.

Physical leafing with Paddle is a more expressive way to navigate through a book compared to swipes using touch. We therefore expect that with physical leafing, participants are more aware of the position of the current page in the book at any time and are thus better at recalling the structure of this book. We therefore hypothesize:

**H3** : Physical leafing results in more accurate estimates of page numbers.

### 6.5.4 Results

#### Quantitative Results

We collected 384 data points for the peeking tasks, 384 data points for the scrolling tasks and 512 data points for the leafing tasks. After removing error

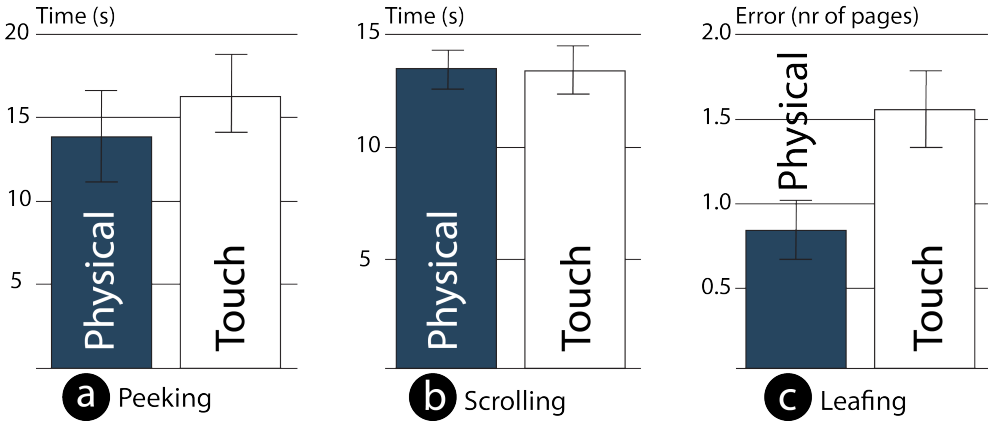


**Figure 6.11:** The results of study 1 comparing physical and touch for (a) peeking, (b) scrolling, (c) leafing.

trials (respectively 3.1%, 2.1% and 0.2%) and outlier response times for the peeking and scrolling task (respectively 1.6% and 1.6%), we were left with respectively 366, 370 and 511 trials in this analysis. Trials were labeled as outliers for each condition when exceeding the mean by three standard deviations.

For every task, we aggregated the trials of every participant, and ran a repeated-measure ANOVA. For the peeking task, we found a significant difference between physical and touch (13.9s vs. 16.5s, $F_{1,15} = 5.75$, $p$=0.03), as shown in Figure 6.11a. This confirms H1. Contrary to H2, we did not find a significant difference between the scrolling conditions ($p$=0.93) (Figure 6.11b). As shown in Figure 6.11c, we found a significant difference between physical leafing and leafing using touch, with the physical condition resulting in 46% more accurate page estimates ($F_{1,15}$ =62.97, $p$<0.001). This confirms H3. We also analyzed that physically leafing through pages resulted in significantly fewer errors when participants were asked if the presented fact was correct (73.3% vs. 57.4%, $F_{1,15} = 25.79$, $p$<0.001). Additionally, participants leafed significantly more in the physical condition (15.2 vs. 12.8 leafs, $F_{1,15} = 7.26$, $p$=0.02).

### Qualitative Results

The comparative ISO 9241-9 questionnaire confirmed our quantitative results for the peeking interaction (H1). Participants reported that they found physical peeking significantly better on the scale of force, intuitiveness, accuracy,

speed, comfort, smoothness and overall operation. Physical peeking performed equally well as peeking using touch on the scales of effort and fatigue. Some noticeable comments during the interview also help to further explain our quantitative results: "When using touch, I have the feeling that my eyes are following my finger unconsciously.", "In the physical condition, you intuitively know how far to open the peeking window to reveal all content. In the touch condition, the peeking window can have any size.", "In the physical condition, the content does not move.", "Touch requires better targeting as one needs to stop at the border of the device to prevent losing track of the touch point."

Similar to the quantitative data, there was no clear preference in interaction style for scrolling (H2). However, participants agreed that physical scrolling requires significantly more effort than scrolling using touch. Contrary to H2, five participants mentioned during the interview that they did not know how many tiles to scroll with the physical ring to go to a specific letter. Two of these participants commented that they were confused because multiple items were displayed on a single tile. Four other participants mentioned that their performance in the physical condition could have been influenced by the latency of our system. This latency was much more noticeable when physically scrolling through the list, because the device moved a lot more.

Contrary to the quantitative results for the leafing task (H3), participants rated leafing using touch significantly better on the scales of force, effort, speed, comfort and smoothness. Physical leafing performed equally well as leafing using touch on the scales of intuitiveness, accuracy, overall operation and fatigue. When participants were asked during the interview which interaction style they preferred overall for the leafing task, the ratings of the questionnaires were confirmed, with 11 participants preferring touch, 3 preferring physical and 2 undecided. However, six of the participants who preferred touch, mentioned that they would prefer physical leafing if the leafing mechanism was more comfortable. Many of these participants commented that when leafing physically, the device was less comfortable to hold as pages have to flip back from behind in order to have a book with an unlimited number of pages. One participant mentioned that he felt more confident about the page numbers he gave when physically leafing through the book.

### 6.5.5 Study Discussion

Overall, hypotheses H1 and H3 were confirmed. Our study clearly shows that physicality provides benefits for leafing interactions. Participants could better recall the structure and content when physically leafing through the book.

Although we also measured a significant benefit for physical peeking compared to touch, it remains unclear if the direction for opening the peeking window in the touch condition played a significant role in the effect that we measured. As one participant correctly noticed, the direction in which the peeking window had to be pulled out in the touch condition might have slowed him down, because the end point of the dragging movement was near the border of the device, which introduces a speed-accuracy trade-off, known as Fitts' law. Additionally, our experiment did not confirm our second hypothesis, in that we did not measure a difference between physical scrolling and scrolling using touch.

We identified four factors that require further investigation in a second experiment, in order get a deeper understanding of the benefits of physical controls supported by Paddle. Three factors focus on scrolling, for which we could not yet find any significant difference. The last factor focuses on peeking, for which we already found significant differences.

1. *Does the mapping of digital items to physical elements contribute to the efficiency of physical scrolling?* We observed that displaying multiple digital items (in our case 4 letters of the alphabet) on a single physical element (in our case a single tile of the physical ring) introduces a mismatch between tactile and visual information and confused participants, as they were not used to think in terms of groups of 4 items. To investigate this, we will compare physical scrolling to scrolling using touch with a list of 8 elements, one for every tile in the physical ring (Figure 6.12b1-b2).

2. *Does visual realism contribute to the efficiency of physical scrolling?* A lack of visual realism in the physical scrolling condition, due to the latency of our prototype, might have influenced the performance of the physical scrolling condition more than the touch condition, as physical scrolling requires more and faster movements of the device. To measure this effect, we will compare a physical ring with 8 elements printed on it, referred to as the paper prototype (Figure 6.12b3), to the condition where 8 elements are projected on the ring (Figure 6.12b2).

3. *Does the quality of scroll inertia contribute to the efficiency of scrolling using touch?* Although inertia was enabled on the scroll list in the touch condition, our prototype implementation could have influenced the performance of scroll inertia, as we tracked the position of the fingertip to the surface of our device, which can slightly vary over different hand/finger postures. To measure this effect, we will compare scrolling using

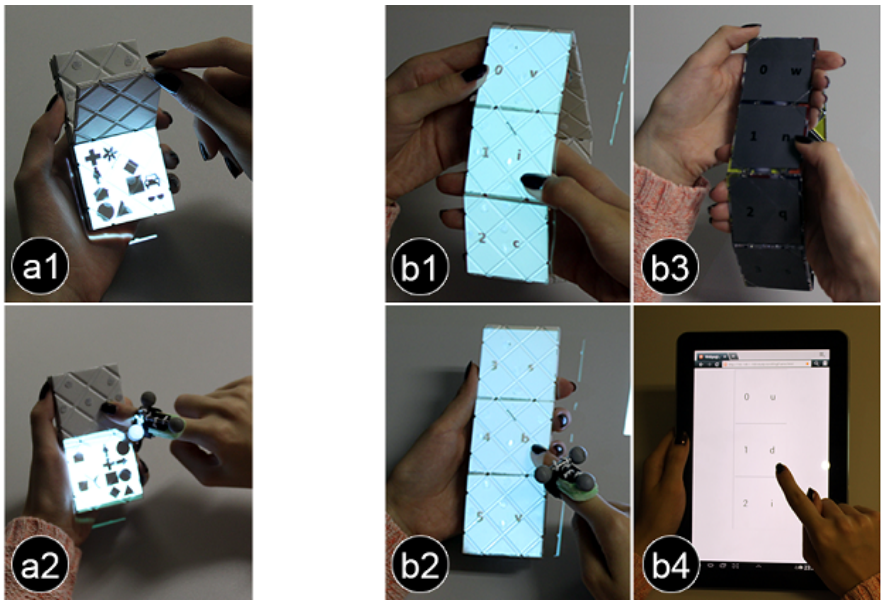


**Figure 6.12:** Our second study further investigates the factors contributing to the efficiency of physical controls supported by Paddle and touch interaction for peeking and scrolling.

touch on Paddle (Figure 6.12b2) to scrolling using touch on a tablet on which exactly the same list is displayed (Figure 6.12b4).

4. *Does homing towards a target near the border of a device when opening a peeking window, contribute to the efficiency of peeking using touch?* For this, we will evaluate touch and physical peeking in another direction, which eliminates precise targeting and similar to the previous experiment, reduces the effects of occlusion by the hand to a minimum (Figure 6.12a).

## 6.6 User Study 2: Contributing Factors

To understand the factors contributing to the differences between physical and touch conditions for both our peeking and scrolling interaction, we conducted a second study. We evaluate the relative effect of the location of the target position when opening a peeking window using touch by evaluating peeking in another direction (Figure 6.12a). For the scrolling task, we investigate the

importance of mapping digital to physical items (Figure 6.12a-b), the potential effects of visual realism for the physical condition (Figure 6.12d) and scroll inertia for the touch condition (Figure 6.12c).

### 6.6.1 Task Designs

For the peeking conditions, we used task sets similar to the ones used in our first experiment, but we displayed them on different regions of Paddle (Figure 6.12a). For the scrolling conditions, the task sets were slightly adapted, as we now have lists consisting of only 8 elements (compared to 26 in the first study). We therefore generated new look-up tables, which translate 8 ordered numbers (0-7) to letters. The resulting word is nonexistent, which prevents participants of giving the answer before looking up all the numbers. Participants were also explicitly instructed to look up the numbers in the order in which they were displayed. We also increased the number of items that participants had to look up, from 4 in the previous study to 5, in order to encourage more scrolling and compensate for the shorter look-up tables.

### 6.6.2 Study Procedure

We recruited 16 participants (4 female, mean age 26), 8 of which were randomly chosen from our previous pool of participants in study 1. All were right-handed. The study used a within subjects design. For the peeking task, the conditions were fully counterbalanced. The four scrolling conditions were presented in a counterbalanced order using a balanced Latin square. Additionally, the task sets were counterbalanced over the conditions for every task, in order to balance potential differences in task difficulty. The entire experiment lasted one hour on average, after which there was an informal discussion with the participant.

### 6.6.3 Hypotheses

**H4** : Physical peeking outperforms peeking using touch, independent of the location of the peeking window.

**H5** : Physical scrolling outperforms scrolling using touch when displaying exactly one digital element on every tile.

**H6** : Physical scrolling with the paper prototype outperforms physical scrolling with Paddle using projection.

**H7** : Scrolling on the tablet outperforms scrolling on Paddle using touch.

**H8** : Physically scrolling with the paper prototype outperforms scrolling on the tablet.

### 6.6.4 Results

#### Quantitative Results

We collected 384 data points for the peeking task and 768 data points for the scrolling task. After removing error trials (respectively 3.6% and 2.6%) and outliers (respectively 2.1% and 0.7%), we were left with 362 and 742 trials, respectively. We analyzed the results using repeated measures ANOVA. All post hoc comparisons used Bonferroni corrected confidence intervals.

Consistent with the findings of our first study, we found a significant difference between physical peeking and peeking using touch as shown in Figure 6.13a (13.4s vs. 16.4s, $F_{1,15} = 5.45$, $p$=0.03), confirming H4.

Figure 6.13b shows the overall trend of our results for the scrolling tasks. There was a significant main effect ($F_{3,45} = 50.5$, $p$<0.001). Pairwise tests show that participants were significantly faster with physical scrolling than scrolling using touch on Paddle (11% faster, $p$=0.02). Furthermore, physically scrolling with the paper prototype outperformed physical scrolling with projection (20% faster, $p$<0.001). Scrolling on the tablet was significantly faster than scrolling on Paddle using touch (22% faster, $p$<0.001). Finally, physically scrolling with the paper prototype was significantly faster than scrolling using touch on the tablet (9% faster, $p$=0.01), confirming H5-H8.

#### Qualitative Results

Although all participants agreed that physical scrolling was more tiring than touch, 11 participants preferred physical scrolling with the paper prototype and only 3 participants preferred the tablet (2 participants were undecided). 13 participants agreed that they could easily remember the position of elements along the physical ring. Five participants even noticed that overshooting targets occurred more often on the tablet than when physically scrolling with the paper prototype.

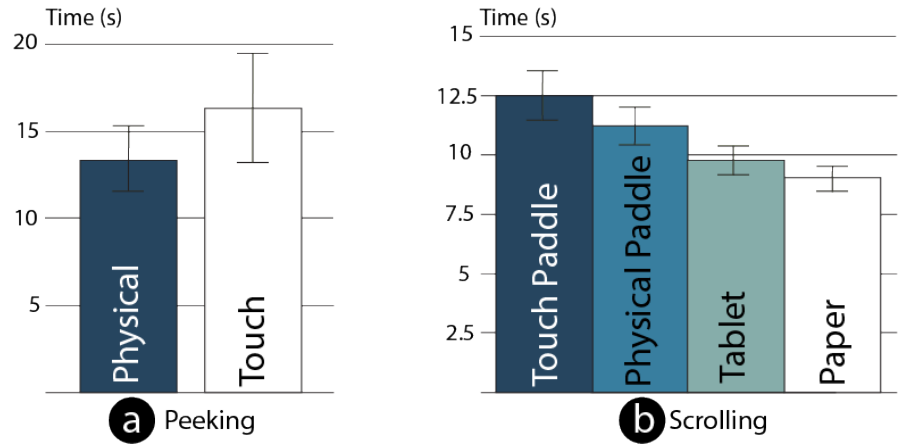


**Figure 6.13:** The results of study 2, investigating the factors contributing to the differences between physical and touch for (a) peeking and (b) scrolling.

## 6.7 Findings and Design Recommendations

The results of our two studies show that physical controls supported by Paddle have certain benefits compared to traditional direct touch interaction techniques.

Starting with the peeking interaction technique, our first and second study reveal that physical peeking is significantly faster and subjectively preferred compared to peeking using touch (H1+H4). We therefore conclude that physical peeking outperforms peeking using touch. Our results suggest that peeking using touch requires users to focus on both the content in the peeking window and how the window is moving. When using physical peeking, on the other hand, users rely mostly on tactile feedback to interact with the peeking window and can therefore better focus on the visual information presented in the task at hand.

For the leafing interaction, the qualitative analysis of our first study shows that physical leafing is a more tiring interaction compared to touch, but results in an improved recall of the structure and content of a book (H3). Similar results have been found in studies on paper vs. e-readers [Jabr 13]. Their results were, however, motivated by the fact that paper books facilitate the process of picturing content on pages, as real books provide implicit cues (e.g. thickness) about where you are in the book. Our finding suggests that the physical leafing interaction itself is a large contributing factor to the benefits

one gets from interacting with real books. So in addition to all efforts of making e-readers look and feel like real paper [Watanabe 08, Wightman 11], Paddle shows how certain benefits one experiences with real books can be brought to devices using physical controls.

For the scrolling interactions, our first study indicates that physical scrolling is more tiring than scrolling using touch. No other significant differences between these input modalities were found (H2). Our second study, however, shows that when displaying exactly one item of a list on every tile of the physical ring, participants were significantly faster with physical scrolling than scrolling using touch (H5). In the physical scrolling condition, users could locate elements very precisely and are thus able to directly grasp at elements further in the ring compared to the touch condition, in which scrolling through all elements is necessary. This suggests that users are better able to use their sensory motor skills and build up spatial memory when scrolling physically.

Our second study also shows that visual realism plays an important role when physically scrolling through lists. Physical scrolling with the paper prototype turns out to be significantly faster than both physical scrolling with Paddle with projection (H6) and a traditional tablet with scroll inertia enabled (H8). The latter result suggests that scrolling with future Paddle devices with perfect visual fidelity (i.e. with real displays) will result in faster acquisition of elements compared to traditional lists on touch screens with scroll inertia.

### 6.7.1 Physical scrolling through longer lists

Our first and second study only show significant benefits for physical scrolling when mapping every item of a list to a single physical element of our ring (i.e. same number of digital and physical items). It remains unclear how to efficiently support longer scrolling lists with Paddle. We therefore conduct a smaller final study in which we compare physical scrolling through longer lists using Paddle to scrolling through the same list on the tablet. The same tasks are used as in the second study, but now the look-up tables are twice as long. Instead of displaying multiple items of the scrolling list on a single physical element at the same time, as we did in the first study, we now wrap the scrolling list multiple times around the physical ring. We asked 8 participants to perform both conditions in a counterbalanced order. Our analysis is based on 185 valid trials.

Although our second study shows that scrolling using touch on the tablet was significantly faster than physically scrolling with projection, we did not find a significant difference between these two conditions when the list is twice

as long ($p$=0.79). These results suggest that participants gained more benefits from physicality when the lists become longer. While users commented that physical scrolling is more tiring with longer lists, analyzing our video recordings clearly shows the dexterity that participants have when scrolling physically through the long lists. Even though participants noticed the latency of our system when physically scrolling using Paddle as in study 1 and 2, participants knew almost exactly how many times to turn the ring to get to a specific item. Our second study shows that visual realism is an important contributing factor for the efficiency of physical scrolling with Paddle. We therefore expect that for longer scrolling lists, future Paddle devices with perfect visual fidelity (i.e. with real displays), will also outperform direct touch.

### 6.7.2 Design Recommendations

The findings from our studies suggest the following recommendations for the design of digitally augmented physical controls:

- *Optimize the physical manipulation to match the task at hand:* Instead of mapping interactions from the physical to the digital world, interactions with well designed physical controls integrate feedback that is a natural consequence of the action. Our study findings demonstrate that when physical interactions are highly intuitive and require low conscious effort, users better focus on the task at hand.

- *Tactile information should correspond with visual content:* When a physical control provides tactile guidance during interactions, this tactile information should correspond with the visual information that is embodied. The physical scrolling condition demonstrates that only if the tactile information corresponds with the digital content (one item on every time), users benefit from these tactile cues to process information. When the tactile and visual content do not correspond, additional cognitive processing is required to map the different sources of information.

- *Physical material should integrate high-fidelity visual information:* Physical controls often allow for fast interactions as tactile information can be processed quickly. However, when the sensors to recognize the interaction or communicate feedback introduce errors and latency, users have to perform additional cognitive steps to verify the content.

## 6.8 Discussion

Physical controls supported by Paddle differ from traditional tangible controls [Ishii 97, Fitzmaurice 95] in two ways: First, Paddle transforms to different physical controls over time. Multiple controls are therefore not available at the same time. Instead of being inherently space-multiplexed, as traditional tangible systems [Fitzmaurice 95], interactions using Paddle's physical controls are time-multiplexed. Secondly, Paddle makes input indistinguishable from its graphical output, integrating them in a single space. In contrast, traditional tangible controls often requires users to map interactions from the physical to the digital world as physical controls are perceptually coupled to digital content [Ullmer 00]. As such, physical controls supported by Paddle provide *inherent feedback*, where feedback is a natural consequence of an action [Djajadiningrat 02].

Besides Paddle's inherent strength to bring various physical controls into a single mobile form factor, our user experiments investigate the difference between these controls and traditional direct touch interactions. Although these experiments initially use quantitative measures, such as speed and success rate as the main measures of performance, our final results revealed substantial differences between the interaction styles we considered including. Notable results include, able to better utilize sensory motor skills for the physical peeking, improved recall of structure and content for the physical leafing interaction, and improved abilities to use spatial memory for the physical scrolling interaction.

This work is also subject to two limitations. First, the transformation model supported by Paddle is unknown to the user, making it challenging to switch between form factors. Furthermore, users can always perform transformations that are not supported in a given context. These problems can be alleviated by providing visual help cues [Bau 08, Vanacken 08] to show transformation possibilities. Although our scenario (Figure 6.4a,e,g) demonstrates this idea, more generic visualizations are needed in the future. Secondly, transforming a device can be time consuming, especially when interactions with a single form factor last for only a few seconds. In these situations, traditional touch interaction on one of the flat shapes supported by Paddle can be used at the expense of the benefits that the custom shape would provide. Alternatively, future Paddle interfaces can better support the training of users' muscle memory to switch between shapes in less than a second after some practice. We also see potential in integrating actuators, such as shape memory alloys [Roudaut 13, Hawkes 10] in our design to ease or even automate transformations. Roudaut et al. [Roudaut 16] recently presented an actuated hinge

mechanism that can turn along two adjacent sides, conform to our Paddle mechanical design.

## 6.9 Summary

In this chapter, we showed how the physical layout of interfaces can adapt in real-time (G3). More specifically, we explored techniques to embed materials in interfaces that allow for fast transformations in their physical state (C1). As such, a single physical form factor transforms to different physical shapes. We demonstrated this concept by prototyping one possible implementation in the context of mobile devices, called Paddle. Engineering concepts used in the design of Paddle are based on principles used in folding plate puzzles. Interfaces like Paddle trade the space-multiplexed input nature of traditional physical controls for flexibility. As such, they combine the flexibility of traditional direct touch interaction concepts with the qualities of real world controls. A usage scenario demonstrated various novel interaction concepts that are enabled by Paddle. Besides this, we provided a comprehensive interaction design space for transformable devices. The results of our two user studies show that although interactions with physical controls are more tiring, as compared to traditional touch interactions, they still provide some inherent benefits, such as being able to better utilize sensory motor skills, improved abilities to use spatial memory, and improved comprehension of structure and content. Although Paddle is still a prototype, we believe that our findings informs and encourages the design of future physical interfaces which adapt their physical layout, in real-time, to the task at hand.

# Chapter 7

## Discussion and Future Work

Each of the previous chapters offers a discussion of the presented concepts and systems. This chapter discusses aspects that bestrides these topics and covers future work that arises from this dissertation. We start by generalizing the research concepts at the intersection of digital fabrication and transformable interfaces. Next, we provide a deeper discussion on the different groups of users that benefit from our systems. Finally, we elaborate on the user-experience challenges that come with computationally generated solutions and discuss how physical user interfaces could seamlessly integrate in environments in the future.

### 7.1 Digital Fabrication and Transformable Interfaces

Digital fabrication systems employ robotic machinery, such as a multi-axis moving platform or robotic arm to add or subtract material permanently from the workpiece (Section 2.2). In contrast, transformable interfaces employ the user's muscles (Section 2.5.1) or integrate actuators inside the material (Section 2.5.2) to make temporary changes to the physical artifact. While many transformable interfaces change the shape of objects faster as compared to digital fabrication systems, the latter class has a much higher resolution to precisely shape materials. However in many ways, both concepts work towards a similar goal of applying computational power to the physical world. This trend towards a shared goal is exemplified by research in digital fabrication that focus on increasing the speed of fabrication machinery [Mueller 14b, Peng 16].

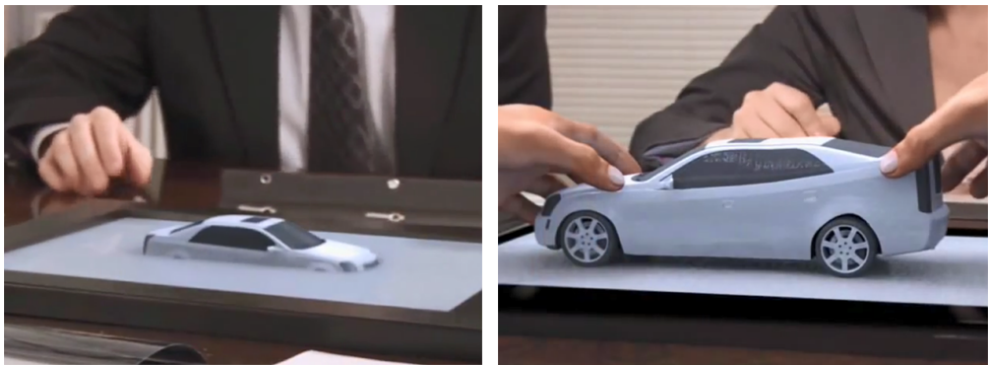

**Figure 7.1:** Claytronics[1]: creating and adapting physical objects on the fly.

Researchers in the field of transformable interfaces, on the other hand, often contribute new systems with an increased shape resolution [Follmer 13]. Figure 7.1 shows the Claytronics project[1] that envisions smart materials, called programmable matter, to exist in the future. With this kind of material it becomes possible to author the form-factor of physical objects on the fly. These kind of smart materials are often considered to be the ultimate form of transformable interfaces. However, the technology shown in Figure 7.1 could also be considered a very fast 3D printer.

Besides overlap in research goals, both tracks of research also influence each other. Recently, digital fabrication technologies are used to realize new forms of shape-change. HapticPrint [Torres 15] shows how to 3D print internal structures of objects that only deform along one axis. Metamaterials [Ion 16] advanced these techniques to realize 3D printed functional mechanisms, such as hinges, that do not require assembly.

As material science did not yet advance to the point of realizing programmable matter, this dissertation considers both digital fabrication as well as transformable interfaces to allow end-users to take control over physical interfaces. PaperPulse (Chapter 3) and RetroFab (Chapter 4) use digital fabrication technologies to allow end-users to either create new or repurpose existing physical interfaces. Paddle (Chapter 6) on the other hand, employs strategies from transformable interfaces to adapt physical interfaces in real-time.

[1] http://www.cs.cmu.edu/ claytronics

## 7.2 Target Audiences

The systems developed in this dissertation target people who are interested in making physical interfaces but do not have a background in electronics, programming, or design. This audience partially overlaps with the Do-It-Yourself (DIY) and Maker community, although a number of people in these communities already gained technical experience through workshops, tutorials, or courses [Qi 14, Mellis 13]. However, even for those users, our user studies revealed (Sections 3.8 and 4.8) that the presented systems speed-up and ease the process of making high-fidelity physical interfaces.

We consider the systems presented in this dissertation as "enabling technologies", as they enable new groups of users to realize interactive physical interfaces. There are however aspects that are traditionally considered while designing interfaces that go beyond making a deployable interactive system, foremost, the usability of the physical interface. The design environments in this dissertation do not encourage users to make an aesthetic or easy-to-use design. Instead, the environments rely on the users' instincts to make decent design decisions. Additional research is needed to realize algorithms that automatically suggest design decisions based on models and heuristics [Nielsen 94]. Similar automated usability approaches have been explored in the context of website design [Todi 16].

Although STEM (science, technology, engineering, and mathematics) education is oftentimes the main motivation to facilitate aspects in the process of making things e.g. LittleBits [Bdeir 12], the systems presented in this work do not educate users. Our systems in contrast, automate many aspects that normally require extensive training, including, complex manual material transformations, electronics, programming, and 2D and 3D design. Learning from automatically generated solutions is an interesting direction for future research, as users immediately start with a working solution and learn details later (bottom-up learning).

Interaction designers, such as product designers or boardgame designers could also benefit from PaperPulse (Chapter 3) or RetroFab (Chapter 4) while prototyping high-fidelity interactive interfaces. During the design process, professional designers often start with low-fidelity prototypes, such as sketches on paper, in software tools [Lin 00], or using low-fidelity physical toolkits [Hudson 06]. The current version of our design environments have no explicit support to automatically convert those early prototypes into PaperPulse or RetroFab designs. Nevertheless, users can benefit from the insights and lessons learned during earlier prototyping stages.

## 7.3 User-Experience of Computationally Generated Solutions

Computationally generated solutions, such as guided physical transformations with Paddle (Chapter 6) or generated circuit designs in PaperPulse (Chapter 3) or RetroFab (Chapter 4) allow for realizing complex solutions without requiring prior knowledge. On the flipside, when solutions fail as a result of, for example, short circuits or human errors, it is hard for the user to recover the system. In the context of the Paddle phone, this issue is resolved by providing help cues to backtrack the system to the initial physical configuration. In the PaperPulse and RetroFab design environment, the computationally generated circuit has various potential points of failure (e.g. disconnectivities in the circuit or failure of electronic components). More research is needed to assist users in locating and resolving these problems.

Some users also appreciate making design changes during the physical assembly process, especially when the artistic qualities of artifacts are important. Devendorf et. al [Devendorf 15] demonstrates how users can remain in control while the machine suggest a computationally generated solution. In PaperPulse and RetroFab, in contrast, all decisions are made while designing the artifact in the software environment. When making changes during the assembly phase, the correct working of the final physical interface is not guaranteed.

## 7.4 Seamlessly Integrated Physical Interfaces

Some of the physical user interfaces realized in this dissertation can be considered as bulky, such as the projector setup required for Paddle (Chapter 6), the 3D printed structure of Retrofit interfaces (Chapter 4), or the microcontroller integrated in PaperPulse designs (Chapter 3). The size of these setups can be reduced significantly by reducing the size of electronic components or the overall number of components in the system. Future versions of these systems can seamlessly integrate thin-film display [Olberding 14], thin-film batteries [Bates 00], or thin-film microcontroller [Myny 12]. Alternatively, the number of components in a system can be reduced by sharing electronic sensing components among multiple passive widgets as demonstrated in Sauron [Savage 13] and Touch&Activate [Ono 13].

# Chapter 8

# Conclusion

In this chapter, we discuss how we have answered the research goals and challenges that we have postulated in the introduction of this dissertation. Figure 8.1 summarizes how the research challenges, research goals, and contributions link to each other.

## 8.1 Addressing the Research Challenges

In Chapter 1, five research challenges were formulated to better support novices in author physical interfaces. In the following, we reflect on the various concepts presented in this dissertation which address these research challenges.

**(C1) Availability of machinery and knowledge required to author physical materials**
This work investigated a number of techniques and DIY machinery that make it possible for non-experts to author physical materials. PaperPulse (Chapter 3), uses a regular inkjet printer filled with conductive ink and a toolkit of components that enable novices to manufacture advanced electronic circuits on flexible substrates. This traditionally requires professional machinery as discussed in Section 2.1. RetroFab (Chapter 4), shows how add-ons for appliances are created with conventional 3D printers. Paddle (Chapter 6), demonstrates how transformable materials can be used to allow novices to adapt the shape of physical interfaces in real-time.

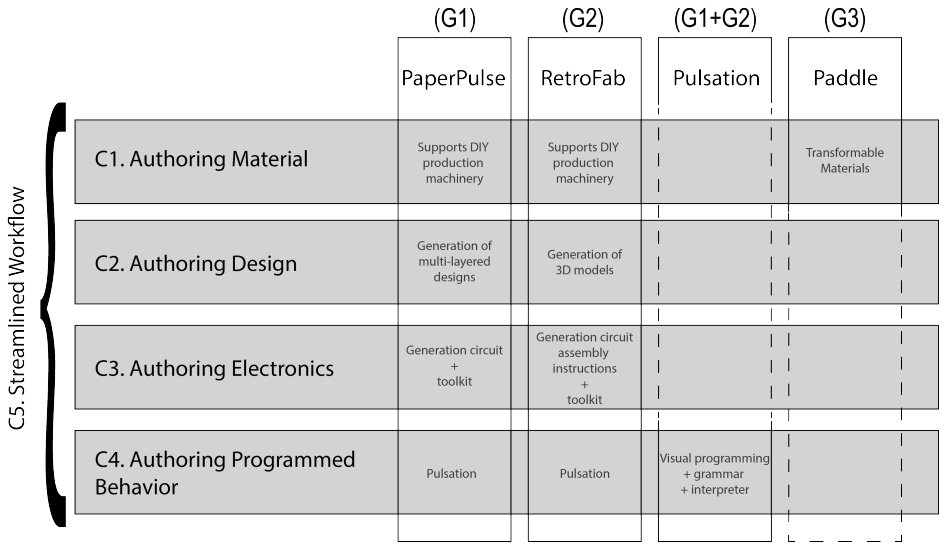


**Figure 8.1:** Linking the research challenges and goals to the contributions of this dissertation: Horizontal items define the research challenges (C1-C5). Vertical items define the contributions. The research goals (G1-G3) are at the top. The intersections summarize how the respective contributions resolve the research challenges.

**(C2) Technical expertise and spatial reasoning skills required for making designs**

The quality of physical artifacts realized with digital fabrication machinery, highly depend on the quality of the digital input model. Designing detailed models traditionally requires high levels of expertise in different domains (e.g. mechanical and electrical engineering). The PaperPulse (Chapter 3) design environment automatically generates a multi-layered designs from a visual layout on a 2D canvas. RetroFab (Chapter 4) contributes an algorithm to generate retrofit 3D structures from an annotated 3D scan of existing physical interfaces. Additionally, a redesign of the interface is generated that the user can adjust. This work therefore contributes new concepts and algorithms for computationally generated design.

**(C3) Electrical engineering knowledge required**

Many physical interfaces support interactivity by integrating electrical components and circuits. Making electronic circuits however, requires a lot of expertise. PaperPulse (Chapter 3) and RetroFab (Chapter 4)

support non-experts in making electronic circuits in three ways: (1) Automatic generation of an electronic system that connects every component to appropriate GPIO-pins on the microcontroller. (2) A toolkit of electrical components that are easy to assemble and interconnect. (3) A custom-generated tutorial that guides users through the assembly of the circuit. Using these techniques, our systems allow users without a technical background to make arbitrarily complex electronic systems.

**(C4) Advanced sensor-based programming knowledge required**
Programming sensor-based systems traditionally requires users to follow extensive tutorials, workshop, or courses. Pulsation 1.0 (Section 3.5) and 2.0 (Chapter 5) provide a visual environment for specifying sensor-based logic behavior. With Pulsation, first-time users compose basic logic by simply interconnecting electronic components. For advanced users, Pulsation offers fine-grained control over parameters, optimized for expressing temporal behavior.

**(C5) A wide diversity of heterogeneous tools and systems available**
PaperPulse (Chapter 3) and RetroFab (Chapter 4) software environments streamline the design and fabrication process to make physical interfaces. This process traditionally requires using advanced tools in electronics, programming, and graphical design. Besides, artifacts created in multiple heterogeneous software environments oftentimes have strong dependencies and require users to switch between tools frequently. The PaperPulse and RetroFab software environments optimize the workflow for realizing specific types of artifacts (i.e. thin-film interactive substrates and retrofit interfaces) and support the user from the initial idea until the final implementation.

## 8.2 Addressing the Research Goals

Addressing the challenges discussed in the previous section, leads to the research goals that were postulated in the introduction of this dissertation. In the following, we discuss how the presented systems and concepts address these research goals.

**(G1) Establish integrated software-hardware solutions that enable users without a technical background to make new interactive physical interfaces by authoring the form-factor and behavior.**

The PaperPulse software-hardware environment (Chapter 3) allows users without a technical background to design and fabricate thin-film flexible interfaces. In PaperPulse, users specify the visual design on a canvas and provide high-level logic specifications using the integrated Pulsation logic specification paradigm (Chapter 5). Using these specifications, PaperPulse generates a multi-layered circuit design, microcontroller code, as well as assembly instructions. Our example designs and use cases demonstrate the versatility of this approach. Additionally, user evaluations confirmed that PaperPulse enables non-experts to make new interactive physical interfaces.

**(G2) Establish integrated software-hardware solutions that enable users without a technical background to author the form-factor and behavior of existing interactive physical interfaces.**

The RetroFab software-hardware environment (Chapter 4) allows users without a technical background to make retrofit interfaces that change the form-factor and behavior of existing physical interfaces. From an annotated 3D scan and high-level lgoic specifications in Pulsation (Chapter 5), RetroFab generates a 3D retrofit structure, microcontroller code, as well as assembly instructions. Our example designs and use cases demonstrate the versatility of our approach. Additionally, a user study confirmed that RetroFab enables non-experts to adapt existing physical interfaces.

**(G3) Establish integrated software-hardware solutions that enable users without a technical background to adapt the form-factor and resulting behavior of interactive physical devices in real-time.**

The Paddle phone (Chapter 6), demonstrates how transformable materials can be used in interfaces to realize physical controls that change their shape and functionality in real-time. Compared to the digital fabrication techniques covered in this dissertation, shaping an artifact using transformable materials is significantly faster and thus allows for real-time transitions. A usage scenario demonstrates the feasibility of our approach during a phone call. A broader overview of interaction scenarios are covered in an extensive design space.

# Appendices

# Appendix A

## Pulsation 2.0 Grammar Instance

**Listing A.1:** Generated Pulsation grammar for an advanced code slot mechanism.

```
private PulsRadioButton m_radioButton;
private PulsLed m_ledCodeCorrect;
private PulsBuzzer m_buzzer;

private PulsLed m_progressLED1;
private PulsLed m_progressLED2;
private PulsLed m_progressLED3;
private PulsLed m_progressLED4;
private PulsLed m_progressLED5;
private PulsLed m_progressLED6;

private InputPatternBase m_inputPattern;

private IndependentOutputPatternBase m_ledCodeCompleted;
private OutputPatternRepeater m_incorrectAction;
private OutputActionBase m_restoreLedTrigger;
private AssignValueOfVariable m_mappingProgress;

protected override void initProg()
{
  // All conditions
  OrderedInputCollectionStrict codePatternCollection =
    new OrderedInputCollectionStrict()
  {
    EventPatterns = new InputPatternBase[]
    {
```

```
27      new LatchOperator(
28        new VarEqualsConstant(m_radioButton.getVarID(),
29        1)
30      ),
31      new SimultaneousInputCollection()
32      {
33        EventPatterns = new InputPatternBase[]
34        {
35          new LatchOperator(
36            new VarEqualsConstant(m_radioButton.getVarID(),
37            8)
38          ),
39          new LatchOperator(
40            new VarEqualsConstant(m_radioButton.getVarID(),
41            9)
42          )
43        }
44      }
45    }
46  };
47
48  UnOrderedInputCollection excludeEvents
49    = new UnOrderedInputCollection(UnOrderedInputCollectionType.OR)
        // Disjunction
50  {
51    EventPatterns = new InputPatternBase[]
52    {
53      new InputPatternSingleAction(
54        new VarEqualsConstant(m_radioButton.getVarID(), 0)
55      ),
56      new InputPatternSingleAction(
57        new VarInBetweenConstants(m_radioButton.getVarID(), 2, 7)
58      )
59    } // Excluding Buttons 0 and 2-7
60  };
61  codePatternCollection.setExcludePattern(excludeEvents);
62
63  ConditionRepeater repeater =
64    new ConditionRepeater(codePatternCollection, true);
65      // Reset on error in child: TRUE
66  repeater.EndCondition = new InputPatternSingleAction(
67    new VarEqualsConstant(repeater.getIterationVarID(), 2)
68    ); // 2 repetitions
69
70  TimeToCompleteCondition timeCondition
71    = new TimeToCompleteCondition(repeater, true);
72      // Reset on error in child: TRUE
73  timeCondition.EndCondition = new InputPatternSingleAction(
```

```
74      new VarSmallerThanConstant(timeCondition.getTimeVarID(), 100)
75      ); // 30 milliseconds
76
77    // Measuring the progress throughout the conditions
78    ProgressProperty progress = new ProgressProperty();
79    ConditionPropertyProxy proxy
80      = new ConditionPropertyProxy(timeCondition, progress);
81
82    m_inputPattern = proxy;
83    m_inputPattern.onActivatedEvent +=
84      new EventHandler(onInputPatternActivated);
85    m_inputPattern.onDeactivatedEvent +=
86      new EventHandler(onInputPatternDeactivated);
87    addCondition(m_inputPattern);
88
89    InputPatternBase timeResetCondition =
90      new VarGreaterThanConstant(timeCondition.getTimeVarID(), 100);
91    timeResetCondition.onActivatedEvent +=
92      timeResetConditionActivated;
93    addCondition(timeResetCondition);
94
95    // All actions
96    OutputPatternCollectionCteTiming incorrectAction
97      = new OutputPatternCollectionCteTiming()
98      {
99        Actions = new IndependentOutputPatternBase[]
100       {
101         new AssignConstant(
102           m_buzzer.getVarID(),
103           PulsBuzzer.ONSTATE
104         ),
105         new AssignConstant(
106           m_buzzer.getVarID(),
107           PulsBuzzer.OFFSTATE
108         ),
109         new DummyAction() // delay before repeat
110       }
111     };
112
113   incorrectAction.setDelay(incorrectAction.Actions[1], 0.5f);
114   incorrectAction.setDelay(incorrectAction.Actions[2], 1.0f);
115   m_incorrectAction = new OutputPatternRepeater(incorrectAction);
116   addAction(m_incorrectAction);
117
118   m_ledCodeCompleted = new AssignConstant(
119     m_ledCodeCorrect.getVarID(),
120     PulsLed.ONSTATE
121     );
```

```
122   addAction(m_ledCodeCompleted);
123
124   m_restoreLedTrigger = new RestoreVariable(
125     m_ledCodeCorrect.getVarID()
126     );
127   addAction(m_restoreLedTrigger);
128
129   //code to track progress
130   OutputPatternCollection progressLEDs
131     = new OutputPatternCollection()
132     {
133       Actions = new IndependentOutputPatternBase[]
134       {
135         new AssignConstant(
136           m_progressLED1.getVarID(),
137           PulsLed.ONSTATE
138         ),
139         new AssignConstant(
140           m_progressLED2.getVarID(),
141           PulsLed.ONSTATE
142         ),
143         new AssignConstant(
144           m_progressLED3.getVarID(),
145           PulsLed.ONSTATE
146         ),
147         new AssignConstant(
148           m_progressLED4.getVarID(),
149           PulsLed.ONSTATE
150         ),
151         new AssignConstant(
152           m_progressLED5.getVarID(),
153           PulsLed.ONSTATE
154         ),
155         new AssignConstant(
156           m_progressLED6.getVarID(),
157           PulsLed.ONSTATE
158         )
159       }
160     };
161
162   OutputProgressRegulator progressRegulator
163     = new OutputProgressRegulator(progressLEDs);
164   addAction(progressRegulator);
165
166   progressRegulator.addUndoAction(
167     progressLEDs.Actions[0],
168     new AssignConstant(m_progressLED1.getVarID(), PulsLed.OFFSTATE)
169   );
```

```
170   progressRegulator.addUndoAction(
171     progressLEDs.Actions[1],
172     new AssignConstant(m_progressLED2.getVarID(), PulsLed.OFFSTATE)
173   );
174   progressRegulator.addUndoAction(
175     progressLEDs.Actions[2],
176     new AssignConstant(m_progressLED3.getVarID(), PulsLed.OFFSTATE)
177   );
178   progressRegulator.addUndoAction(
179     progressLEDs.Actions[3],
180     new AssignConstant(m_progressLED4.getVarID(), PulsLed.OFFSTATE)
181   );
182   progressRegulator.addUndoAction(
183     progressLEDs.Actions[4],
184     new AssignConstant(m_progressLED5.getVarID(), PulsLed.OFFSTATE)
185   );
186   progressRegulator.addUndoAction(
187     progressLEDs.Actions[5],
188     new AssignConstant(m_progressLED6.getVarID(), PulsLed.OFFSTATE)
189   );
190
191   m_mappingProgress = new AssignValueOfVariable(
192     progressRegulator.getProgressVarID(),
193     progress.ProgressVarID
194   );
195   addAction(m_mappingProgress);
196
197
198   protected override void onLoop()
199   {
200     m_mappingProgress.execute();
201   }
202
203   private void timeResetConditionActivated(
204     object sender,
205     EventArgs e
206     )
207   {
208     m_codeInputPattern.reset();
209     m_incorrectAction.execute();
210   }
211
212   private void onInputPatternActivated(object sender, EventArgs e)
213   {
214     m_codeInputPattern.reset();
215     m_restoreLedTrigger.initRestorePoint();
216     m_ledCodeCompleted.execute();
217   }
```

```
218
219   private void onInputPatternDeactivated(object sender, EventArgs e)
220   {
221     m_codeInputPattern.reset();
222     m_ledCodeCompleted.pause();
223     m_restoreLedTrigger.execute();
224   }
225 }
```

# Appendix B

## Nederlandstalige Samenvatting

Visuele gebruikersinterfaces liggen aan de basis van de meeste computersystemen. Dagdagelijks maken we gebruik van de zogenaamde WIMP (windows, icons, menus, pointer) interactie stijl op desktop computers of aanraakgevoelige schermen op mobiele toestellen. Dergelijke systemen waarbij de visuele gebruikersinterface centraal staat zijn zeer populair doordat ze voor vele taken gebruikt kunnen worden zoals bijvoorbeeld het opstellen van documenten, communicatie, opzoeken van informatie etc. Nieuwe tools en technieken maken het zelfs mogelijk voor mensen zonder technische kennis om visuele gebruikersinterfaces te personaliseren zoals bijvoorbeeld het gebruik van filters voor het bewerken van foto's, interactieve profielen op sociale media, en gebruiksvriendelijke beheersystemen om websites of enquêtes te maken. In tegenstelling tot visuele gebruikersinterfaces zijn fysieke gebruikersinterfaces veel moeilijker om te maken en aan te passen doordat in deze systemen inhoudelijke componenten, zoals elektronische componenten, rechtstreeks verwerkt worden in het materiaal. Hierdoor is het personaliseren van fysieke interfaces veel moeilijker dan visuele gebruikersinterfaces. Voorbeelden van fysieke interfaces zijn traditionele huishoudelijke toestellen, muziek instrumenten, en meetapparatuur. Dergelijke systemen hebben vaak een zeer specifieke functionaliteit die geïntegreerd is in de fysieke vorm van het toestel. Hierdoor vergt het maken of aanpassen van dergelijke fysieke interfaces kennis van materialen, elektronica, design, alsook programmeren.

In deze doctoraatsthesis worden nieuwe tools en technieken onderzocht die het mogelijk maken voor mensen zonder een technische achtergrond om fysieke

interfaces te maken. Hierbij starten we in dit doctoraat met het onderzoeken van machines voor digitale fabricatie. Digitale fabricatie machines zoals 3D printers, laser cutters en inkjet printers met geleidende inkt, zijn zeer geschikt voor het maken van gepersonaliseerde fysieke interfaces doordat de productie kan verlopen in kleine volumes aan een lage prijs. Alhoewel deze machines aangestuurd worden vanaf een computer en er dus geen handmatige operaties nodig zijn, vergt het nog zeer veel kennis om vooraf een accuraat digitaal model op te bouwen in software omgevingen. Allereerst moet er een fysieke vorm in 3D of grafisch materiaal gemodelleerd worden in een virtuele omgeving. Daarnaast moeten er elektronische schema's gemaakt worden voor realiseren van een interactieve interface. Tenslotte moeten de elektronische componenten geprogrammeerd worden. Enkel mensen met een technische achtergrond zijn vertrouwd met deze technieken.

In deze thesis wordt er allereerst PaperPulse besproken, een design en fabricatie omgeving die gebruikers zonder enige technische kennis ondersteunt om fysieke interfaces te maken op dunne substraten. PaperPulse bestaat uit een software alsook hardware componenten. Gebruikers starten in de software omgeving met het ontwerpen van een fysieke interface door afbeeldingen en elektronische componenten op een canvas toe te voegen. De functionaliteit van deze componenten wordt beschreven in een hoog-niveau programmeertaal, genaamd Pulsation. Op basis van het visuele design op de canvas en de hoog-niveau instructies, genereerd PaperPulse een elektronisch circuit, laag-niveau programma instructies, en een handleiding. De handleiding beschrijft stap-voor-stap het proces om de laag-niveau instructies in te laden op de microcontroller en het proces om het elektronisch circuit te printen op een substraat m.b.v. een printer gevuld met geleidende inkt. Daarnaast voorziet PaperPulse een uitgebreide set van elektronische componenten die gemakkelijk bevestigd kunnen worden op dunne substraten. Na het doorlopen van deze stappen hebben gebruikers een persoonlijke fysieke interface gerealiseerd.

Om het mogelijk te maken voor gebruikers zonder enige technische kennis om bestaande fysieke toestellen aan te passen en te herprogrammering, bespreken we RetroFab. Het doel van RetroFab is het realiseren van een ondersteunende constructie die over een bestaand fysiek toestel (e.g. huishoudelijk toestel) geplaatst kan worden. Deze constructie integreert sensoren en actuatoren om het bestaande toestel te bedienen. Bovenop deze ondersteunende constructie kan dan een nieuwe gebruikersinterface van het bestaande toestel geïntegreerd worden. Gelijkaardig aan PaperPulse ondersteunt RetroFab de gebruiker in het realiseren van een dergelijke ondersteunende constructie. RetroFab ondersteunt ook een set van actuatoren en basis elektronica com-

ponenten voor het aansturen van veel gebruikte componenten in huishoudtoestellen.

In zowel PaperPulse als RetroFab specifiëren gebruikers zonder programmeerkennis logica met behulp van een visuele programmeertaal genaamd Pulsation. Pulsation is geoptimaliseerd voor het definiëren van logica in sensorgebaseerde systemen. Hiervoor biedt Pulsation grammatica die voornamelijk focust op temporele relaties. Leken specifiëren logica in deze grammatica door het tekenen van visuele links tussen elektronische componenten op een canvas. Geavanceerde constructies zijn beschikbaar voor gebruikers met meer ervaring. De resulterende Pulsation grammatica wordt uitgevoerd door een zogenaamde Pulsation vertolker. Pulsation is compatibel met zowel desktop computers alsook microcontrollers.

Tenslotte bekijken we in deze doctoraatsthesis hoe fysieke interfaces dynamische veranderingen kunnen toelaten tijdens hun gebruik. In traditionele visuele gebruikersinterfaces is het vaak mogelijk om de lay-out van de omgeving aan te passen met een simpele actie. Voor fysieke interfaces is het een grote uitdaging om de fysieke vorm tijdens het gebruik dynamisch aan te passen. Daarom bespreken we Paddle, een vervormbare fysieke interface die een aantal fysieke besturingselementen ondersteund. Elk besturingselement heeft een andere vorm en ondersteunt unieke interactiemogelijkheden. Gebruikers zonder technische achtergrond krijgen simpele instructies om de interface snel te vervormen tussen deze verschillende besturingselementen. Paddle wordt bestudeerd in de context van mobiele toestellen gezien het zeker in mobiele situaties onmogelijk is om wijzigingen aan te brengen aan de fysieke vorm van een toestel met behulp van andere gereedschappen.

# Bibliography


[Abelson 74] Hal Abelson, Nat Goodman & Lee Rudolph. *Logo manual.* 1974.

[Agrawal 15] Harshit Agrawal, Udayan Umapathi, Robert Kovacs, Johannes Frohnhofen, Hsiang-Ting Chen, Stefanie Mueller & Patrick Baudisch. *Protopiper: Physically Sketching Room-Sized Objects at Actual Scale.* In Proc. UIST '15, pages 427–436. ACM, 2015.

[Ahmaniemi 14] Teemu T Ahmaniemi, Johan Kildal & Merja Haveri. *What is a device bend gesture really good for?* In Proc. CHI '14, pages 3503–3512. ACM, 2014.

[Alexander 12] Jason Alexander, Andrés Lucero & Sriram Subramanian. *Tilt displays: designing display surfaces with multi-axis tilting and actuation.* In Proc. MobileHCI '12, pages 161–170. ACM, 2012.

[Anabuki 07] Mahoro Anabuki & Hiroshi Ishii. *AR-Jig: a handheld tangible user interface for modification of 3D digital form via 2D physical curve.* In Proc. ISMAR '07, pages 55–66. IEEE, 2007.

[Annett 15] Michelle Annett, Tovi Grossman, Daniel Wigdor & George Fitzmaurice. *MoveableMaker: Facilitating the Design, Generation, and Assembly of Moveable Papercraft.* In Proc. UIST '15, pages 565–574. ACM, 2015.

[Apitz 04] Georg Apitz & François Guimbretière. *CrossY: A Crossing-based Drawing Application.* In Proc. UIST '04, pages 3–12. ACM, 2004.

[Ash 11] Jordan Ash, Monica Babes, Gal Cohen, Sameen Jalal, Sam Lichtenberg, Michael Littman, Vukosi Marivate, Phillip Quiza, Blase Ur & Emily Zhang. *Scratchable devices: user-friendly programming for household appliances.* In Proc. HCII '11, pages 137–146. Springer, 2011.

[Ballagas 03] Rafael Ballagas, Meredith Ringel, Maureen Stone & Jan Borchers. *iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments.* In Proc. CHI '03, pages 537–544. ACM, 2003.

[Barnes 02] David J Barnes. *Teaching introductory Java through LEGO MINDSTORMS models.* In Proc. SIGCSE Bulletin, volume 34, pages 147–151. ACM, 2002.

[Barr 99] Michael Barr. *How programmable logic works.* Embedded Systems Programming, pages 75–84, 1999.

[Bates 00] JB Bates, NJ Dudney, B Neudecker, A Ueda & CD Evans. *Thin-film lithium and lithium-ion batteries.* Solid State Ionics, vol. 135, no. 1, pages 33–45, 2000.

[Bau 08] Olivier Bau & Wendy E. Mackay. *OctoPocus: A Dynamic Guide for Learning Gesture-based Command Sets.* In Proc. UIST '08, pages 37–46. ACM, 2008.

[Bdeir 12] Ayah Bdeir & Paul Rothman. *Electronics As Material: LittleBits.* In Proc. TEI '12, pages 371–374. ACM, 2012.

[Block 08] Florian Block, Michael Haller, Hans Gellersen, Carl Gutwin & Mark Billinghurst. *VoodooSketch: Extending Interactive Surfaces with Adaptable Interface Palettes.* In Proc. TEI '08, pages 55–58, 2008.

[Buechley 08] Leah Buechley, Mike Eisenberg, Jaime Catchen & Ali Crockett. *The LilyPad Arduino: Using Computational Textiles to Investigate Engagement, Aesthetics, and Diversity in Computer Science Education.* In Proc. CHI '08, pages 423–432, 2008.

[Burstyn 15a] Jesse Burstyn, Nicholas Fellion, Paul Strohmeier & Roel Vertegaal. *PrintPut: Resistive and Capacitive Input Wid-*

*gets for Interactive 3D Prints.* In Proc. HCI '15, pages 332–339. Springer, 2015.

[Burstyn 15b] Jesse Burstyn, Paul Strohmeier & Roel Vertegaal. *DisplaySkin: Exploring Pose-Aware Displays on a Flexible Electrophoretic Wristband.* In Proc. TEI '15, pages 165–172. ACM, 2015.

[Buxton 86] W. Buxton & B. Myers. *A Study in Two-handed Input.* In Proc. CHI '86, pages 321–326. ACM, 1986.

[Carter 99] D. A. Carter & J. Diaz. The elements of pop-up: A pop-up book for aspiring paper engineers. Little Simon, 1999.

[Chen 15] Xiang'Anthony' Chen, Stelian Coros, Jennifer Mankoff & Scott E Hudson. *Encore: 3D printed augmentation of everyday objects with printed-over, affixed and interlocked attachments.* In Proc. UIST '15, pages 73–82. ACM, 2015.

[Chikofsky 90] Elliot J Chikofsky, James H Cross*et al.* *Reverse engineering and design recovery: A taxonomy.* Software, IEEE, vol. 7, no. 1, pages 13–17, 1990.

[Coelho 09] Marcelo Coelho, Lyndl Hall, Joanna Berzowska & Pattie Maes. *Pulp-based Computing: A Framework for Building Computers out of Paper.* In Proc. CHI EA '09, pages 3527–3528. ACM, 2009.

[Colella 01] Vanessa Stevens Colella, Eric Klopfer & Mitchel Resnick. Adventures in modeling: Exploring complex, dynamic systems with starlogo. Teachers College Press, 2001.

[Conway 00] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove & Kevin Christiansen. *Alice: lessons learned from building a 3D system for novices.* In Proc. CHI '00, pages 486–493. ACM, 2000.

[Daliri 16] Farshad Daliri & Audrey Girouard. *Visual Feedforward Guides for Performing Bend Gestures on Deformable Prototypes.* In Proc. GI '16, 2016.

[Davidoff 11] Scott Davidoff, Nicolas Villar, Alex S Taylor & Shahram Izadi. *Mechanical hijacking: how robots can accelerate*

*UbiComp deployments.* In Proc. Ubicomp '11, pages 267–270. ACM, 2011.

[Devendorf 15] Laura Devendorf & Kimiko Ryokai. *Being the Machine: Reconfiguring Agency and Control in Hybrid Fabrication.* In Proc. CHI '15, pages 2477–2486. ACM, 2015.

[Devendorf 16] Laura Devendorf, Joanne Lo, Noura Howell, Jung Lin Lee, Nan-Wei Gong, M. Emre Karagozler, Shiho Fukuhara, Ivan Poupyrev, Eric Paulos & Kimiko Ryokai. *"I Don'T Want to Wear a Screen": Probing Perceptions of and Possibilities for Dynamic Displays on Clothing.* In Proc. CHI '16, pages 6028–6039. ACM, 2016.

[Dey 04] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li & Daniel Hsu. *A CAPpella: Programming by Demonstration of Context-aware Applications.* In Proc. CHI '04, pages 33–40. ACM, 2004.

[Dijkstra 11] Rob Dijkstra, Christopher Perez & Roel Vertegaal. *Evaluating effects of structural holds on pointing and dragging performance with flexible displays.* In Proc. CHI '11, pages 1293–1302. ACM, 2011.

[Dixon 10] Morgan Dixon & James Fogarty. *Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure.* In Proc. CHI '10, pages 1525–1534, 2010.

[Djajadiningrat 02] Tom Djajadiningrat, Kees Overbeeke & Stephan Wensveen. *But how, Donald, tell us how?: on the creation of meaning in interaction design through feedforward and inherent feedback.* In Proc. DIS '02, pages 285–291. ACM, 2002.

[Erwin 00] Ben Erwin, Martha Cyr & Chris Rogers. *LEGO engineer and ROBOLAB: Teaching engineering with LabVIEW from kindergarten to graduate school.* International Journal of Engineering Education, vol. 16, no. 3, pages 181–192, 2000.

[Everett 57] Robert R Everett, CA Zraket & HD Benington. *SAGE: A data-processing system for air defense.* In Papers and discussions presented at the December 9-13, 1957, eastern joint computer conference: Computers with deadlines to meet, pages 148–155, 1957.

[Fitzmaurice 95] George W. Fitzmaurice, Hiroshi Ishii & William A. S. Buxton. *Bricks: laying the foundations for graspable user interfaces.* In Proc. CHI '95, pages 442–449. ACM, 1995.

[Follmer 13] Sean Follmer, Daniel Leithinger, Alex Olwal, Akimitsu Hogge & Hiroshi Ishii. *inFORM: Dynamic Physical Affordances and Constraints Through Shape and Object Actuation.* In Proc. UIST '13, pages 417–426. ACM, 2013.

[Fourney 12] Adam Fourney & Michael Terry. *PICL: Portable In-circuit Learner.* In Proc. UIST '12, pages 569–578. ACM, 2012.

[Gallant 08] David T. Gallant, Andrew G. Seniuk & Roel Vertegaal. *Towards more paper-like input: flexible input devices for foldable interaction styles.* In Proc. UIST '08, pages 283–286. ACM, 2008.

[Gannon 15] Madeline Gannon, Tovi Grossman & George Fitzmaurice. *Tactum: A Skin-Centric Approach to Digital Design and Fabrication.* In Proc. CHI '15, pages 1779–1788. ACM, 2015.

[Gannon 16] Madeline Gannon, Tovi Grossman & George Fitzmaurice. *ExoSkin: On-Body Fabrication.* In Proc. CHI '16, pages 5996–6007. ACM, 2016.

[Girouard 15] Audrey Girouard, Jessica Lo, Md Riyadh, Farshad Daliri, Alexander Keith Eady & Jerome Pasquero. *One-handed bend interactions with deformable smartphones.* In Proc. CHI '15, pages 1509–1518. ACM, 2015.

[Goldstine 46] Herman H Goldstine & Adele Goldstine. *The electronic numerical integrator and computer (ENIAC).* Mathematical Tables and Other Aids to Computation, vol. 2, no. 15, pages 97–110, 1946.

[Gomes 13] Antonio Gomes, Andrea Nesbitt & Roel Vertegaal. *MorePhone: a study of actuated shape deformations for flexible thin-film smartphone notifications.* In Proc. CHI '13, pages 583–592. ACM, 2013.

[Gong 11] Nan-Wei Gong, Steve Hodges & Joseph A Paradiso. *Leveraging conductive inkjet technology to build a scalable and versatile surface for ubiquitous sensing.* In Proc. Ubicomp '11, pages 45–54. ACM, 2011.

[Gong 14] Nan-Wei Gong, Jürgen Steimle, Simon Olberding, Steve Hodges, Nicholas Edward Gillian, Yoshihiro Kawahara & Joseph A Paradiso. *PrintSense: a versatile sensing technique to support multimodal flexible surface interaction.* In Proc. CHI '14, pages 1407–1410. ACM, 2014.

[Gorbet 98] Matthew G. Gorbet, Maggie Orth & Hiroshi Ishii. *Triangles: tangible interface for manipulation and exploration of digital information topography.* In Proc. CHI '98, pages 49–56. ACM, 1998.

[Gotsch 16] Daniel Gotsch, Xujing Zhang, Jesse Burstyn & Roel Vertegaal. *HoloFlex: A Flexible Holographic Smartphone with Bend Input.* In Proc. CHI EA '16, pages 3675–3678. ACM, 2016.

[Greenberg 01] Saul Greenberg & Chester Fitchett. *Phidgets: Easy Development of Physical Interfaces Through Physical Widgets.* In Proc. UIST '01, pages 209–218. ACM, 2001.

[Groeger 16] Daniel Groeger, Elena Chong Loo & Jürgen Steimle. *HotFlex: Post-print Customization of 3D Prints Using Embedded State Change.* In Proc. CHI '16, pages 420–432. ACM, 2016.

[Grossman 10] Tovi Grossman & George Fitzmaurice. *ToolClips: An Investigation of Contextual Video Assistance for Functionality Understanding.* In CHI '10, pages 1515–1524. ACM, 2010.

[Hancock 09] Mark Hancock, Otmar Hilliges, Christopher Collins, Dominikus Baur & Sheelagh Carpendale. *Exploring tangible*

*and direct touch interfaces for manipulating 2D and 3D information on a digital table.* In Proc. ITS '09, pages 77–84. ACM, 2009.

[Hartmann 06] Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher & Jennifer Gee. *Reflective Physical Prototyping Through Integrated Design, Test, and Analysis.* In Proc. UIST '06, pages 299–308. ACM, 2006.

[Hartmann 07] Björn Hartmann, Leith Abdulla, Manas Mittal & Scott R. Klemmer. *Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition.* In Proc. CHI '07, pages 145–154. ACM, 2007.

[Harvey 14] Brian Harvey, Daniel D Garcia, Tiffany Barnes, Nathaniel Titterton, Omoju Miller, Dan Armendariz, Jon McKinsey, Zachary Machardy, Eugene Lemon, Sean Morris*et al. Snap!(build your own blocks).* In Proc. of the 45th ACM technical symposium on Computer science education, pages 749–749. ACM, 2014.

[Hawkes 10] E. Hawkes, B. An, N. M. Benbernou, H. Tanaka, S. Kim, E.D Demaine, D. Rus & R. J. Wood. *Programmable matter by folding.* Proc. NAS, vol. 107, no. 28, pages 12441–12445, 2010.

[Hemmert 10] Fabian Hemmert, Susann Hamann, Matthias Löwe, Josefine Zeipelt & Gesche Joost. *Shape-changing mobiles: tapering in two-dimensional deformational displays in mobile phones.* In Proc. CHI EA '10, pages 3075–3080. ACM, 2010.

[Hinckley 09] Ken Hinckley, Morgan Dixon, Raman Sarin, Francois Guimbretiere & Ravin Balakrishnan. *Codex: a dual screen tablet computer.* In Proc. CHI '09, pages 1933–1942. ACM, 2009.

[Hodges 13] Steve Hodges, James Scott, Sue Sentance, Colin Miller, Nicolas Villar, Scarlet Schwiderski-Grosche, Kerry Hammil & Steven Johnston. *.NET Gadgeteer: A New Platform*

*for K-12 Computer Science Education.* In Proc. SIGCSE '13, pages 391–396. ACM, 2013.

[Hodges 14] Steve Hodges, Nicolas Villar, Nicholas Chen, Tushar Chugh, Jie Qi, Diana Nowacka & Yoshihiro Kawahara. *Circuit Stickers: Peel-and-stick Construction of Interactive Electronic Prototypes.* In Proc. CHI '14, pages 1743–1746. ACM, 2014.

[Holman 05] David Holman, Roel Vertegaal, Mark Altosaar, Nikolaus Troje & Derek Johns. *Paper windows: interaction techniques for digital paper.* In Proc. CHI '05, pages 591–599. ACM, 2005.

[Holman 11] David Holman & Roel Vertegaal. *TactileTape: Low-cost Touch Sensing on Curved Surfaces.* In Proc. UIST '11 Adjunct, pages 17–18. ACM, 2011.

[Hook 14] Jonathan Hook, Thomas Nappey, Steve Hodges, Peter Wright & Patrick Olivier. *Making 3D printed objects interactive using wireless accelerometers.* In Proc. CHI EA '14, pages 1435–1440. ACM, 2014.

[Hudson 06] Scott E. Hudson & Jennifer Mankoff. *Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape.* In Proc. UIST '06, pages 289–298. ACM, 2006.

[Igarashi 07] Takeo Igarashi, Satoshi Matsuoka & Hidehiko Tanaka. *Teddy: a sketching interface for 3D freeform design.* In Proc. Siggraph '07, page 21. ACM, 2007.

[Ion 16] A. Ion, J. Frohnhofen, M. Alistar, L. Wall, J. Kovacs R. andLindsay, P. Lopes, H.-T. Chen & P. Baudisch. *Metamaterial Mechanisms.* In Proc. UIST '16. ACM, 2016.

[Ishiguro 14] Yoshio Ishiguro & Ivan Poupyrev. *3D printed interactive speakers.* In Proc. CHI '14, pages 1733–1742. ACM, 2014.

[Ishii 97] Hiroshi Ishii & Brygg Ullmer. *Tangible bits: towards seamless interfaces between people, bits and atoms.* In Proc. CHI '97, pages 234–241. ACM, 1997.

[Jabr 13] Ferris Jabr. *Why the Brain Prefers Paper.* Scientific American, vol. 309, no. 5, pages 48–53, 2013.

[Jawitz 97] Martin W Jawitz. Printed circuit board materials handbook. McGraw Hill Professional, 1997.

[Johnson 89] Jeff Johnson, Teresa L Roberts, William Verplank, David C Smith, Charles H Irby, Marian Beard & Kevin Mackey. *The xerox star: A retrospective.* Computer, vol. 22, no. 9, pages 11–26, 1989.

[Jones 16] Michael D. Jones, Kevin Seppi & Dan R. Olsen. *What You Sculpt is What You Get: Modeling Physical Interactive Devices with Clay and 3D Printed Widgets.* In Proc. CHI '16, pages 876–886. ACM, 2016.

[Karagozler 13] Mustafa Emre Karagozler, Ivan Poupyrev, Gary K. Fedder & Yuri Suzuki. *Paper Generators: Harvesting Energy from Touching, Rubbing and Sliding.* In Proc. UIST '13, pages 23–30. ACM, 2013.

[Kawahara 13] Yoshihiro Kawahara, Steve Hodges, Benjamin S. Cook, Cheng Zhang & Gregory D. Abowd. *Instant Inkjet Circuits: Lab-based Inkjet Printing to Support Rapid Prototyping of UbiComp Devices.* In Proc. UbiComp '13, pages 363–372. ACM, 2013.

[Kelleher 05] Caitlin Kelleher & Randy Pausch. *Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers.* ACM Comput. Surv., vol. 37, no. 2, pages 83–137, June 2005.

[Khalilbeigi 11] Mohammadreza Khalilbeigi, Roman Lissermann, Max Mühlhäuser & Jürgen Steimle. *Xpaaand: interaction techniques for rollable displays.* In Proc. CHI '11, pages 2729–2732. ACM, 2011.

[Khalilbeigi 12] Mohammadreza Khalilbeigi, Roman Lissermann, Wolfgang Kleine & Jürgen Steimle. *FoldMe: interacting with double-sided foldable displays.* In Proc. TEI '12, pages 33–40. ACM, 2012.

[Kildal 12] Johan Kildal & Graham Wilson. *Feeling it: the roles of stiffness, deformation range and feedback in the control of deformable ui.* In Proc. ICMI '12, pages 393–400. ACM, 2012.

[Kildal 13] Johan Kildal, Andrés Lucero & Marion Boberg. *Twisting Touch: Combining Deformation and Touch As Input Within the Same Interaction Cycle on Handheld Devices.* In Proc. MobileHCI '13, pages 237–246, 2013.

[Kim 07] Seung Han Kim & Jae Wook Jeon. *Programming LEGO Mindstorms NXT with visual programming.* In Control, Automation and Systems, 2007. ICCAS'07. International Conference on, pages 2468–2472. IEEE, 2007.

[Kirsh 94] David Kirsh & Paul Maglio. *On distinguishing epistemic from pragmatic action.* Cognitive science, vol. 18, no. 4, pages 513–549, 1994.

[Kramer 11] Rebecca K Kramer, Carmel Majidi & Robert J Wood. *Wearable tactile keypad with stretchable artificial skin.* In Proc. ICRA '11, pages 1103–1107. IEEE, 2011.

[Lahey 11] Byron Lahey, Audrey Girouard, Winslow Burleson & Roel Vertegaal. *PaperPhone: understanding the use of bend gestures in mobile devices with flexible electronic paper displays.* In Proc. CHI '11, pages 1303–1312. ACM, 2011.

[Lau 11] Manfred Lau, Akira Ohgawara, Jun Mitani & Takeo Igarashi. *Converting 3D furniture models to fabricatable parts and connectors.* In Proc. SIGGRAPH '11, volume 30, page 85. ACM, 2011.

[Lee 04] Johnny C. Lee, Daniel Avrahami, Scott E. Hudson, Jodi Forlizzi, Paul H. Dietz & Darren Leigh. *The Calder Toolkit: Wired and Wireless Components for Rapidly Prototyping Interactive Devices.* In Proc. DIS '04, pages 167–175. ACM, 2004.

[Lee 08] Johnny C. Lee, Scott E. Hudson & Edward Tse. *Foldable interactive displays.* In Proc. UIST '08, pages 287–290. ACM, 2008.

[Lee 10] Sang-Su Lee, Sohyun Kim, Bopil Jin, Eunji Choi, Boa Kim, Xu Jia, Daeeop Kim & Kun-pyo Lee. *How users manipulate deformable displays as input devices.* In Proc. CHI '10, pages 1647–1656. ACM, 2010.

[Leflar 14] Meagan Leflar & Audrey Girouard. *Navigating in 3D space with a handheld flexible device.* Entertainment Computing, vol. 5, no. 4, pages 205–209, 2014.

[Li 16] Hanchuan Li, Eric Brockmeyer, Elizabeth J. Carter, Josh Fromm, Scott E. Hudson, Shwetak N. Patel & Alanson Sample. *PaperID: A Technique for Drawing Functional Battery-Free Wireless Interfaces on Paper.* In Proc. CHI '16, pages 5885–5896. ACM, 2016.

[Lin 00] James Lin, Mark W. Newman, Jason I. Hong & James A. Landay. *DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design.* In Proc. CHI '00, pages 510–517. ACM, 2000.

[Lyons 12] Kent Lyons, David Nguyen, Daniel Ashbrook & Sean White. *Facet: a multi-segment wrist worn system.* In Proc. UIST '12, pages 123–130. ACM, 2012.

[Lysecky 09] Susan Lysecky & Frank Vahid. *Enabling Nonexpert Construction of Basic Sensor-based Systems.* ACM Transactions on Human-Computer Interaction, vol. 16, no. 1, pages 1:1–1:28, April 2009.

[Maqsood 14] Sana Maqsood, Sonia Chiasson & Audrey Girouard. *Bend Passwords: using gestures to authenticate on flexible devices.* pages 1–28. Springer, 2014.

[Mellis 13] David A. Mellis, Sam Jacoby, Leah Buechley, Hannah Perner-Wilson & Jie Qi. *Microcontrollers As Material: Crafting Circuits with Paper, Conductive Ink, Electronic Components, and an "Untoolkit".* In Proc. TEI '13, pages 83–90. ACM, 2013.

[Merrill 07] David Merrill, Jeevan Kalanithi & Pattie Maes. *Siftables: Towards Sensor Network User Interfaces.* In Proc. TEI '07, pages 75–78. ACM, 2007.

[Michelitsch 04] Georg Michelitsch, Jason Williams, Martin Osen, Beatriz Jimenez & Stefan Rapp. *Haptic chameleon: a new concept of shape-changing user interface controls with force feedback.* In Proc. CHI EA '04, pages 1305–1308. ACM, 2004.

[Millner 11] Amon Millner & Edward Baafi. *Modkit: blending and extending approachable platforms for creating computer programs and interactive objects.* In Proc. IDC '11, pages 250–253. ACM, 2011.

[Mori 07] Yuki Mori & Takeo Igarashi. *Plushie: an interactive design system for plush toys.* In Proc. TOG '07, volume 26, page 45. ACM, 2007.

[Mueller 12] Stefanie Mueller, Pedro Lopes & Patrick Baudisch. *Interactive construction: interactive fabrication of functional mechanical devices.* In Proc. UIST '12, pages 599–606. ACM, 2012.

[Mueller 13] Stefanie Mueller, Bastian Kruck & Patrick Baudisch. *LaserOrigami: laser-cutting 3D objects.* In Proc. CHI '13, pages 2585–2592. ACM, 2013.

[Mueller 14a] Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François Guimbretière & Patrick Baudisch. *WirePrint: 3D printed previews for fast prototyping.* In Proc. UIST '14, pages 273–280. ACM, 2014.

[Mueller 14b] Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen & Patrick Baudisch. *faBrickation: fast 3D printing of functional objects by integrating construction kit building blocks.* In Proc. CHI '14, pages 3827–3834. ACM, 2014.

[Mugellini 07] Elena Mugellini, Elisa Rubegni, Sandro Gerardi & Omar Abou Khaled. *Using Personal Objects As Tangible Interfaces for Memory Recollection and Sharing.* In Proc. TEI '07, pages 231–238. ACM, 2007.

[Myny 12] Kris Myny, Erik van Veenendaal, Gerwin H Gelinck, Jan Genoe, Wim Dehaene & Paul Heremans. *An 8-bit, 40-instructions-per-second organic microprocessor on plastic foil.* IEEE Journal of Solid-State Circuits, vol. 47, no. 1, pages 284–291, 2012.

[Nielsen 94] Jakob Nielsen. *Enhancing the Explanatory Power of Usability Heuristics.* In Proc. CHI '94, pages 152–158. ACM, 1994.

[Olberding 13] Simon Olberding, Nan-Wei Gong, John Tiab, Joseph A. Paradiso & Jürgen Steimle. *A Cuttable Multi-touch Sensor.* In Proc. UIST '13, pages 245–254. ACM, 2013.

[Olberding 14] Simon Olberding, Michael Wessely & Jürgen Steimle. *PrintScreen: Fabricating Highly Customizable Thin-film Touch-displays.* In Proc. UIST '14, pages 281–290. ACM, 2014.

[Olberding 15] Simon Olberding, Sergio Soto Ortega, Klaus Hildebrandt & Jürgen Steimle. *Foldio: Digital Fabrication of Interactive and Shape-Changing Objects With Foldable Printed Electronics.* In Proc. UIST '15, pages 223–232. ACM, 2015.

[Olsen 07] Dan R. Olsen Jr. *Evaluating User Interface Systems Research.* In Proc. UIST '07, pages 251–258. ACM, 2007.

[Ono 13] Makoto Ono, Buntarou Shizuki & Jiro Tanaka. *Touch & activate: adding interactivity to existing objects using active acoustic sensing.* In Proc. UIST '13, pages 31–40, 2013.

[Parzer 16] Patrick Parzer, Kathrin Probst, Teo Babic, Christian Rendl, Anita Vogl, Alex Olwal & Michael Haller. *FlexTiles: A Flexible, Stretchable, Formable, Pressure-Sensitive, Tactile Input Sensor.* In Proc. CHI EA '16, pages 3754–3757. ACM, 2016.

[Pedersen 14] Esben W Pedersen, Sriram Subramanian & Kasper Hornbæk. *Is my phone alive?: a large-scale study of shape change in handheld devices using videos.* In Proc. CHI '14, pages 2579–2588. ACM, ACM, 2014.

[Peng 16] Huaishu Peng, Rundong Wu, Steve Marschner & François Guimbretière. *On-The-Fly Print: Incremental Printing While Modelling.* In Proc. CHI '16, pages 887–896. ACM, 2016.

[Piper 02] Ben Piper, Carlo Ratti & Hiroshi Ishii. *Illuminating clay: a 3-D tangible interface for landscape analysis.* In Proc. CHI '02, pages 355–362. ACM, 2002.

[Post 00] E Rehmi Post, Maggie Orth, PR Russo & Neil Gershenfeld. *E-broidery: Design and fabrication of textile-based computing.* IBM Systems journal, vol. 39, no. 3.4, pages 840–860, 2000.

[Poupyrev 16] Ivan Poupyrev, Nan-Wei Gong, Shiho Fukuhara, Mustafa Emre Karagozler, Carsten Schwesig & Karen E. Robinson. *Project Jacquard: Interactive Digital Textiles at Scale.* In Proc. CHI '16, pages 4216–4227. ACM, 2016.

[Puckette 96] Miller Puckette*et al. Pure Data: another integrated computer music environment.* The second intercollege computer music concerts, pages 37–41, 1996.

[Qi 10] Jie Qi & Leah Buechley. *Electronic Popables: Exploring Paper-based Computing Through an Interactive Pop-up Book.* In Proc. TEI '10, pages 121–128. ACM, 2010.

[Qi 14] Jie Qi & Leah Buechley. *Sketching in Circuits: Designing and Building Electronics on Paper.* In Proc. CHI '14, pages 1713–1722. ACM, 2014.

[Raffle 04] Hayes Solos Raffle, Amanda J. Parkes & Hiroshi Ishii. *Topobo: a constructive assembly system with kinetic memory.* In Proc. CHI '04, pages 647–654. ACM, 2004.

[Ramakers 12] Raf Ramakers, Davy Vanacken, Kris Luyten, Karin Coninx & Johannes Schöning. *Carpus: A Non-intrusive User Identification Technique for Interactive Surfaces.* In Proc. UIST '12, pages 35–44. ACM, 2012.

[Ramakers 13] Raf Ramakers, Kris Luyten & Johannes Schöning. *Learning from 3D puzzles to inform future interactions with de-*

*formable mobile interfaces.* In Proc. Workshop on Displays Take New Shape: An Agenda for Interactive Surfaces CHI '13, 2013.

[Ramakers 14] Raf Ramakers, Johannes Schöning & Kris Luyten. *Paddle: Highly Deformable Mobile Devices with Physical Controls.* In Proc. CHI '14, pages 2569–2578. ACM, 2014.

[Ramakers 15a] Raf Ramakers. *Reconfiguring and Fabricating Special-Purpose Tangible Controls.* In Adj. Proc. UIST '15, pages 5–8. ACM, 2015.

[Ramakers 15b] Raf Ramakers, Kashyap Todi & Kris Luyten. *An End-User Development Approach for Designers to create Interactive Paper.* In Proc. Workshop on End User Development in the Internet of Things Era CHI '15, 2015.

[Ramakers 15c] Raf Ramakers, Kashyap Todi & Kris Luyten. *PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs.* In Proc. CHI '15, pages 2457–2466. ACM, 2015.

[Ramakers 15d] Raf Ramakers, Kashyap Todi & Kris Luyten. *PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs.* In Proc. SIGGRAPH 2015: Studio, pages 3:1–3:1. ACM, 2015.

[Ramakers 15e] Raf Ramakers, Kashyap Todi & Kris Luyten. *PaperPulse: An Integrated Approach to Fabricating Interactive Paper.* In Proc. CHI '15, pages 267–270. ACM, 2015.

[Ramakers 16] Raf Ramakers, Fraser Anderson, Tovi Grossman & George Fitzmaurice. *RetroFab: A Design Tool for Retrofitting Physical Interfaces Using Actuators, Sensors and 3D Printing.* In Proc. CHI '16, pages 409–419. ACM, 2016.

[Rasmussen 12] Majken K. Rasmussen, Esben W. Pedersen, Marianne G. Petersen & Kasper Hornbaek. *Shape-changing interfaces: a review of the design space and open research questions.* In Proc. CHI '12, pages 735–744. ACM, 2012.

[Rendl 12] Christian Rendl, Patrick Greindl, Michael Haller, Martin Zirkl, Barbara Stadlober & Paul Hartmann. *PyzoFlex:*

*printed piezoelectric pressure sensing foil.* In Proc. UIST '12, pages 509–518. ACM, 2012.

[Rendl 14] Christian Rendl, David Kim, Sean Fanello, Patrick Parzer, Christoph Rhemann, Jonathan Taylor, Martin Zirkl, Gregor Scheipl, Thomas Rothländer, Michael Haller*et al.* *FlexSense: a transparent self-sensing deformable surface.* In Proc. UIST '14, pages 129–138. ACM, 2014.

[Rendl 16] Christian Rendl, David Kim, Patrick Parzer, Sean Fanello, Martin Zirkl, Gregor Scheipl, Michael Haller & Shahram Izadi. *FlexCase: Enhancing Mobile Interaction with a Flexible Sensing and Display Cover.* In Proc. CHI '16, pages 5138–5150. ACM, 2016.

[Resnick 09] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman*et al.* *Scratch: programming for all.* Communications of the ACM, vol. 52, no. 11, pages 60–67, 2009.

[Roudaut 13] Anne Roudaut, Abhijit Karnik, Markus Löchtefeld & Sriram Subramanian. *Morphees: toward high "shape resolution" in self-actuated flexible mobile devices.* In Proc. CHI '13, pages 593–602. ACM, 2013.

[Roudaut 16] Anne Roudaut, Diana Krusteva, M McCoy, A Karnik, Karthik Ramani & Sriram Subramanian. *Cubimorph: designing modular interactive devices.* In Proc. ICRA '16, pages 3339–3345. IEEE, 2016.

[Rudeck 12] Frederik Rudeck & Patrick Baudisch. *Rock-paper-fibers: bringing physical affordance to mobile touch devices.* In Proc. CHI '12, pages 1929–1932. ACM, 2012.

[Saul 10] Greg Saul, Cheng Xu & Mark D. Gross. *Interactive Paper Devices: End-user Design & Fabrication.* In Proc. TEI '10, pages 205–212. ACM, 2010.

[Saul 11] Greg Saul, Manfred Lau, Jun Mitani & Takeo Igarashi. *SketchChair: an all-in-one chair design system for end users.* In Proc. TEI '11, pages 73–80. ACM, 2011.

[Savage 12] Valkyrie Savage, Xiaohan Zhang & Björn Hartmann. *Midas: Fabricating Custom Capacitive Touch Sensors to Prototype Interactive Objects.* In Proc. UIST '12, pages 579–588. ACM, 2012.

[Savage 13] Valkyrie Savage, Colin Chang & Björn Hartmann. *Sauron: embedded single-camera sensing of printed physical user interfaces.* In Proc. UIST '13, pages 447–456. ACM, 2013.

[Savage 14] Valkyrie Savage, Ryan Schmidt, Tovi Grossman, George Fitzmaurice & Björn Hartmann. *A series of Tubes: Adding Interactivity to 3D Prints Using Internal Pipes.* In Proc. UIST '14, pages 3–12. ACM, 2014.

[Savage 15] Valkyrie Savage, Sean Follmer, Jingyi Li & Björn Hartmann. *Makers' Marks: Physical Markup for Designing and Fabricating Functional Objects.* In Proc. UIST '15, pages 103–108. ACM, 2015.

[Schmidt 88] Richard A Schmidt & Tim Lee. Motor control and learning, 5e. Human kinetics, 1988.

[Schmidt 10] Ryan Schmidt & Karan Singh. *Meshmixer: An Interface for Rapid Mesh Composition.* In Proc. SIGGRAPH '10 Talks, pages 6:1–6:1. ACM, 2010.

[Schmidt 14] Dominik Schmidt, Raf Ramakers, Esben W. Pedersen, Johannes Jasper, Sven Köhler, Aileen Pohl, Hannes Rantzsch, Andreas Rau, Patrick Schmidt, Christoph Sterz, Yanina Yurchenko & Patrick Baudisch. *Kickables: Tangibles for Feet.* In Proc. CHI '14. ACM, 2014.

[Schwesig 04] Carsten Schwesig, Ivan Poupyrev & Eijiro Mori. *Gummi: a bendable computer.* In Proc. CHI '04, pages 263–270. ACM, 2004.

[Shorter 14] Michael Shorter, Jon Rogers & John McGhee. *Enhancing Everyday Paper Interactions with Paper Circuits.* In Proc. DIS '14, pages 39–42. ACM, 2014.

[Song 06] Hyunyoung Song, François Guimbretière, Chang Hu & Hod Lipson. *ModelCraft: capturing freehand annotations*

*and edits on physical 3D models.* In Proc. UIST '06, pages 13–22. ACM, 2006.

[Spielberg 16] Andrew Spielberg, Alanson Sample, Scott E. Hudson, Jennifer Mankoff & James McCann. *RapID: A Framework for Fabricating Low-Latency Interactive Objects with RFID Tags.* In Proc. CHI '16, pages 5897–5908. ACM, 2016.

[Steimle 13] Jürgen Steimle, Andreas Jordt & Pattie Maes. *Flexpad: highly flexible bending interactions for projected handheld displays.* In Proc. CHI '13, pages 237–246. ACM, 2013.

[Strohmeier 16] Paul Strohmeier, Jesse Burstyn, Juan Pablo Carrascal, Vincent Levesque & Roel Vertegaal. *ReFlex: A Flexible Smartphone with Active Haptic Feedback for Bend Input.* In Proc. TEI '16, pages 185–192. ACM, 2016.

[Sugiura 12] Yuta Sugiura, Masahiko Inami & Takeo Igarashi. *A thin stretchable interface for tangential force measurement.* In Proc. UIST '12, pages 529–536. ACM, 2012.

[Sutherland 64] Ivan E Sutherland. *Sketch pad a man-machine graphical communication system.* In Proc. of the SHARE design automation workshop, pages 6–329, 1964.

[Tarun 11] Aneesh P Tarun, Byron Lahey, Audrey Girouard, Winslow Burleson & Roel Vertegaal. *Snaplet: using body shape to inform function in mobile flexible display devices.* In Proc. CHI EA '11, pages 329–334. ACM, 2011.

[Tarun 13] Aneesh P. Tarun, Peng Wang, Audrey Girouard, Paul Strohmeier, Derek Reilly & Roel Vertegaal. *PaperTab: an electronic paper computer with multiple large flexible electrophoretic displays.* In Proc. CHI EA '13, pages 3131–3134. ACM, 2013.

[Teibrich 15] Alexander Teibrich, Stefanie Mueller, François Guimbretière, Robert Kovacs, Stefan Neubert & Patrick Baudisch. *Patching Physical Objects.* In Proc. UIST ' 15, pages 83–91. ACM, 2015.

[Terrenghi 08] Lucia Terrenghi, David Kirk, Hendrik Richter, Sebastian Krämer, Otmar Hilliges & Andreas Butz. *Physical handles at the interactive surface: exploring tangibility and its benefits.* In Proc. AVI '08, pages 138–145. ACM, 2008.

[Todi 16] Kashyap Todi, Daryl Weir & Antti Oulasvirta. *Sketchplore: Sketch and Explore Layout Designs with an Optimiser.* In Proc. DIS '16, pages 3780–3783. ACM, 2016.

[Torres 15] Cesar Torres, Tim Campbell, Neil Kumar & Eric Paulos. *HapticPrint: Designing Feel Aesthetics for Digital Fabrication.* In Proc. UIST '15, pages 583–591. ACM, 2015.

[Ullmer 00] B. Ullmer & H. Ishii. *Emerging frameworks for tangible user interfaces.* IBM Syst. J. '00, pages 915–931, 2000.

[Umetani 16] Nobuyuki Umetani & Ryan Schmidt. *SurfCuit: Surface Mounted Circuits on 3D Prints.* arXiv preprint arXiv:1606.09540, 2016.

[Underkoffler 99] John Underkoffler & Hiroshi Ishii. *Urp: a luminous-tangible workbench for urban planning and design.* In Proc. CHI '99, pages 386–393. ACM, 1999.

[Ur 14] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho & Michael L. Littman. *Practical Trigger-action Programming in the Smart Home.* In Proc. CHI '14, pages 803–812. ACM, 2014.

[Vanacken 08] Davy Vanacken, Alexandre Demeure, Kris Luyten & Karin Coninx. *Ghosts in the interface: Meta-user Interface Visualizations as Guides for Multi-touch interaction.* In Proc. TABLETOP '08, pages 81–84. IEEE, 2008.

[Varun 15] Perumal Varun & Daniel Wigdor. *Printem: Instant Printed Circuit Boards with Standard Office Printers and Inks.* In Proc. UIST '15, pages 243–251. ACM, 2015.

[Villar 07] Nicolas Villar, Kiel Mark Gilleade, Devina Ramdunyellis & Hans Gellersen. *The VoodooIO Gaming Kit: A Real-time Adaptable Gaming Controller.* Computers in Entertainment, vol. 5, no. 3, July 2007.

[Vlachokyriakos 14] Vasilis Vlachokyriakos, Rob Comber, Karim Ladha, Nick Taylor, Paul Dunphy, Patrick McCorry & Patrick Olivier. *PosterVote: Expanding the Action Repertoire for Local Political Activism.* In Proc. DIS '14, pages 795–804. ACM, 2014.

[Warren 13] Kristen Warren, Jessica Lo, Vaibhav Vadgama & Audrey Girouard. *Bending the rules: bend gesture classification for flexible displays.* In Proc. CHI '13, pages 607–610. ACM, 2013.

[Watanabe 08] Jun-ichiro Watanabe, Arito Mochizuki & Youichi Horry. *Bookisheet: bendable device for browsing content using the metaphor of leafing through the pages.* In Proc. UbiComp '08, pages 360–369. ACM, 2008.

[Weichel 13] Christian Weichel, Manfred Lau & Hans Gellersen. *Enclosed: A Component-centric Interface for Designing Prototype Enclosures.* In Proc. TEI '13, pages 215–218. ACM, 2013.

[Weichel 14] Christian Weichel, Manfred Lau, David Kim, Nicolas Villar & Hans W. Gellersen. *MixFab: A Mixed-reality Environment for Personal Fabrication.* In Proc. CHI '14, pages 3855–3864. ACM, 2014.

[Weichel 15a] Christian Weichel, Jason Alexander, Abhijit Karnik & Hans Gellersen. *SPATA: Spatio-tangible tools for fabrication-aware design.* In Proc. TEI '15, pages 189–196. ACM, 2015.

[Weichel 15b] Christian Weichel, John Hardy, Jason Alexander & Hans Gellersen. *ReForm: integrating physical and digital design through bidirectional fabrication.* In Proc. UIST '15, pages 93–102. ACM, 2015.

[Weigel 15] Martin Weigel, Tong Lu, Gilles Bailly, Antti Oulasvirta, Carmel Majidi & Jürgen Steimle. *Iskin: flexible, stretchable and visually customizable on-body touch sensors for mobile computing.* In Proc. CHI '15, pages 2991–3000. ACM, 2015.

[Wells 96] Lisa K Wells & Jeffrey Travis. Labview for everyone: graphical programming made even easier. Prentice-Hall, Inc., 1996.

[Wigdor 16] Daniel Wigdor*et al.* *Foldem: Heterogeneous Object Fabrication via Selective Ablation of Multi-Material Sheets.* In Proc. CHI '16, pages 5765–5775. ACM, 2016.

[Wightman 11] Doug Wightman, Tim Ginn & Roel Vertegaal. *Bendflip: examining input techniques for electronic book readers with flexible form factors.* In Proc. INTERACT '11, pages 117–133. IEEE, 2011.

[Willis 12] Karl Willis, Eric Brockmeyer, Scott Hudson & Ivan Poupyrev. *Printed optics: 3D printing of embedded optical elements for interactive devices.* In Proc. UIST '12, pages 589–598. ACM, 2012.

[Wolber 11] David Wolber. *App inventor and real-world motivation.* In Proc. Technical symposium on Computer science education, pages 601–606. ACM, 2011.

[Wolper 83] Pierre Wolper. *Temporal logic can be more expressive.* Information and control, vol. 56, no. 1, pages 72–99, 1983.

[Zhang 16] Yunbo Zhang, Wei Gao, Luis Paredes & Karthik Ramani. *CardBoardiZer: Creatively Customize, Articulate and Fold 3D Mesh Models.* In Proc. CHI '16, pages 897–907. ACM, 2016.