

TangoHapps - An Integrated Development Environment for Smart Textiles

Juan Haladjian

INSTITUT FÜR INFORMATIK DER TECHNISCHEN UNIVERSITÄT
MÜNCHEN

Forschungs- und Lehrereinheit I Angewandte Softwaretechnik

TangoHapps - An Integrated Development Environment for Smart Textiles

Juan Haladjian

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen
Universität München zur Erlangung des akademischen Grades eines **Doktors der
Naturwissenschaften (Dr. rer. nat.)** genehmigten Dissertation.

Vorsitzende: Univ.-Prof. Dr. Hans-Joachim Bungartz
Prüfer der Dissertation: 1. Univ.-Prof. Bernd Brügge, Ph.D
2. Univ.-Prof. Daniel Siewiorek, Ph.D

Die Dissertation wurde am 28.06.2016 bei der Technischen Universität München
eingereicht und durch die Fakultät für Informatik am angenommen.

A Chechu, con cariño.

Acknowledgments

This work would not have been possible without the support of many people.

First of all I would like my professor Bernd Brügge to know how much of a positive influence he has been to my personal and professional development and thank him for encouraging me to grow in his team. I would also like to thank my second supervisor Daniel Siewiorek for his great insight and exhaustive feedback on my *Interactex* and *KneeHapp* papers.

I am very grateful to all the members of the Chair for Applied Software Engineering at the TUM. A special mention goes to my colleagues Zardosht Hodaie, Han Xu, Stephan Krusche, Emitzá Guzmán and Yang Li for proof reading my papers and different parts of my thesis and to Jan Knobloch for his support during the Custodian project. Furthermore, I would like to thank my friends Mohamed Souiai and Jelena Prsa for their warm friendship and for our motivational meetings at the Mensa.

TangoHapps was developed in collaboration with the Design Research Lab at the Universität der Künste in Berlin. In particular, *TangoHapps* came to life thanks to the great insights and efforts from Katharina Bredies. I would like to thank Gesche Joost, Katharina Bredies, Sara Diaz Rodriguez, Pauline Vierne, Hannah-Perner Wilson for our fruitful four year collaboration. Thanks also to Martijn ten Böhmer and Oscar Tomico from Technische Universiteit Eindhoven, Kyle Zhang from the Universiteit Twente and Jussi Mikkonen from Aalto University for contributing to *TangoHapps*' code base.

The research leading to these results has received funding from:

- Federal Ministry of Education and Research (BMBF) under grant number "01IS12057" as part of the German program "Software Campus".
- EIT ICT "Connected Textiles" activity in the Activity Nr. 13087 under the "Smart Spaces" Action Line.
- Bavaria California Technology Center (BaCaTeC).

The responsibility concerning the content in this dissertation is left to me, the author.

Abstract

Construction kits and development environments have dramatically reduced the time and expertise required for the development of physical interactive devices. The field of smart textiles, however, is relatively new and still lacks of established tools that support their development. In this dissertation, we introduce *TangoHapps*, a visual programming and simulation environment specifically designed for smart textiles. *TangoHapps* supports different activities related to smart textile development, including application software development, testing and circuit design. *TangoHapps*' visual programming capabilities span across mobile phones and smart textiles enabling users to take advantage of capabilities present on both devices without writing source code. *TangoHapps* supports the development of use cases from different application domains, which we demonstrate by recreating two smart textiles from different application domains. Furthermore, we provide evidence from two user studies that *TangoHapps* lowers entrance barrier to smart textile development.

Contents

1	Introduction	1
1.1	Research Process	2
1.1.1	Use Case Development	3
1.1.2	Use Case Validation	4
1.1.3	IDE Extension	6
1.2	Outline	6
2	Foundations	9
2.1	History	12
2.1.1	Wearable Computers	12
2.1.2	Textile as a Substrate	14
2.1.3	Electronic and Textile Integration	15
2.1.4	Fibertronics	15
2.2	Fabrication	16
2.3	Smart Textile Modeling	19
2.3.1	Sensors	19
2.3.2	Output Devices	22
2.3.3	Connections	24
2.4	Applications	25
3	Requirements Specification	27
3.1	Requirements	27
3.1.1	Functional Requirements	28
3.1.2	Non-Functional Requirements	30
3.2	Related Work	31
3.2.1	Hardware Construction Toolkits	31
3.2.2	Development Tools for Physical Devices	33
3.2.3	Circuit Layout Software	34
3.2.4	Development Tools for Smart Textiles	35
3.3	Analysis	36
3.3.1	Model of a Smart Textile	37
3.3.2	Model of the Editor	38
3.3.3	Model of the Simulator	38
3.3.4	Model of the Deployer	39

CONTENTS

4	TangoHapps Framework	41
4.1	Running Engine	41
4.2	Happ Object Library	42
5	System Design	49
5.1	Design Decisions	49
5.2	Architecture	52
5.3	Hardware/Software Mapping	54
6	Object Design	57
6.1	Editor	57
6.1.1	Palette	58
6.1.2	Toolbar	59
6.1.3	Canvas	60
6.1.4	Circuit Layout	61
6.2	Simulator	64
6.3	Deployer	65
6.4	Plugin Editor	67
6.5	Plugin Interpreter	69
6.6	Firmata Library	70
7	User Interface	73
7.1	Interactex Designer	73
7.1.1	Project Selection Screen	73
7.1.2	Editor Screen	75
7.1.3	Canvas	76
7.1.4	Circuit Layout	78
7.1.5	Simulator Screen	78
7.2	Interactex Client	79
7.2.1	Download Screen	79
7.2.2	User Applications Screen	80
7.2.3	Default Applications Screen	80
7.2.4	Application Screen	82
7.3	TextIT	82
7.4	Usage Example	82
7.4.1	Development of an Interactex Application	83
7.4.2	Development of a TextIT plugin	85
8	Applications	89
8.1	Application 1: KneeHapp Bandage	90
8.1.1	Problem	91
8.1.2	KneeHapp Bandage	92
8.1.3	Implementation with TangoHapps	98
8.2	Application 2: Custodian Jacket	108
8.2.1	Problem	108
8.2.2	Custodian Jacket	109

8.2.3	Implementation with TangoHapps	112
9	Evaluation	121
9.1	User Study 1: Novice Users	121
9.1.1	Results	122
9.2	User Study 2: Professional Smart Textile Developers	123
9.2.1	Study Results	124
9.2.2	Questionnaire results	124
10	Conclusions and Future Work	127
10.1	Future Work	128
A	Happ Objects	131
A.1	Events Methods and Variables	131
A.2	Palette Objects	141
A.3	Variables	141
A.4	Simulation Objects	150
A.5	TextIT Objects	156
A.5.1	Code Editor	163
A.5.2	Line Chart Displaying Jogging Signal	165
A.5.3	TextIT Plugin for Counting Peaks in a Signal	165
	Bibliography	173

CONTENTS

Chapter 1

Introduction

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

Mark Weiser

The first computing devices occupied an entire room and were so expensive that only governments and big corporations could afford them. Furthermore, only technicians with special training were able to use them. Miniaturization of electronics, reduction in hardware prices and advances in usability engineering have enabled the appearance of smaller, yet more powerful, cheaper and more usable computing devices. Computing devices evolved from personal computers to mobile phones and wearable devices. Each new generation of computing device is smaller and integrates more seamlessly into users' lives.

Mark Weiser foresaw that computing devices would disappear entirely from a user's perspective, becoming "*indistinguishable from the fabric of everyday life*". His metaphor has become a reality today, as electronic devices are being integrated in the fibers used to create fabrics. For example, tiny electronic components can be hidden inside yarn by wrapping fiber strands around them [101]. Textiles with integrated electronic devices are called *smart textiles* and are the focus of this dissertation.

Smart textiles worn on the body, also called *smart garments*, are particularly interesting if they access information about their wearer such as his/her posture [79], physical activity [45], heart rate [19], pH levels [92] or even the presence of an infection [1]. This information can be used for monitoring vital signs of patients [94], soldiers [95], firefighters [22], tracking athlete's performance [53] and helping blind users avoid obstacles [6]. Smart textiles enable new ways of interaction that can be integrated seamlessly into users' lives. For example, smart textiles do not require users to look at a screen (eyes-free) or to use their hands (hands-free) for interaction.

Several construction kits [51, 56, 102] and development environments [26, 50, 7] have been created, which lower the entrance barrier and reduce the time needed to develop physical electronic devices. Smart textiles, however, still have no established tools that support their development. Furthermore, smart textile developers are still

using tools originally designed for other purposes such as circuit layout software for conventional electronics and programming environments designed for general-purpose microcontrollers. Software development environments for smart textiles face the following challenges:

1. **The development of a smart textile is time consuming.** In contrast to the field of physical devices, ready-to-use textile components can rarely be bought. Instead, they usually have to be designed and fabricated manually and tailored to the textile. Textile sensors and connections often have to be created in the same fabrication process as the rest of the textile. In addition, the production of a smart textile is relatively risky due to the uncertainty during its creation, whether the choice of materials and structuring (e.g. sewing, knitting) pattern will fulfill the requirements of the application.
2. **The development of a smart textile requires knowledge in multiple disciplines.** Smart textile development encompasses activities of different nature: textile design (e.g. choosing a sewing pattern), circuit design (e.g. deciding on the shape and placement of components) and software development. This knowledge is rarely present in a single individual and hence, a collaboration of people with a background in different disciplines is required. These disciplines include computer science, electrical engineering, material science and textile design [18].
3. **Smart textiles require the development of complex algorithms.** One of the facts that make smart garments so interesting is that they can provide diverse kinds of information about the wearer. However, high-level information needs to be extracted from the sensor data using complex signal processing and classification algorithms. Furthermore, data produced by smart garments tends to be noisy and unreliable for two reasons. First, garments change their position due to the wearer's movements when they slide and fold during usage. Second, textile sensors and connections based on conductive yarn or thread typically used in smart textiles are not as accurate and stable as traditional silicon-based sensors and connections.

To tackle these problems, we have developed *TangoHapps*. *TangoHapps* is an Integrated Development Environment (IDE) specifically designed for the development of smart textiles.

1.1 Research Process

The field of smart textiles is expected to evolve in the near future, in particular with sensing, heating and lighting applications [20]. In order to explore an evolving field, we applied a formative model-based research process. Formative approaches have the goal to form a technology or process by iteratively applying, assessing and improving it. Summative approaches, in contrast, do not intend to improve the technology or

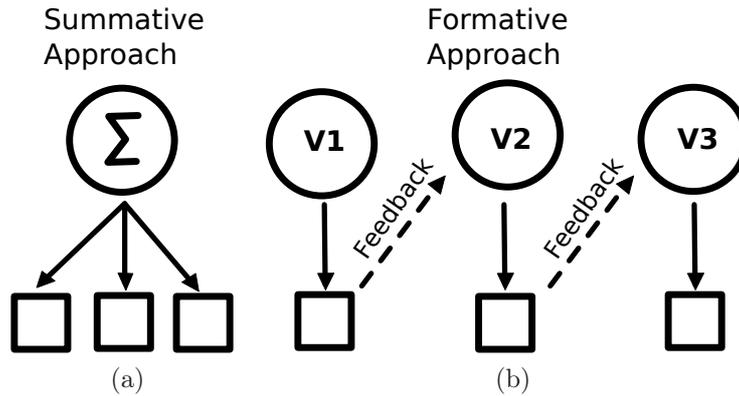


Figure 1.1: a) In the *summative* research approach the artifacts produced by a technology are assessed. b) In the *formative* research approach, the artifacts produced by a technology are used to assess and improve the technology.

process but rather assess the artifacts produced by them. The difference between the formative and summative approach is illustrated in Figure 1.1. In the context of education, a formative approach would assess a teaching methodology with the goal to identify possible improvements to the curriculum. A summative approach, on the other hand, would be an examination written by students at the end of the course to assess how much they have learned.

The formative assessment approach we applied consists of three phases: *Use Case Development*, *Use Case Validation* and *IDE Extension*. We applied these three phases iteratively. In each iteration, we refined *TangoHapps*' abstractions to improve the goals of *TangoHapps*.

1.1.1 Use Case Development

The core requirements of *TangoHapps* were elicited based on the realization of a set of smart textiles developed in collaboration with researchers from the Universität der Künste (UdK) in Germany, Technische Universiteit Eindhoven and Universiteit Twente in The Netherlands, Aalto University in Finland, Telekom Innovation Laboratories and Philips¹. The smart textiles were chosen with two main criteria. *First*, we chose the smart textiles that required the longest development times, had the most functionality in terms of lines of code and used most sensors and output devices. *Second*, we chose smart textiles from different application domains in order to derive requirements that are applicable also in new application domains. Table 1.1 describes the smart textiles we chose for the core requirements elicitation. Note that these textiles were developed for the initial requirements elicitation, not with *TangoHapps*.

After the initial version of *TangoHapps* was developed based on the core require-

¹Part of this collaboration was funded by the European Institute of Innovation & Technology (EIT ICT) and is documented in the EIT ICT's "Connected Textiles" Activity Nr. 13087 under the "Smart Spaces" Action Line.

ments, we extended its functionality to support the development of two additional smart textiles developed at the Technical University Munich (TUM), the KneeHapp Bandage and the Custodian Jacket.

KneeHapp is a smart textile that tracks the rehabilitation progress of a patient after a knee injury. The bandage uses integrated motion sensors to measure a patient's performance at different rehabilitation exercises. For example, KneeHapp measures the amount of shaking of the leg while the patient performs a squat. An additional sock with an integrated textile pressure sensor is used to determine the duration of a one-leg hop - an important measurement used by orthopedists to assess whether the patient is ready to go back to sports.

The Custodian Jacket supports technicians during maintenance activities in a supercomputer center with the goal to improve their safety and work performance. The jacket uses motion sensors to determine user physical activity (e.g. walking, running, lying down on the ground) and proximity sensors to help technicians find servers in a rack. Both of these smart textiles and their implementation with *TangoHapps* are described in detail in Chapter 8.

1.1.2 Use Case Validation

After a use case was developed, we implemented it with *TangoHapps* and validated it using the following metrics:

- Coverage. The versions of the smart textiles that we developed with *TangoHapps* usually had less functionality than the smart textile developed without *TangoHapps*. An important metric to validate the use cases was the amount of functionality covered by the replication.
- Wearability. This metric is used to assess how comfortable the smart textile is to the user.
- Accuracy. This metric was used to validate the accuracy with which the smart textile could extract user information.
- Performance. *TangoHapps* facilitates the development of smart textiles by making high-level reusable functionality components available to developers. These functionality components are general-purpose and not designed for any specific use case. As a consequence, the generality might have caused a loss in performance. This metric assesses execution performance and energy consumption of smart textiles developed with and without *TangoHapps*.

We used different research techniques for the validation of *TangoHapps*. One technique included case studies to estimate degree of functionality coverage. Another used semi-structured interviews with users and application domain experts to inquire about wearability and usability of the application. A third one shadowed developers while replicating the smart textile. We also conducted controlled experiments with users to estimate accuracy.

1.1. RESEARCH PROCESS

Name	Domain	Description
WaveCap	Music	A knitted hood and shawl that plays radio through a textile speaker inside the hood. The ends of the shawl - made of conductive yarn - close a switch when knotted together causing the radio to be turned on. The volume is controlled by pulling strings attached to the hood. Textile-based switch and sensor on the hood are used to control the radio sender and frequency.
Knit Alarm	Elderly	A knitted jacket for elderly patients to call for help in case of emergencies. Knit Alarm sends an emergency signal when users either pull the left sleeve or when they touch their right chest with the left sleeve. An analog textile sensor integrated in the left sleeve changes resistance depending on the deformation .
CTS-Gauntlet	Rehabilitation	A sleeve for patients of Carpal Tunnel Syndrome (CTS) that detects strain on the wrist. The sleeve triggers a visual and auditive signal when the strain goes beyond a certain threshold for a specific amount of time.
Shuffle Sleeve	Music	A knitted sleeve that plays music. A handful of coins is inserted in the sleeve - which has the shape of a circular tube. Rotating the sleeve causes coins to fall due to gravity, closing textile switches that control the playlist and volume. Four conductive strings can be knotted together to start, stop and pause the music.

Table 1.1: Smart textiles used to elicit the initial requirements of *TangoHapps*.

1.1.3 IDE Extension

The research process we followed in order to develop *TangoHapps* was iterative. We developed *TangoHapps* in a series of sprints based on the new requirements of the smart textile developed that we identified together with smart textile developers from the Universität der Künste (UdK) in Berlin. In each sprint, we refined and extended *TangoHapps* with respect to three main aspects.

First, we added support for the hardware devices used in each smart textile and developed an interface to control them. We also added common operations that developers might want to perform on the hardware device, even if these operations were not required by the smart textile we were supporting in each iteration.

Second, we identified software functionality that repeated across the different smart textiles and refined the abstractions to cover every smart textile. For instance, the functionality that enabled the *Custodian Jacket* to detect user physical activity (e.g. running, walking, climbing, not moving) is currently available in *TangoHapps* as a built-in type called “*Activity Classifier*”. Similarly, the signal processing functionality components (e.g. Peak Detector, Deviation Extractor) were added to *TangoHapps* in order to support the development of *KneeHapp*.

Third, in addition to adding functionality that enabled the development of each smart textile, we continuously refined *TangoHapps*’ user interface and visual programming semantics based on feedback provided by smart textile developers from the UdK. We applied the continuous delivery process described in [66, 64]. We delivered the latest release of *TangoHapps* during every sprint several times. The smart textile developers tested it and provided feedback, usually new functionality requests or ideas for usability improvements. Feedback provided by end users was stored in a repository in the form of backlog items and implemented during the next sprint.

1.2 Outline

This dissertation is divided in seven Chapters. Chapter 2 provides an overview of the field of smart textiles, including definitions, their evolution from wearable computers in the early 90s and describes different types of technologies used to construct smart textiles and give an overview of common smart textile applications. The concepts, techniques and processes provided in this chapter set the basis for the requirements elicitation process we followed in order to develop *TangoHapps*. Chapter ?? compares *TangoHapps* with similar development tools for smart textiles, wearable computers and other physical devices. Furthermore, it describes the core requirements and analysis models of *TangoHapps*. Chapter ?? serves as a starting point for the design of *TangoHapps*. Chapter ?? provides a detailed description of the decisions we made in the design of *TangoHapps*. It also describes the IDE’s architecture, the mapping of the software components to hardware nodes and how the functionality of each software component is organized in different classes. Chapter 7 focuses on *TangoHapps*’ user interface. It describes the dynamic model of the user interface and provides

an example to demonstrate step-by-step how an application can be created with the IDE. Chapter 8 demonstrates the applicability of *TangoHapps* in the development of the *KneeHapp Bandage* and *Custodian Jacket*. We chose these smart textiles for two reasons. First, because we had access to real-world projects with real customers, end users and target environments. Second, because we wanted to assess the applicability of *TangoHapps* to different application domains. Chapter 9 presents the results of two user studies to gain insight into the usage of *TangoHapps*.

CHAPTER 1. INTRODUCTION

Chapter 2

Foundations

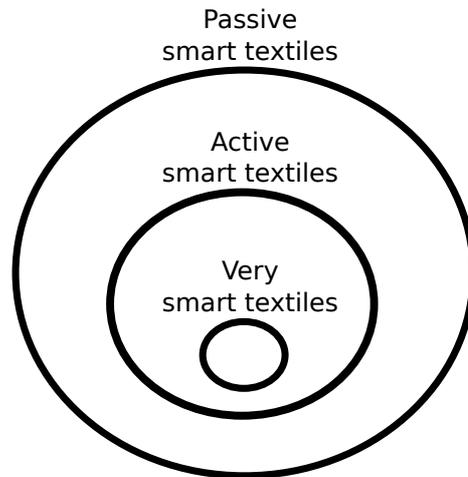
We elicited the requirements for *TangoHapps* based on a series of interviews with professional smart textile developers. In this chapter, we provide the background information needed to understand the requirements elicited. We provide different definitions of smart textiles and describe ways that have been presented in the literature to categorize them. We provide a historical overview of smart textiles including their evolution from the wearable computers and describe the types of materials, sensors and output devices that are currently being used to create smart textiles. Furthermore, we mention strategies and technologies to add sensing and other “smart” capabilities to traditional textiles and provide examples of applications developed based on smart textiles.

Smart textiles, also known as *intelligent textiles*, *electro textiles* or *e-textiles*, are textiles that have the ability to sense and respond to stimuli of diverse nature, including mechanical, electrical, thermal, chemical and optical environmental stimuli [18][124]. Smart textiles can be wearable (garments) and non-wearable (carpets, curtains). Wearable smart textiles are called *smart garments* or *smart clothing*. *Functional textiles* are textiles with a specific function added, for instance, by using additives or coatings [20]. Examples of functional textiles include fire-resistant or phase changing¹ textiles. Smart textiles are sometimes defined to include functional textiles. However, this is technically incorrect because smart textiles have *special properties* (e.g. electrical and optical conductivity) or *special functionality* (e.g. the ability to sense mechanical stimuli such as pressure) which is not present in functional textiles.

Smart textiles have been categorized according to the extent of their “intelligence” into *passive*, *active* and *very smart* textiles [119]:

¹Phase-changing materials are materials that release energy when in the process of changing its phase (e.g. melting or freezing)

- *Passive* smart textiles are textiles that can sense the environment.
- *Active* smart textiles are textiles that can sense stimuli from the environment and react to them.
- *Very* smart textiles are textiles that can sense, react and adapt to the environment.



Smart garments can be compared to other wearable devices according to their level of integration with the human body. Steve Mann categorized devices according to their wearability (or portability) into: environmental intelligence (e.g. cameras and microphones installed in a building), hand-held devices (e.g. mobile phone, tablets), in-pocket devices (e.g. smart card), clothing, underclothing and implants [75]. We refine Mann’s categorization into devices that are *carriable*, *wearable*, *attachable* or *insertable* in the body, as shown in Figure 2.1.

The lowest level of integration consists of devices that users *carry*, such as hand-held devices. Devices that are *worn on* the body include smart garments and wearable computers. Wearable computers are fully functional, self-powered and self-contained computer worn on the body [10]. The difference between smart garments and wearable computers is that the electronics of a smart garment are integrated in the textile, whereas wearable computers have no integration to the garments of the wearer [20]. The third level of integration includes devices that are *attached to* specific parts of the body, such as smart contact lenses and hearing aids. These devices are added to parts of the body without surgical intervention. Attachment and de-attachment of devices at this level of integration is usually done by the user. The strongest level of integration consists of devices that are *inserted into* the human body such as smart tattoos, prostheses and smart implants. The insertion and removal of such devices is done by specialized individuals and usually requires surgery.

Smart textiles have also been categorized according to the way in which the electronic components are integrated into the textile [20]. Wearable computers are electronic systems including sensors and computational components that are built with standard off-the-shelf components. Although wearable computers are considered to be a *predecessor* of smart textiles, there is no integration between the electronics and the textile [20, 76]. The *first* generation of smart textiles was similar to wearable computers in that it used the textile only as a substrate for attachment of electronics (e.g. sensors, output devices or printed circuit boards). The textile itself did not have any special functionality. The *second* generation started to integrate electronics into the textiles and adapted traditional textile fabrication methods for adding

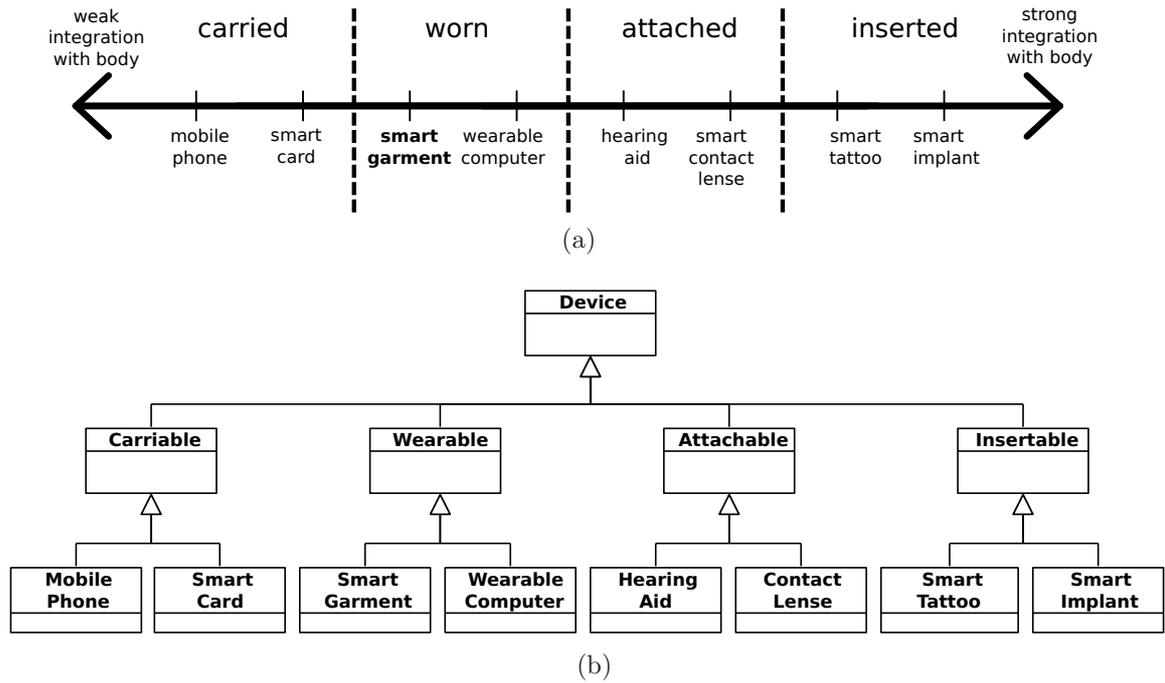


Figure 2.1: a) Smart garments compared to other devices according to their integration with the human body, based on previous work from Steve Mann [75]. b) Wearable devices represented as a taxonomy.

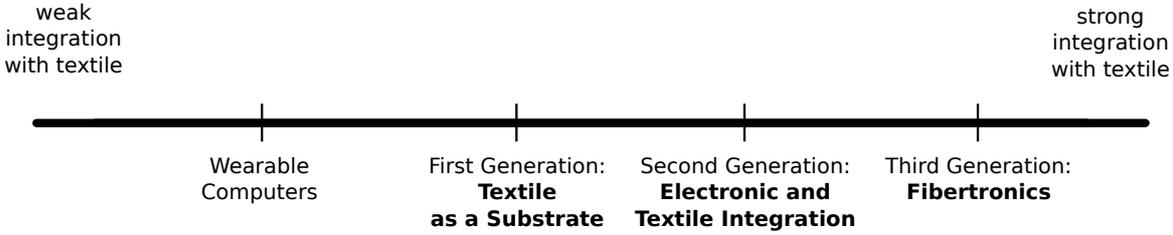


Figure 2.2: Classification of wearable computers and smart textiles according to the degree of integration of the electronics into the textile.

special functionality to the textile [20]. E-broidery is a textile fabrication technique which uses embroidery machines adapted to sew or weave conductive textiles using a numerically controlled process [99]. In the *third* generation, electronics and materials with sensing properties are being integrated directly in the fibers of the textile. This technique is also referred to as *fibertronics*. Figure 2.2 displays the different categories ranging from weak to strong integration into the textile. In the following section, we discuss these categories in more detail.

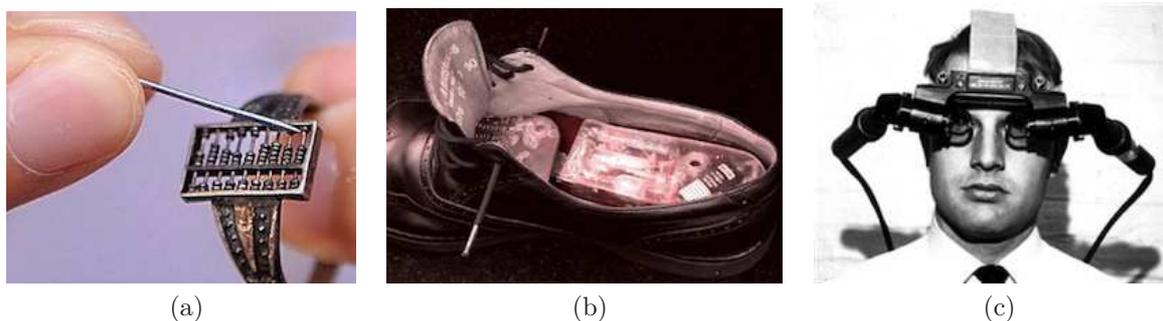


Figure 2.3: a) Qing Dynasty’s abacus ring. Image taken from <http://www.chinaculture.org>. b) Eudaemonics’ shoe used to cheat at the casino. Image taken from: <http://eyetap.org/wearcam/eudaemonic/> with permission of Steve Mann. c) Ivan Sutherland’s HMD [116]. Communications of the ACM 2002, Vol. 11:2. Copyright ©2002 by ACM, Inc. Reprinted with permission of ACM, Inc.

2.1 History

2.1.1 Wearable Computers

Wearable computers have a longer history than smart textiles. Wearables with “computation” capabilities date back to the 17th century. A silver ring with an inlaid abacus created by the Qing Dynasty that has recently been found is considered by some to be the first wearable “device” in history. The ring is shown in Figure 2.3 a).

Wrist watches emerged in the 19th century because pocket watches became impractical in some situations such as while riding horses or driving cars². Wrist watches became popular later on during the war at the end of the 19th century and beginning of the 20th century mainly due to the need to coordinate attacks during battles.

In 1957 Earl Bakken invented the first wearable pacemaker [59, 67]. The pacemaker’s pulse generator and battery were worn by the patient and the leads (the parts that conduct the electrical impulses into the heart) were implanted through the tissue inside the heart.

Most people consider Edward Thorp and Claude Shannon’s smart shoe for cheating at a casino as the first wearable device in history [122]. The shoe was developed in 1961 and had switches controlled by the toes that enabled the user to time when a roulette’s ball crossed a reference line. The electronics in the shoe were connected over a wire to a computer hidden in a cigarette box. The computer in the cigarette box communicated wirelessly with a receiver device that provided auditive output to the user. A similar shoe was created later by a group that called themselves the ‘Eudaemonics’, shown in Figure 2.3 b).

The first *Head-Mounted Display* (HMD) or *Heads Up Display* (HUD) was developed in 1968 by Ivan Sutherland. The HMD recognized the user’s head position and

²<http://www.smithsonianmag.com/innovation/pocket-watch-was-worlds-first-wearable-tech-game-changer-180951435/?no-ist>

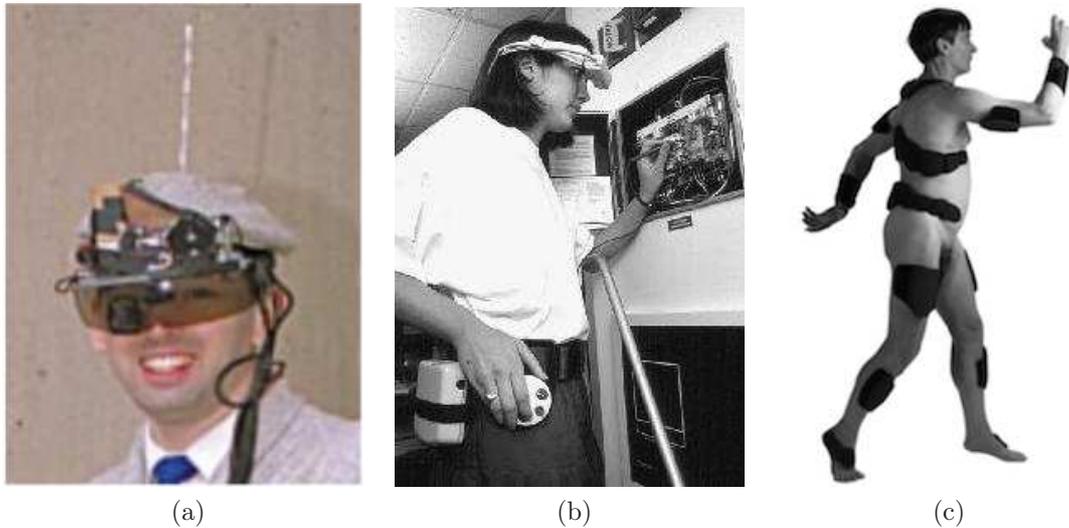


Figure 2.4: a) Steve Mann’s HMD [73]. Reprinted with permission of Steve Mann. b) CMU’s VuMan 2 wearable computer [110]. Reprinted with permission of Daniel Siewiorek. c) Forms for wearability [37]. Reprinted with permission of Francine Gemperle.

rotation within a room using an arm that hanged from the ceiling of the room. The HMD is also widely considered to be the first *Augmented Reality* HMD [116], although the term was defined a decade later as a system in which 1) real and virtual imagery are combined, 2) is interactive in real time and 3) is registered in three dimensions [5]. *Virtual Reality* is a system that “senses the participant’s position and actions and replaces or augments the feedback to one or more senses, giving the feeling to the user of being immersed or present in a virtual world” [104]. Sutherland’s head mounted display is shown in Figure 2.3 c).

Until 1980 wearable devices were not general purpose, but rather bound to specific use cases or testing environments. In 1980, Steve Mann created a general purpose multimedia wearable computer [74]. Mann attached a camera, a display and two antennas to a helmet and connected them to a 6502 computer that was carried in a backpack. During subsequent years, Mann continued shrinking the size of his wearable device and transmitted live images from his head-mounted camera to the internet for the first time in history [73]. An early version of Mann’s device is shown in Figure 2.4 a).

Research on wearable computing gained more attention after 1990 and was led primarily by institutions such as Carnegie Mellon University (CMU), Massachusetts Institute of Technology (MIT), Georgia Tech and Columbia University. Technologies used to develop wearable computers between 1990 and 2000 included HMDs such as the *Private Eye*, speech recognition units, cameras, GPS, wearable mice and keyboards. Popular application fields of wearable computers during this period were navigation and maintenance of vehicles and machinery.

Relevant wearable systems created after 1990 include CMU’s VuMan and Navigator series, Steve Mann’s backpack-based multipurpose wearable computer and the

Java Ring from Sun Microsystems. The VuMan and Navigator series were developed between 1991 and 1996. Vuman 1 included a HMD and a unit fixed to the user's belt with three buttons and was used to navigate the user through the blueprints of a building. VuMan 2 displayed maintenance information and helped the user locate people and buildings. VuMan 3 provided maintenance information during inspection of amphibious military tractors. Navigator 1 consisted of a computer mounted on a backpack, the *Private Eye* HMD, a speech recognition unit, a portable mouse for user input and a GPS for position tracking. Navigator 2 supported maintenance workers during aircraft inspection. An image of the VuMan 3 is shown in Figure 2.4 b). The Java Ring was a ring introduced by Sun Microsystems in 1998 [23]. The Java Ring had an integrated microprocessor able to run Java scripts and had no internal power but received power when it was in contact with the receiver device. One of the use cases of the Java Ring was unlocking doors after identifying the wearer with a unique identifier integrated within the ring.

In 1991 Mark Weiser coined the term *Ubiquitous Computing* or *ubicomp* referring to a vision where computing devices embedded in everyday artifacts support people in daily activities and work [65, 127].

In 1997 the first International Symposium in Wearable Computing was co-hosted by CMU, MIT and Georgia Tech. Most wearable devices presented during the event were based on Head Mounted Displays and eye glasses [24, 34, 107, 111, 121, 86, 73, 113]. Voice was commonly used as an input strategy [24, 109]. Application fields included blue collar working [24, 86, 91, 107, 121], navigation [34], sports [93], affective wearables [97] and disabled support [113]. Research presented at the conference focused on usability of wearables, such as wearable keyboard based input [120] and haptic output based on vibrations [118]. During the conference, Post and Orth introduced the concept of *smart fabric* together with some techniques to create textile connections and sensors [98].

In a study published in 1998, Gemperle et al. studied *wearability* (i.e. the interaction between a wearable device and the body) [37, 58]. The study provided guidelines for the design of wearable devices. For example, the study identified the positions around the body where wearable devices can be placed based on the size and amount of movement in the different body parts. The guidelines led to the term *wearable forms*. A wearable form is proven ways how wearable devices can be mounted on the body. Wearable forms are shown in Figure 2.4 c).

2.1.2 Textile as a Substrate

The *first* generation of smart textiles used the finished *textile as a substrate* for integration of electronic devices. The textile itself did not play any role besides that of a carrier for electronic devices and connections. Examples of this category include Georgia Tech's Wearable Motherboard (GTWM) and Virginia Tech's Beam-Forming textile [87]. GTWM, shown in Figure 2.5 a), was a soldier's vest able to detect bullet wounds and monitor soldier's vital signs during combat [20, 95]. GTWM had an integrated grid of optical fibers and attached off-the-shelf sensors. Virginia Tech's



Figure 2.5: a) Georgia Tech's Wearable Motherboard (GTWM) [95]. Copyright ©2002 by ACM, Inc. Reprinted with permission of ACM, Inc. b) *Musical Jacket* [99]. Reprinted with permission of Maggie Orth.

Beam-Forming textile was used to estimate the position and direction of moving vehicles [20, 87]. For this purpose, it contained several microphones attached to a woven grid of interconnection lines.

2.1.3 Electronic and Textile Integration

In the *second* generation of smart textiles, the textile played an essential role in the electrical system, rather than just being a substrate for attachment of electronic devices. Traditional textile fabrication techniques (e.g. embroidery) were adapted to provide the textile with special functionality [20]. Another characteristic of this generation was that it merged electronic design techniques with technologies from the textile industry. For example, the *Musical Jacket* - developed by MIT in the late 1990s - used Computer-Aided Design (CAD) to specify the circuit layout and stitch pattern of a textile keyboard, which was embroidered in the jacket. The jacket and embroidered textile keyboard are shown in Figure 2.5 b).

2.1.4 Fibertronics

The *third* and most recent generation, *fibertronics*, is characterized by the integration of electronics and materials with special properties into the textile during the fabrication of the textile. One way to achieve this is by replacing traditional fibers (e.g. cotton) with conductive or sensitive fibers in the creation of yarn. Another approach is the integration of miniaturized electronics during the fabrication of yarn by twisting strands of fiber around an electronic device [128, 101]. Zysset et al. developed a technique to create woven textiles with electronic circuits [132]. Electronic circuits are

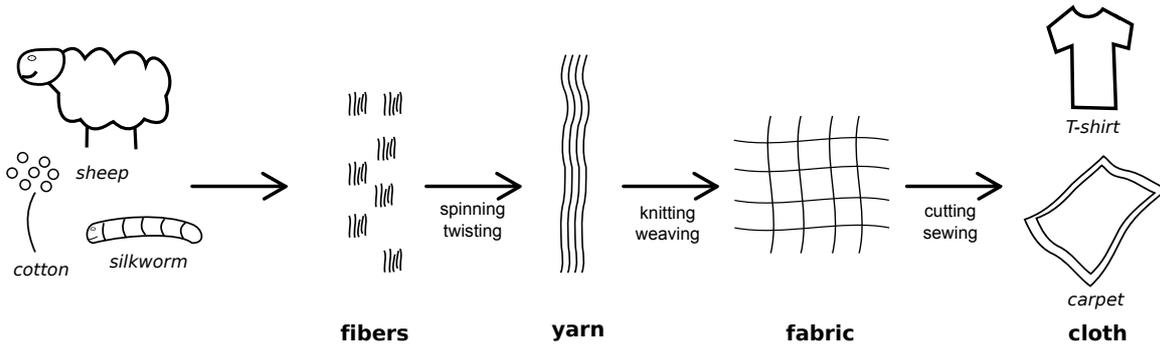


Figure 2.6: Textile fabrication process.

integrated in thin plastic film lines which are woven with conductive and traditional yarn lines.

2.2 Fabrication

The textile fabrication process consists of different phases which may vary depending on the materials used and the purpose of the textile. In the beginning of this chapter, we described *what* smart textiles are. In this section, we describe *how* smart textiles are created. We start by describing how traditional textiles are created and then explain how special functionalities are added. It should be noted that the textile fabrication process described in this section has been simplified for the purpose of this dissertation - textile fabrication processes include other phases and activities that are not shown in this section. The textile fabrication process we describe in this section is illustrated in Figure 2.6.

Textiles are hierarchically structured fibrous materials [18]. *Fibers* are the smallest unit in the textile hierarchy [18]. Fibers are raw materials which are either extracted from natural sources or created synthetically. Natural sources of fibers are either animal (sheep, silkworm), vegetable (cotton, flax) or mineral (asbestos). Fibers with no further processing have little tensile strength which limits their use to stuffing objects such as pillows and jackets. For this reason, fibers are spun into long strands (called *spinning*) which are then *twisted* together in order to form *yarn*. *Thread* is a type of yarn which is usually finer and suitable for sewing. *Fabric* is created by structuring yarn in specific patterns. Two popular ways to structure yarn into fabric are *knitting* and *weaving*. Knitting is the process of intertwining two sets of yarn in a series of consecutive loops [20]. Weaving is the process of interlacing two sets of yarns perpendicularly [20]. Fabric panels are cut in parts and different parts are sown together in order to form *cloth*. The terms cloth and textile are often used interchangeably. In this document, we use the definitions provided by Castano et al. [18]. We refer to “cloth” as the finished textile product and treat the term “textile” as fibers in any phase of the textile hierarchy.

Special functionality can be added to the textile at any phase of the fabrication process (i.e. by making modifications to the fibers, yarns or fabrics used to create

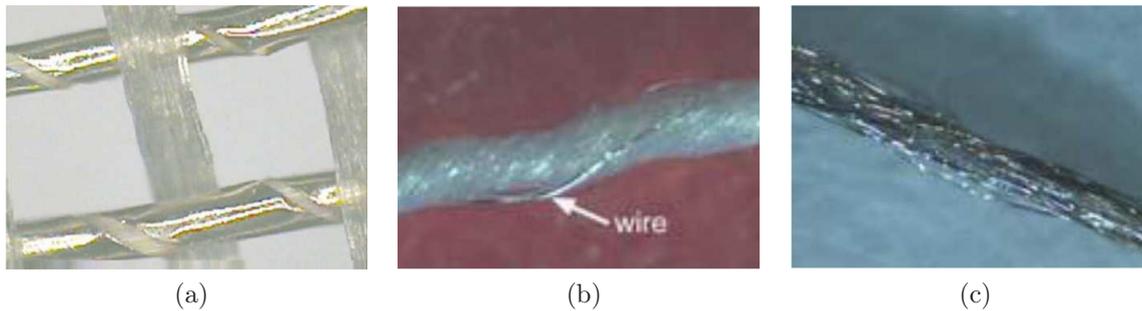


Figure 2.7: Different types of conductive yarns. (a) A copper foil wrapped around a non-conductive yarn. Reprinted with permission of Maggie Orth of *International Fashion Machines*. (b) A wire wrapped around non-conductive fiber strands [114]. Licensed under *Creative Commons BY 4.0*. (c) Multifilament yarn coated with a metallic layer [114]. Licensed under *Creative Commons BY 4.0*.

cloth). **Fibers** can be naturally conductive or be treated to become conductive [83]. Naturally conductive fibers are made from metallic materials such as stainless steel, titanium or aluminum. Metallic fibers can be created by “shaving-off” fibers from the edge of a thin metal sheet [83]. Electrically conductive fibers can be created by coating non-conductive fibers with metals, galvanic substances and metallic salts [83]. Metallic fiber coatings produce highly conductive fibers. However, it can be challenging to achieve a metallic coating that adheres to the fiber and does not corrode [83]. Galvanic coatings are coverings rich in zinc that provide high conductivity and corrosion resistance but can only be applied to conductive substrates [83]. Metallic salt solutions achieve low conductivity which is further reduced during laundry [83]. Fibers can be also mixed with materials sensitive to mechanical or chemical stimuli [18]. For example, piezoresistive materials³ can be used to coat fibers in order to make them sensitive to strain [18][54]. We discuss textile based sensors in more detail in Section 2.3.1.

Yarns can also be made conductive and sensitive to stimuli. Conductive yarns are created using either fully metallic wires or by combining traditional yarn with conductive materials. Metallic wires tend to have a high conductivity but are less flexible and therefore more likely to break during weaving and knitting [20]. Yarn and conductive material can be combined in different ways, including: wrapping a metallic foil around a non-conductive yarn [99], twisting non-conductive fiber strands around a metallic core [83] or by twisting a metallic wire together with non-conductive fiber strands [114]. Like fibers, yarns can be made conductive by coating them with conductive metals or polymers [83] and sensitive, by wrapping sensitive fibers around an elastic core [18]. Figure 2.7 displays different types of conductive yarn.

Special functionality can also be added to **fabric** after its creation. A common way to achieve this is by attaching conventional electronic devices (e.g. microcontrollers, memory units, wireless links) to interconnect lines in the textile [20]. Electronic devices can be attached to circuits in the textile using cables, conducting adhesives,

³Piezoresistive materials react to mechanical stress by presenting a change in the electrical resistance, which can be measured

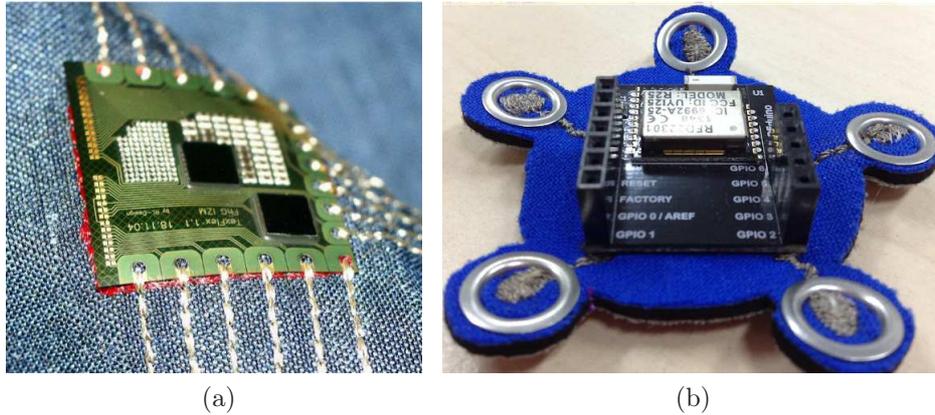


Figure 2.8: Electronic devices attached to fabric. (a) Flexible electronic module glued to the fabric and connected with embroidered conductive lines developed by the *Fraunhofer's Institute for Reliability and Microintegration* [71]. (b) Microcontroller connected with conductive thread to conductive snap buttons crimped. Developed in collaboration between the *Technische Universität München* and the *Universität der Künste* [45].

conductive thread or by means of mechanical methods. Cables add rigidity to the textile and are likely to get entangled or break due to strain in areas of the body where movement occurs. The Arduino Lilypad [15] hardware kit elements feature broad circular pins to facilitate their attachment to the fabric using conductive thread. The same conductive thread is used to connect the circuit. Mechanical methods (e.g. crimping⁴) apply force on the electronic devices or conductive elements in order to attach them to the textile. Figure 2.8 shows electronic devices attached to a fabric using different techniques.

Other techniques to add capabilities to the fabric after its creation include lamination and coating. Lamination is the process of adding a layer (e.g. a film) with special functionality (e.g. an LED display) to a fabric substrate, usually by applying adhesives [20]. Leah Buechley has attached electronics by ironing them to the fabric using heat-activated adhesive [15]. Coatings can be used to add conducting and sensing properties to fabrics [18]. For example, polymer coatings have been used to create fabric sensitive to temperature [11] and strain [70].

⁴Crimping: deforming one or two pieces of metal such that one can hold the other

2.3 Smart Textile Modeling

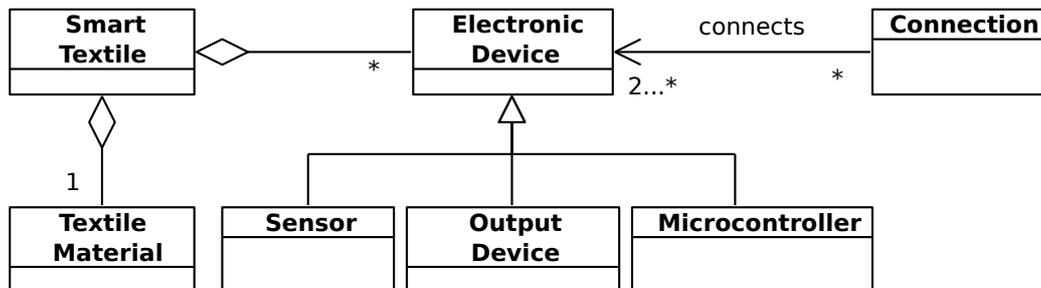


Figure 2.9: Model of a smart textile.

Current generations of computing devices such as personal computers and mobile phones are constructed on rigid plastic substrates. The trend in smart textiles is to replace the traditional silicon-based plastic substrates with textile-based components and connections that integrate more seamlessly into the textile. In this section, we provide an overview of the different types of textile-based components and connections used in smart textiles.

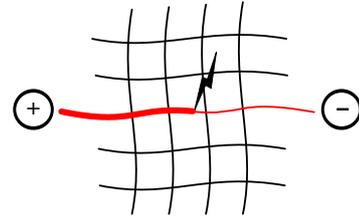
Figure 2.9 displays an UML model of a smart textile. Smart textiles consist of textile materials and electronic devices. Electronic devices can be sensors, output devices and microcontrollers. Connections transmit signals and power between two or more electronic devices. An electronic device can be connected to several other electronic devices. In particular, electronic devices usually have two or more connections to a microcontroller.

2.3.1 Sensors

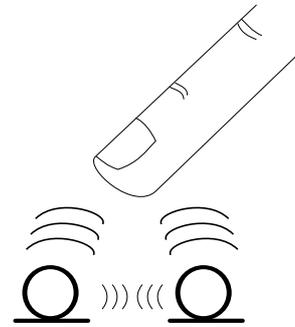
Sensors devices measure physical properties and transform them into digital signals that can be processed [20]. Physical properties include parameters from the environment such as temperature, light intensity and physiological parameters from humans such as heart rate. Most textile sensors operate following one of three principles: *resistive*, *capacitive* and *optical* sensing. In the following sections we describe these sensing principles and different sensor types.

Sensing Principles

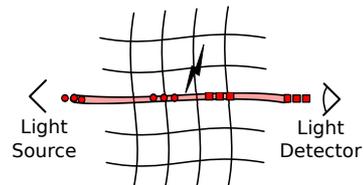
Resistance is the difficulty of an electrical conductor to conduct an electric current. *Conductivity* is the opposite of resistance. **Resistive sensing** works by measuring the resistance of a material, which changes depending on the physical property to be measured. For example, a temperature sensor can be made with a specific type of fiber that change their resistance depending on temperature [20].



Capacitive sensing works by measuring the amount of electric charge transferred from two charged objects - called *electrodes* - into a third conductive object (e.g. the human finger). The amount of electric charge transferred between both electrodes changes depending on the parameters to be measured. This principle can be used to measure environmental parameters such as humidity and skin proximity - indeed, it is the technology used by most modern mobile phones to detect human touch. Capacitive sensing can be realized in smart textiles using conductive fabric and coatings [18].



Optical fibers are flexible and transparent strands of plastic or glass. Due to the transparency of optical fibers, light can travel through them. **Optical sensing** works by measuring different properties of the light transmitted through optical fibers (e.g. intensity, wavelength), which can change due to external stimuli such as strain. The GTWM used optical sensing to detect wounds in military scenarios by detecting cuts in the optical fibers [95].



Sensor Types

This subsection describes the different types of sensors that are used in smart textiles and presents examples of each sensor type. Figure 2.10 displays an UML model of textile sensors and their relationship with the environment.

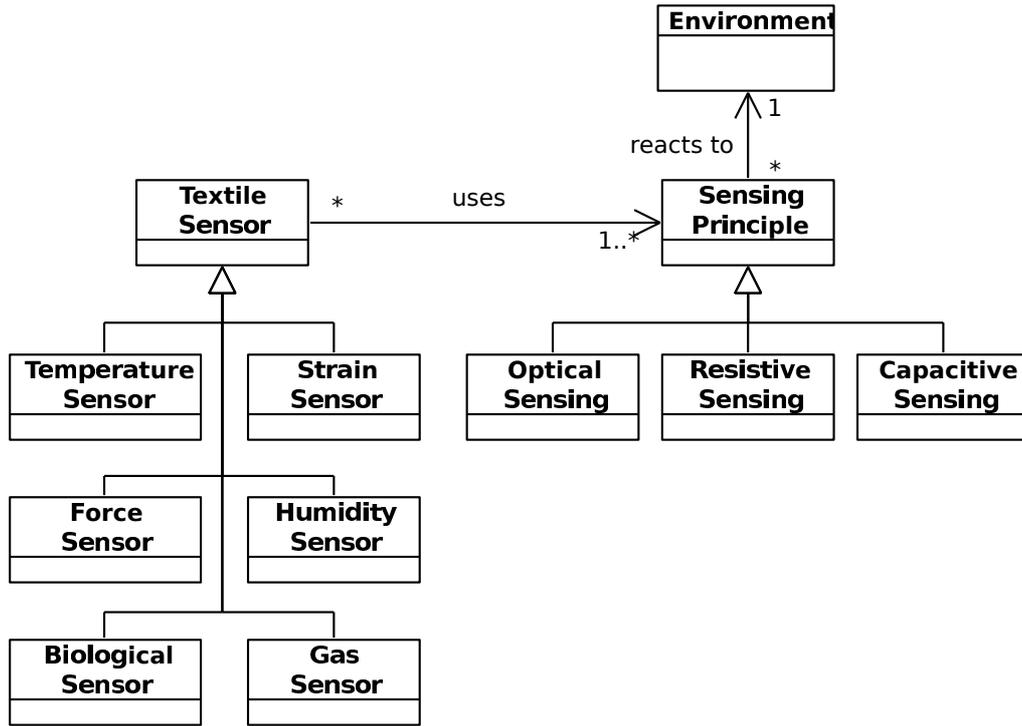


Figure 2.10: Taxonomy of textile sensors.

Pressure sensors, also called *force sensors* measure the amount of force applied to a surface. Pressure sensors integrated within textiles can be used as input sources such as buttons. They can also be used to monitor physiological parameters from a wearer. For example, pressure sensors can measure the weight distribution when integrated into socks.

Pressure sensors can be built using the resistive or capacitive principle. Resistive pressure sensors can be created by mixing conductive and non-conductive fibers or yarn in a textile and measuring the resistance of the material. When pressure is applied to the textile, the conductive fibers are squeezed, which increases the conductivity of the material. Another way to create a resistive pressure sensor is by using polymer coatings that generate an electric signal when pressure is applied to them [18, 13]. At a yarn level, specific structures of yarn can be created so that resistance of the textile structure changes when deformations are applied to it [57]. Capacitive pressure sensors have been achieved by separating two conducting textile surfaces with a spacer material and measuring changes in capacitance occurring when the conductive surfaces get closer to each other due to pressure [84]. Pressure sensors have been used to create keyboards [3], touchpads [20] and sportswear [53][115].

Heart rate sensors, also called *electrocardiogram* or *ECG sensors* measure the electrical change on the skin caused by heart contractions. Several textile heart rate sensors have been developed, such as the *MagIC* vest [27], the *NuMeTrex* sports bra [108] and the *Intellitex* suit [114]. **Respiratory activity sensors** are created using strain sensors that change their electrical resistance due to stretching of the thorax. The *Wealthy* suit used strain fabric sensors and piezoresistive yarns to measure users' breathing patterns [94].

Temperature sensors can be of resistive or optic types. Resistive temperature sensors work by measuring conductivity of metal fibers woven or knitted inside the textile, which change depending on the temperature [20]. Fiber optic sensors and temperature sensitive coatings have also been used for temperature sensing [18, 106]. Alternatively, thin-film temperature sensors can be integrated in yarn [20].

Strain sensors can be used to detect body postures or joint movements. This can be useful in sports and rehabilitation scenarios. Strain sensors can be resistive or optical. Resistive strain sensors can be created by knitting piezoresistive metal fibers [18]. Zhang et al. structured yarn in a special pattern that favors changes in resistance when strain occurs [130]. Optical strain sensors work based on the fact that the strain applied to the textile alters the strength of the optical signal traveling along the optical fiber. The strength of an optical signal can be measured by the optical sensor [20, 27].

Biological sensors detect the presence and composition of biological fluids such as sweat, urine and blood. These sensors are created using chemicals that react electrically to specific substances present in the biological fluids. Chemicals are either applied as a coating to textile fibers or screen printed⁵ on fabric [20].

Gas sensors detect the presence of gases such as hydrogen (H_2), carbon monoxide (CO) in an area [28]. Gas sensors are of particular interest in hazardous or contaminated environments (e.g. firefighting) because of their ability to detect the presence of toxic gases for humans or animals. Similar to biological sensors, gas sensors are realized using coatings that react electrically when exposed to specific gases [18]. The PROeTEX firefighter’s suit incorporated a CO_2 sensor within the boot and a CO sensor in the jacket next to the collar [21].

Humidity sensors can be resistive or capacitive. Resistive humidity sensors change their conductivity depending on the humidity in the air and capacitive humidity sensors react to water vapor [18]. Humidity sensors can also be built using coatings [18].

2.3.2 Output Devices

Output devices, also called *actuators*, are devices used to communicate changes in the smart textile to the environment [20]. Output devices provide mainly visual, auditive and haptic information to users. Figure 2.11 displays a taxonomy of output devices.

Visual output suitable for smart textiles include Light Emitting Diodes (LEDs), electroluminescent fabrics and chromic materials. Arrays of LEDs have been combined in a grid-fashion using conductive yarn in order to create a textile display [18]. Electroluminescent fabrics emit light when they receive an electrical stimulus and can be constructed by applying coatings with electroluminescent polymers [18]. Chromic materials are materials that change their color depending on external stimuli such as light (photochromic), temperature (thermochromic), pressure (piezochromic) or electrical (electrochromic). [18]. Chromic fabric can be obtained using chromic fibers or material coatings. Figure 2.12 a) displays a thermochromic display developed by the

⁵Screen printing is a technique to transfer ink into a textile by using a “mold” to fix the distribution of ink in the textile.

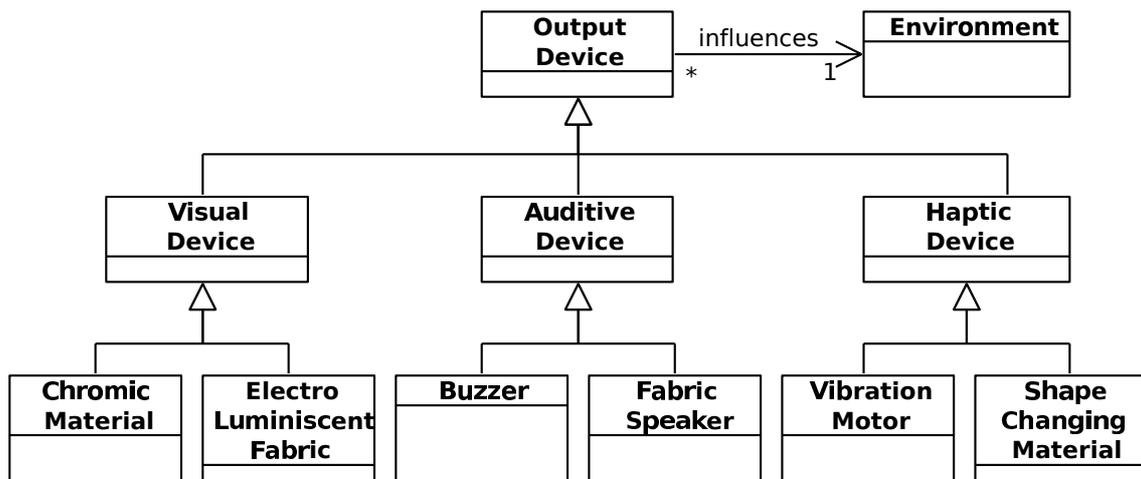


Figure 2.11: Taxonomy of output devices.



Figure 2.12: Textile output devices: a) Textile thermochromic display and b) textile speaker. Reprinted with permission of Katharina Bredies.

UdK. A chromic material coating has been applied to the textile. Wires behind the textile heat due electrical stimulus, causing a change in the color of the textile.

Smart textiles can provide **auditive output** using miniaturized off-the-shelf speakers. These speakers are smaller than a fingertip and can be sewn to interconnect lines on the textile. Speakers can also be constructed on fabric by arranging conductive yarn or ink in a spiral form. A magnet in the center of the spiral generates an electromagnetic wave which is intensified as the signal flows along the spiral. Figure 2.12 b) shows a textile speaker integrated into a sweatshirt developed by the UdK.

Haptic output technologies include miniaturized vibration motors and shape-changing materials (i.e. materials that change their shape upon certain stimuli). Shape-changing materials are either polymers or alloys. Electrically active polymers are materials able to change their shape or dimensions when they receive an electrical stimulus [17]. Shape-memory alloys and shape-memory polymers are materials that “remember” a shape to which they return when influenced by external stimuli such

as heat. Haptic output can also be produced using using motors. The Adidas smart shoe uses a motor connected to the cushioning system in order to adapt the cushioning depending on the surface on which the wearer runs [31].

Visual output is the most common output modality of many computing devices such as desktop computers and mobile phones. However, the usefulness of visual outputs on a smart garment is limited to the relatively small range of the wearer’s field of vision (FoV) - around the face and on the upper-front body. Instead, visual output in the case of smart garments is often used to address other individuals around the wearer of the smart garment. For example, a textile display attached to the back of the wearer has been used to communicate performance parameters to athletes during team sports [82]. Auditive output devices have to be placed at a position where they can be heard by the wearer. Haptic output is more discrete than the visual and auditive output modalities, therefore it is socially more acceptable. However, haptic output devices might not be suitable if the user is wearing several layers.

2.3.3 Connections

Connections for smart textiles can be realized in several ways. Figure 2.13 displays a taxonomy of connections.

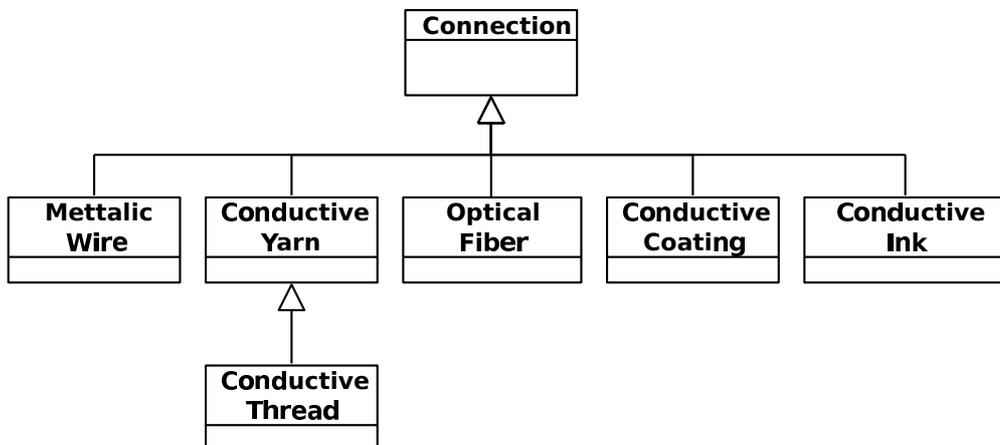


Figure 2.13: Taxonomy of connections used in smart textiles.

One way to connect electronic devices in a smart textile is by attaching metallic **wires** to the textile. However, wires tend to have a limited flexibility and to be less resistant to strain. Wires in smart garments which are subject to strain and movement have a higher risk to break or get entangled with objects in the wearer’s environment.

Conductive yarn and **conductive thread** can be woven into a textile using traditional weaving machinery. Conductive thread can be additionally sewn and embroidered into a textile [83]. Embroidery with conductive thread offers some advantages like the ability to create multiple layers of fabric in a single step and to specify circuit layouts using Computer Assisted Design (CAD) [99].

Connections with varying degrees of conductivity can be realized using **conductive coatings**. Some polymers used to create conductive coatings are even more conductive than metals and have also high adhesion and non-corrosive properties [83]. Additionally, coatings do not alter the textile properties (e.g. flexibility) significantly and are applicable to most textile substrates [18].

Conductive ink is created by adding conductive materials such as carbon, copper or silver to traditional printing inks [83]. Conductive ink can be applied to a textile using inkjet printers or by means of screen printing. Inkjet printers propel drops of ink into the textile [83]. Screen printing comprises the printing of a viscous pasta through a patterned fabric screen, which is followed by a drying process [114]. Screen printing is also suitable for integration of electronics in a textile [114]. In contrast to some conductive coatings, conductive inks do not lose conductivity due to bending or laundering [83].

Optical fibers are transparent fibers where signals are transmitted through pulses of light [18]. Optical fibers possess good strength and sunlight resistance and are in general not affected by electromagnetic interference [83]. Optic fibers are relatively stiff and possess poor flexibility, drapability (i.e. the ability to wrap loosely with folds around an object) and abrasion resistance [83].

2.4 Applications

Smart textiles have been developed for a wide range of fields. In this section we focus on three main areas: health and rehabilitation; sports and fitness; and security and safety. We provide examples of relevant research and industrial smart textiles developed in these domains.

Health and Rehabilitation

Several smart T-Shirts are able to detect physiological parameters from the wearer such as heart, respiration rate, electrocardiogram, activity and posture. Examples include the *LifeShirt* by Vivometrics [41] the *MagIC* system [27] and the T-Shirts developed in the EU-funded projects *Wealthy* [92, 94] and *MyHeart* [42].

Smart textiles for rehabilitation measure performance in different rehabilitation exercises and provide live feedback to the wearer or track the exercising over time [45, 4, 38, 2]. Most of these systems use motion sensors, some use the textile itself to provide feedback to the user [2]. The *UdK* collaborated with Philips on a sleeve worn on the wrist for rehabilitation of Carpal Tunnel Syndrome (CTS). The sleeve uses strain sensors to measure the range of motion in the user's wrist and contains blue light LEDs which can help relax muscles and increase blood flow. Zhang et al. [131] created a textile band able to react to muscle electrical signals (also called electromyography or EMG). The textile band classifies electrical signals and enables amputees to control their prostheses.

Sports and Fitness

Holeczeck et al. [53] developed a sock with integrated pressure sensors that tracks and classifies movements during snowboarding. A similar product is the *Sensoria Socket*⁶ which uses textile pressure sensors to track jogging and walking speed, pace and cadence of a jogger or walker.

Sundolm et al. developed a smart textile mattress with integrated pressure sensors that classifies physical exercises performed on the mattress such as push ups and squats [115]. A recent study investigated the use of conductive ink and flexible LED displays for supporting athletes during group sports [82]. Smart textile products have been introduced to the market by Nike, Polar and Adidas. Nike and Polar collaborated on a heart rate monitor that straps around the wearer's chest and Adidas developed a smart jogging shoes [31]. Another smart textile that reached the market was *NumeTrex*'s sports bra. *NumeTrex* used knitted conductive fibers as electrodes to monitor the wearer's heart rate [20].

Security and Safety

An early smart textile for safety was the *Georgia Tech Wearable Motherboard (GTWM)*. The *GTWM* had an integrated heart rate and temperature sensor and a network of optical fibers to detect possible wounds caused by projectiles in the vest. Another smart garment for safety was the *ProeTEX* firefighter's jacket. The *ProeTEX* jacket used several sensors to measure physiological parameters from the wearer (e.g. breathing rate, heart rate, body temperature) and other parameters from the environment such as the presence of toxic gases [22] in order to foresee dangerous situations. The University of Twente in The Netherlands developed a T-Shirt also for firefighters able to track the user's motion and interpret heart beats using accelerometer and microphone data ⁷.

⁶<http://www.sensoriafitness.com/>

⁷EIT ICT project. Action Line: "Connected Textiles". Activity Number: 13087. Year: 2014.

Chapter 3

Requirements Specification

This chapter consists of three sections. Section 3.1 describes the core requirements for a development environment for smart textiles. Section 3.2 provides an overview of state of the art tools for the development of smart textiles, wearable computers and similar physical devices. Section 3.3 presents the analysis models for a development environment for smart textiles that match the core requirements. The requirements and models presented in this chapter form the basis for *TangoHapps*, which we address in the next two chapters.

3.1 Requirements

In this section, we list the functional and non-functional requirements for an IDE for smart textiles. As described in Chapter 1, these requirements were elicited iteratively during a four-year collaboration with professional smart textile developers.

Throughout this dissertation, we refer to the users of *TangoHapps* as *developers* because they use *TangoHapps* to *develop* applications for smart textiles. The term *developer* refers to a *smart textile* developer and not necessarily to a *software* developer. Indeed, most smart textile developers have no or very limited experience in programming, such as individuals with a background in fashion or textile design. We use the term *user* to refer to target users of the smart textile and applications developed with *TangoHapps*.

It should be noted that the term *user* is not always equivalent to *wearer*. In most cases, the *user* is also the *wearer* of the smart textile. As described in Chapter 2, not every smart textile is wearable. In addition, the *user* of a smart garment application might be a different individual than the one *wearing* it [82].

3.1.1 Functional Requirements

FR1 Software functionality

Similar software functionality repeats across different smart textiles, such as:

- Transferring sensor signals to remote devices for processing.
- Filtering noise produced by resistive and capacitive textile sensors.
- Calibrating custom-made textile sensors.
- Extracting data from raw sensor signals (e.g. mean, deviation, correlation, peak detection).
- Recognizing physiological states such as human posture and motion (e.g. walking, standing, running, lying down).
- Showing plots of sensor signals for development purposes.

An IDE for smart textiles must offer this software functionality for reuse.

FR2 Hardware support

There exists a limited number of electronic devices suitable for attachment or integration onto smart textiles. Electronic devices include:

- Sensors able to measure environmental parameters (e.g. light intensity, temperature).
- Sensors that measure physiological parameters from the wearer of a smart garment (e.g. motion, heart rate, pressure applied on a specific surface).
- Output devices to communicate information to users in a visual, auditive and haptic manner (e.g. LEDs, speakers, vibration motors).
- Microcontrollers that execute the application (e.g. Arduino Lilypad, Intel's Curie).

An IDE for smart textiles must offer high-level APIs to control these electronic devices.

FR3 Smartphone support

Modern smartphones offer capabilities that can help complement those of a smart textile and are therefore considered as an important component of a smart textile IDE. Some capabilities of smartphones include:

- High resolution displays that can render text, images and videos.
- User interfaces with a variety of input mechanisms (e.g. sliders, switches, multi-touch gestures).
- Access to contacts and functionality to make calls.

- Access to music libraries and functionality to play music.
- Ability to use the camera to take photos and videos.

An IDE for smart textiles must enable the creation of applications that use capabilities available on the smart textile and on modern smartphones at the same time.

FR4 Circuit design

The design of electronic circuits for smart textiles is more challenging than the design of circuits for printed circuit boards. This is mainly due to the flexibility of the textile substrate and the strain it is subject to. Uninsulated textile connections can break or produce shortcuts when the substrate deforms or folds. Furthermore, the circuit design of a smart garment must additionally consider user comfort and wearability [37], adding constraints to placement, shapes and materials to use. An integrated development environment for smart textiles must facilitate the design of circuits. In addition, it should provide a model of a smart textile, in particular its layout so that its folding and usage patterns can be taken into consideration. Because a smart textile developer might not have experience in electronics, the IDE must additionally provide feedback about the correctness of circuits.

FR5 Simulation

When creating a smart textile, the choice of materials and structuring pattern (e.g. sewing, knitting) must fulfill the requirements of the application. Mal-functioning circuits and the unsuitability of data delivered by sensors might be discovered only after having created the smart textile and developed its application. Furthermore, the creation of a textile circuit might be time consuming because connections and custom-made textile sensors usually have to be created (e.g. sewn) manually.

Simulations allow flaws to be identified before considerable costs have been invested in a product. Therefore, an IDE for smart textiles must provide a way to execute applications within the environment and to simulate behavior of sensors and output devices. Smartphones might be used together with the smart textile. Therefore, the IDE must enable the simulation of a smartphone capabilities.

FR6 Debugging

In order to facilitate comprehension of runtime behavior of an application, an IDE for smart textiles must communicate runtime information to developers. In particular, the IDE should display sensor signals as plots and communicate the state of output devices during the simulation of an application. For example, the sound emitted by a speaker could be played within the environment and the vibration produced by a vibration motor could be communicated to developers by means of a visual animation.

FR7 Deployment

In *Circuit Stickers* [52] and *PaperPulse* [100] circuit layouts are printed on a sheet of paper using an inkjet printer filled with conductive ink. After printing the circuit sheet, developers attach the electronic elements to the sheet of paper. Most development environments for physical devices facilitate software deployment by generating source code, which is uploaded to a microcontroller either manually by users or automatically by the environment. An IDE for smart textiles should transform a development representation of the application (e.g. source code) into an executable application. Furthermore, the IDE should be able to upload the generated executable into the smart textile.

3.1.2 Non-Functional Requirements

NFR1 Low entrance barrier

Similar development tools for physical devices facilitate development by offering ready-to-use functions [39, 77, 30, 29, 12, 125, 52, 100, 72]. For example, Microsoft's *.NET Gadgeteer* offers software functions to perform the most common operations with the devices available in the kit (e.g. functionality to make the camera start recording a video) [125]. Developers reuse the provided software functions without having to understand their internal behavior. Target users of an IDE for smart textiles might not have experience in software development or electronics. In order to lower the entrance barrier to such users, the IDE should provide ready-to-use functions for common operations.

NFR2 Reusability across application domains

Low-level software abstractions might require a higher degree of expertise from developers. High-level application-specific software abstractions might be easier to grasp by beginners but often limit their freedom to specific applications. An IDE for smart textiles should offer enough flexibility to experienced software developers and have a low entrance barrier to users with less experience in software development.

NFR3 Performance

Hardware integrated into a smart textile is ideally small, thin and lightweight to preserve the positive properties of the textile (flexibility, softness, lightness). Computations performed on the smart textile hardware lead to more memory and processing power requirements and to bigger and bulkier microcontrollers and batteries. Software functionality available in an IDE for smart textiles should be optimized for execution on hardware with limited resources.

NFR4 Extensibility

In the Functional Requirements, we described software functionality an IDE for smart textiles should feature. In addition, the IDE must make it possible to

add new software functionality components. Furthermore, smart textiles are not widespread in the market yet and new target hardware platforms are likely to appear as the field grows. An IDE for smart textiles must make it possible to add new electronic devices to the development environment.

NFR5 Availability

Smart textiles might not be reachable over a wireless connection in some environments (e.g. in the metro). Smart textiles developed with the IDE must be functional independently of the environment.

3.2 Related Work

Our related work research encompasses development solutions for smart textiles and other physical devices. We categorized related work in four main categories: hardware construction toolkits, development tools for physical devices, circuit layout software and development tools for smart textiles. In this section, we describe each category and state how *TangoHapps* is different from any solution previously created.

3.2.1 Hardware Construction Toolkits

A hardware construction kit is a set of physical devices that users connect to each other in order to create an electronic device. These kits are usually accompanied by a software tool that facilitates the development of software using the toolkit's components. Components include sensors, output devices, communication modules (e.g. a Bluetooth module) that are connected to a microcontroller.

Phidgets was one of the first hardware device construction kits [39, 77]. Hardware components in the kit receive the name of “physical widgets” (i.e. *phidgets*). *Phidgets* are a mean to facilitate the development of physical user interfaces as an analogy to the UI widgets that developers reuse when building graphical user interfaces. The framework consists of the phidgets, a software API to access and control the phidgets and an architecture for communicating and connecting to the phidgets. *Phidgets* offers a way to simulate the runtime of the hardware device.

Input Configurator Toolkit is a toolkit used to create software for computers visually [30, 29]. Applications developed with the *Input Configurator Toolkit* accept inputs from a variety of computer peripherals such as keyboards, mouses and microphones. The *Input Configurator Toolkit* uses a flow-based programming paradigm to support the development of behaviors that react to user input (e.g. scrolling an application's window based on a keystroke).

Calder is an IDE for rapidly building physical devices similar to *Phidgets*, consisting of a variety of reusable input and output devices equipped with a wireless transmitter [68]. *Calder* supports product and interaction designers during early prototyping activities.

Papier Mache is a toolkit for building tangible user interfaces that use computer vision, electronic tags and barcodes [63]. The input sources sense the presence, removal and modification of objects and notify the application. Developers can choose how the application reacts to such events.

e-blocks is based on reusable off-the-shelf components that are plug-and-played into a main board [72]. A rule-based system defines the behavior of the *e-blocks* and a simulation environment enables developers to test the behavior before producing the actual device.

Midas is a software and hardware toolkit to support the design, construction and programming of flexible capacitive touch sensors that are attached to physical objects [102]. Developers use a high-level specification in order to define shape, layout and type of the areas sensitive to touch. After the touch sensitive area has been defined, *Midas* provides instructions to users for creating and assembling the parts using conductive ink and vinyl cutters.

Boxes is a toolkit consisting of easy-access materials like cardboards, foil, tape and thumbtacks to create very rapid prototypes (in seconds or minutes) [56]. *Boxes* can react to users pinning or unpinning thumbtacks to a sensitive material and features a tool to specify software reactions to user pinning actions.

iStuff is a toolkit of physical devices (e.g. buttons, sliders, speakers, buzzers, microphones) and a software framework to support the development of applications using such physical devices [8].

iStuff Mobile is a tool based on the *iStuff* environment that helps designers explore interactions with mobile phones [7]. *iStuff Mobile* makes use of a visual flow-based programming paradigm and enables development of applications that use mobile phones as output or input device to control other devices.

Amarino is a toolkit that enables the rapid prototyping of mobile ubiquitous computing applications [61]. *Amarino* consists of a mobile phone application, a set of tutorials and a library to facilitate the communication between the mobile phone and the microcontroller.

.NET Gadgeteer offers a set of hardware components that can be plugged to a motherboard [125]. The *.NET Gadgeteer* uses different types of connections to prevent users from connecting two electrically incompatible components, thus eliminating the risk of shortcuts. An IDE facilitates electronic layout and application development using the *.NET Gadgeteer* hardware components.

Arduino Lilypad is a construction kit for smart textiles [15]. The *Arduino Lilypad Kit* consists of conductive thread, needles, small batteries and hardware components. Its hardware components are lightweight, flat and feature broad pin holes. Thread - in particular conductive thread - is wrapped around the pin holes. Conductive thread fulfills the purposes of attaching the hardware components to the textile and interconnecting them at the same time.

Perner-Wilson et al. [96] propose the usage of craft materials for the creation of electronic textiles and present evidence about the increased freedom the approach gives to creators when compared to kits of parts.

This section has described toolkits that facilitate the assembly of hardware devices by plug and playing electronic devices. *TangoHapps* does not offer electronic devices but enables the design of the circuits and development of applications for smart textiles

using existing electronic devices.

3.2.2 Development Tools for Physical Devices

d.tools is a tool for the visual design and programming of physical devices built with off-the-shelf components [51]. *d.tools* divides its workflow in three phases: *design*, *testing* and *analyze*. During the *design* phase, designers drag and drop input and output elements and define links or transitions between them. Users can demonstrate transition events in the physical world in order to teach the tool desired behaviors. For example, to cause the tilting of a device to transition from one screen to another, the user selects the transition and performs the tilting. *d.tools* features a tight coupling between the digital and physical world. Plugging a sensor to the microcontroller board causes the sensor to appear in the canvas immediately. During the *testing* phase, users use the physical device while being recorded. The recording contains every state transition performed by users and can be later on reviewed together with a usage video during the *analyze* phase.

a CAPpella is a tool for building context aware applications by exploiting the programming by demonstration technique [25]. Programming by demonstration is a programming paradigm that enables a system to learn behaviors based on demonstrations performed by humans. *a CAPpella* records sensor data and displays a visual representation of the data on top of which users annotate and label time fragments. Data and user annotations are used for training a classification algorithm.

Exemplar is a tool similar to *a CAPpella* for programming input sensor devices using the programming by demonstration technique [50]. Users can grab a physical device and perform a gesture in order to program *Exemplar* to detect that particular gesture. *Exemplar* divides its workflow in three phases: *demonstrate*, *edit* and *review*. During the *demonstrate* phase, users demonstrate the usage of an input device. During the *edit* phase, users label and apply signal-processing algorithms to different parts of the received signal stream. During the *review* phase, users are able to test whether the system recognizes user gestures as expected.

VoodooSketch is a toolset for customizing physical interfaces by drawing on a sheet of paper with a smart pen [12]. *VoodooSketch* allows physical devices to be plugged into the sheet of paper. Next to the physical device, a label can be written with the smart pen. *VoodooSketch* digitalizes the content of the pen automatically and associates physical devices with specific software functions.

Circuit Stickers is an IDE to create electronic circuits on paper using an inkjet printer filled with conductive ink [52]. *Circuit Stickers* offers a widget for each electronic element available in the kit. Users build circuits by dragging widgets into a canvas. After printing the circuit on a sheet of paper, electronic elements can be glued to the circuit with conductive adhesive.

PaperPulse is an IDE similar to *Circuit Stickers* which additionally enables users to demonstrate program behavior by recording actions [100]. *PaperPulse* also features simulation functionality to test the circuits before printing them. In addition to the printable file containing the circuit's layout, it outputs the code that can be uploaded

to the microcontroller.

Topiary is a tool for rapid prototyping location-enhanced mobile applications [69]. It features a storyboard-like UI where users define transitions between screens based on location events such as the arrival of the user to a specific location.

CRN Toolbox is a visual flow-based programming tool that focuses solely on the rapid prototyping of activity recognition algorithms [9]. Flow-based programming is a programming paradigm in which data is passed through different components or *boxes*. Each box executes specific computations in order to transform the data. Program behavior is defined by connecting boxes to each other causing the output of a box to be the input of a subsequent box. Some of its features (e.g. multiple threads of execution, data stream merging and splitting) target software developers with knowledge on machine learning algorithms.

Intuíno is a visual authoring tool that enables users to define behavior of physical devices using graphical representations (e.g. time-line operations and drawing splines) [126]. For instance, users can define the pattern how an LED should change its intensity by drawing a timeline on a canvas. This saves users from having to program such behavior.

*Node-RED*¹ is a visual flow-based environment for developing software for different microcontrollers, including the *Arduino Lilypad*. Its goal is to facilitate integrating devices to the Internet of Things (IoT).

*Cloud9*² is a text-based development environment in the cloud. It facilitates simultaneous collaboration between different developers.

*Espruino*³ is a hybrid text and visual block-based programming environment for the *Espruino* microcontroller.

*ModKit*⁴ is a block-based programming environment for microcontrollers.

Most of the hardware components provided in these toolkits were designed for physical devices with different requirements than those for smart textiles. In particular, their sizes, weights, dimensions and energy consumption would not fulfill the non-functional requirements for a smart textile which we described in the previous section.

3.2.3 Circuit Layout Software

Smart textile developers are still using tools designed for other domains such as circuit layout software for conventional electronics. Circuit design tools range in their complexity and target audience from tools for prototyping or professional electronic circuit design. *Fritzing*⁵, on the other hand, targets hobbyists and users with minimal experience in electronics and facilitates the creation of hardware layouts using breadboards and jumper wires. *Fritzing* contains diverse hardware components available

¹<http://nodered.org/>

²<https://c9.io/>

³<http://www.espruino.com/Web+IDE>

⁴<http://www.modkit.com/>

⁵<http://www.fritzing.org/>

in the market, including the *Arduino Lilypad* and its set of sensors and actuators. *Eagle*⁶ is a circuit layout tool for users with a high level of expertise in electronics. It enables the creation of custom hardware designs that can then be printed on a circuit board.

These tools were not designed for smart textiles. However, it is particularly important to design the circuits of a smart textile in advance because of the high amount of effort required for the construction of textile circuits manually. *TangoHapps* supports the layout of electronic devices and connections on a smart textile.

3.2.4 Development Tools for Smart Textiles

Only few solutions for smart textiles have been developed. These include frameworks, tools and simulation environments.

The *i*Catch* framework is a set of plug-and-play components and a flow-based visual programming tool that produces source code as output [89]. Deployment of source code must be done manually by copy-pasting the produced source code into the Arduino IDE⁷. *i*Catch* targets novice users with little background experience in electronics and programming such as children.

The *Co-eTex* framework consists of craft materials and a visual block-based programming environment to support the creation of smart textiles [88]. Block-based programming is a visual programming paradigm in which language constructs such as if-conditions and for-loops are represented as lego-type blocks. Blocks feature color codings and quasi-physical constraints to facilitate comprehension of the language's syntax. One of *Co-eTex*'s distinguishing features is its support for collaborative work - it enables users to work on different activities (e.g. programming, circuit layout) in parallel.

Plushbot facilitates the creation of interactive stuffed toys [55]. It includes features to layout hardware elements from the *Arduino Lilypad* family and to create textile circuit layouts. *Plushbot* also targets young users with little experience: it lacks any support for software development. *Plushbot* facilitates the design of stuffed textile toys with electronic circuits embedded in them but lacks functionality to create applications.

Eduwear is a construction Kit for wearables and tangible textile interfaces [60]. *Eduwear* consists of actuators, LEDs, buzzers and vibration motors. A block-based visual programming language facilitates application development using the parts in the *Eduwear* kit. The development environment outputs Arduino code that can be uploaded to an Arduino Lilypad microcontroller.

Harms et al. [49] developed a system for rapid prototyping smart garments for activity recognition applications based on a shirt called *SMASH*. *SMASH* can quickly attach sensors to clothing, interconnect these sensors and provides a processing architecture optimized for distributed on-body signal processing and pattern recognition.

⁶<http://www.eagle.org/>

⁷<https://www.arduino.cc/>

Martin et al. [78, 112] developed an environment that enables the simulation of parameters related to the design of a smart textile. For example, the environment enables the simulation of behavior of sensors when attached to unstable fabrics. A similar environment was developed by Mattmann et al. [80]. This environment was created to investigate how different postures and physical activity influence the elongation of clothing.

Schneegass et al. [103] developed an operating system called *Garment OS*. *Garment OS* facilitates the development of applications for smart garments with software components for data processing and user activity recognition.

Most of these tools have an educational purpose and target children and non-programmers. In contrast, *TangoHapps* is intended for the development and prototyping of smart textiles and targets a broad user audience including professional software developers and users without experience in programming or electronics. Furthermore, these tools focus on specific activities related to smart textile development, such as circuit design or application development. *TangoHapps* is the first attempt to support the circuit design, application development and testing of a smart textile in a single environment.

3.3 Analysis

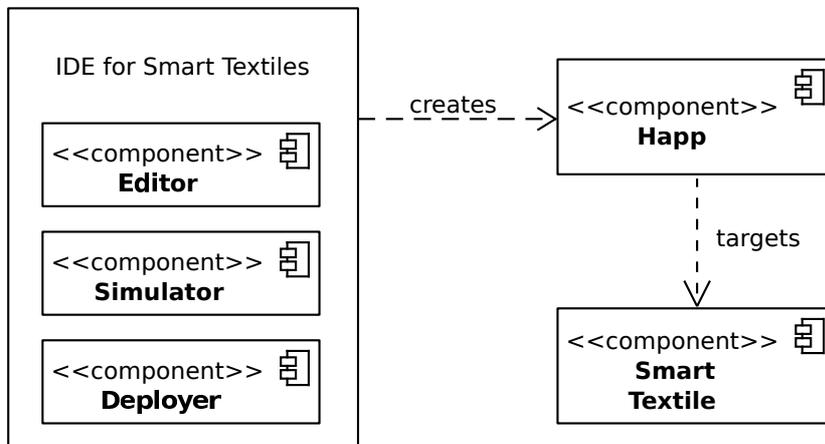


Figure 3.1: Main components of an IDE for smart textiles.

An IDE for smart textiles consists of various components that fulfill specific purposes in the creation of applications for smart textiles.

The *Editor* enables developers to create a *Happ*. A *Happ* is an application that targets a smart textile. This is in analogy to the term “App” that refers to applications that target a mobile device. The added “H” in “*Happ*” originates from the term “Haptics”. This is because we perceive *Happs* as extensions that add haptic input and output capabilities to Apps.

The *Simulator* supports developers to understand the application’s behavior at runtime before deploying it to the smart textile.

The *Deployer* is similar to a compiler - it transforms representations of applications used for development (e.g. source code) into executable applications and additionally uploads the executable into the *Smart Textile*.

This section provides an analysis object model for the *Editor*, *Simulator* and *Deployer*. Figure 3.1 shows an overview of the main components of an IDE for smart textiles.

3.3.1 Model of a Smart Textile

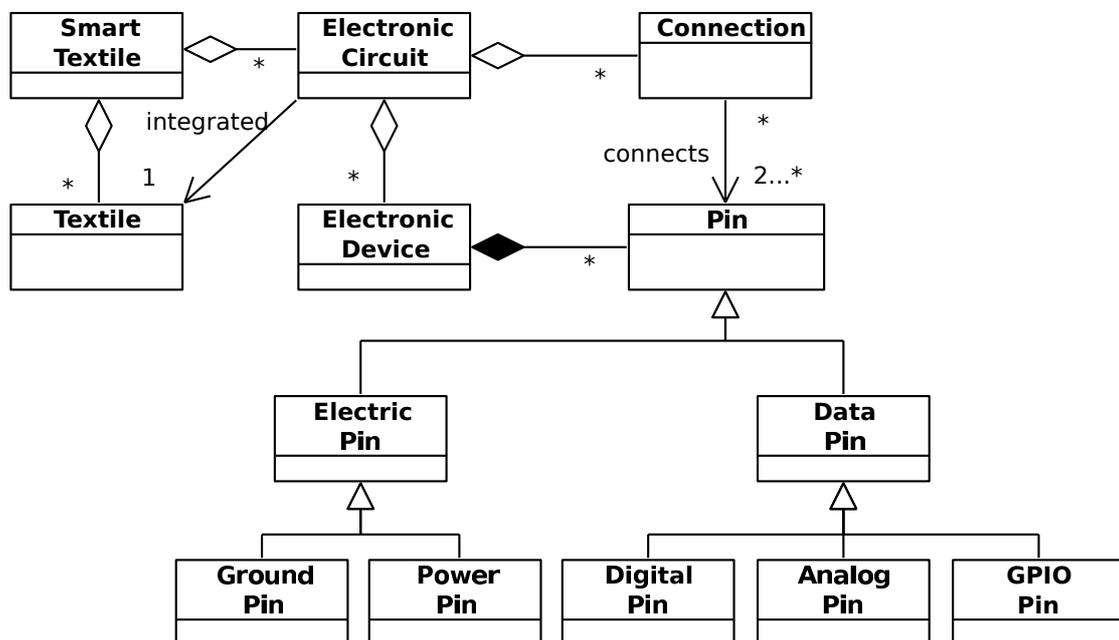


Figure 3.2: Model of a *Smart Textile*.

An UML model of a *Smart Textile* is shown in Figure 3.2. A *Smart Textile* consists of *Textile* and *Electronic Circuits*, which are integrated into the *Textile*. *Electronic Circuits* consist of *Electronic Devices* and *Connections* between them. *Electronic Devices* can be a *Sensor*, an *Output Device* or a *Microcontroller*, as described in Chapter 2. *Electronic Devices* have *Pins*, which are an interface for transmission of electric signals with other *Electronic Devices*. There are two main categories of *Pins*: *Electric Pins* and *Data Pins*. *Electric Pins* cause current to flow through *Electronic Devices* and *Data Pins* are used to transfer electric signals from *Sensors* to *Microcontrollers* and from *Microcontrollers* to *Output Devices*. On the other hand, there are two types of *Electric Pins*: *Power Pins* and *Ground Pins*. The current in a *Circuit* flows from *Power Pins*, through *Electronic Devices* into *Ground Pins*.

Data Pins can be digital or analog depending on the way how signals are transmitted along them. *Digital Pins* operate with two possible values: *high current* and *low current* and can be used for input (i.e. transmitting data from a *Sensor* to a *Microcontroller*) and for output (i.e. transmitting data from a *Microcontroller* to an

Output Device). In contrast to *Digital Pins*, *Analog Pins* operate with a range of values and can be used only for input purposes. *GPIO Pins* (General-Purpose Input/Output Pins) are *Pins* that can be configured at runtime to be used for input or output. *Connections* connect two or more *Electronic Devices* through their *Pins*.

3.3.2 Model of the Editor

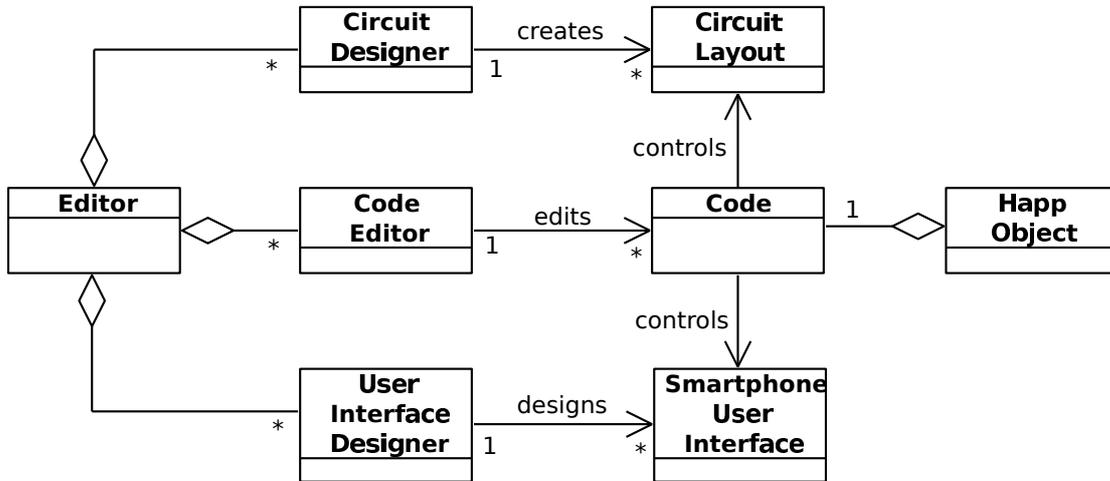
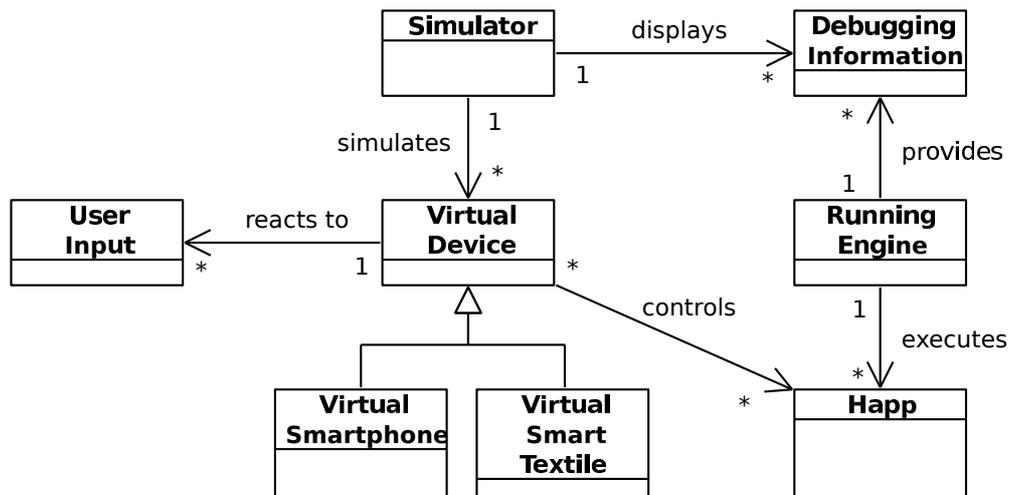


Figure 3.3: Model of the *Editor*.

The *Editor* offers developers the means to implement the behavior of an application by reusing existing software functionality (FR1), to create user interfaces for smartphones (FR3) and to design electric circuits (FR4). The *Editor* consists of three main components, the *Circuit Designer*, the *Code Editor* and the *UI Designer*. The *Circuit Designer* is used to create *Circuit Layouts* visually by arranging electronic devices on top of a model of the smart textile. The *Code Editor* is an interface for implementing, extending and reusing *Code*. Instances of the *Code* class specify the runtime behavior of the *Happ*, including how *Electronic Devices* and capabilities of a smartphone should be controlled. The *User Interface Designer* enables the creation of *Smartphone User Interfaces* by making components that wrap common smartphone capabilities (e.g. music playing) available for reuse. An UML model of the *Editor* is shown in Figure 3.3.

3.3.3 Model of the Simulator

The *Simulator* offers a way to execute a *Happ* within the development environment with the purpose of helping developers understand its runtime behavior (FR5). The *Simulator* displays *Debugging Information* provided by the *Running Engine* during

Figure 3.4: Model of the *Simulator*.

execution of a *Happ*. In order to execute a *Happ* without having to deploy it to the real hardware, the behavior of the hardware is simulated by a virtual device.

Virtual Devices fulfill two main purposes. *First*, they communicate the state of the device to the developer by means of images, animations and sounds. For example, a buzzer might shake and produce a sound to indicate that it has been turned on. *Second*, *Virtual Devices* accept *User Input* in order to simulate user interaction with the device and environmental parameters. For example, touching a simulated temperature sensor causes it to deliver a higher temperature reading. The *Virtual Device* superclass provides the means for its subclasses to control an *Happ*.

In order to improve reusability and maintainability of source code, the *Virtual Devices* offer the same interface to the *Simulator* than to the *Running Engine*. The *Simulator* executes *Happs* in *Virtual Devices* using the same interface as the *Running Engine* to execute *Happs* on the real devices. Furthermore, in order for simulations to be as close to real executions as possible, the *Happ* class does not offer special functionality or a special interface for simulation purposes. An UML model of the *Simulator* is shown in Figure 3.4.

3.3.4 Model of the Deployer

The main tasks of the *Deployer* are to transform the development representation of a *Happ* into a *Happ* and to upload the *Happ* into the target devices (FR7). A *Project* contains the assets necessary to create a *Happ*. Target *Devices* are *Smartphones* or *Smart Textiles*. The *Deployer* delegates the task to serialize and transfer applications wirelessly to the *Uploader*. An UML model of the *Deployer* is shown in Figure 3.5.

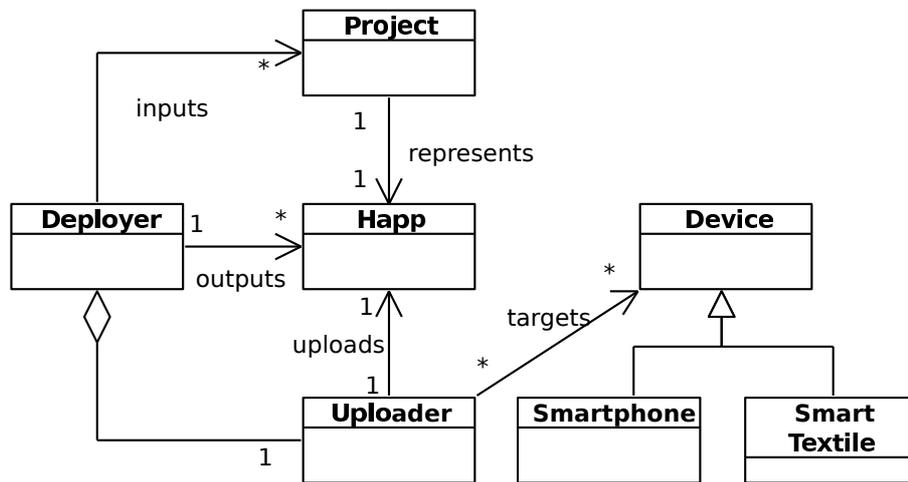


Figure 3.5: Model of the *Deployer*.

Chapter 4

TangoHapps Framework

In the previous chapters, we referred to *TangoHapps* as an IDE for smart textiles. This chapter introduces the *TangoHapps Framework (THF)*, a set of object-oriented reusable software components to facilitate the development of smart textile applications. The two key features of *THF* are *inversion of control* (i.e. the flow of control is not dictated by the caller but by the *THF* framework) and *extensibility* (i.e. the *THF* can be extended by a developer to provide the functionality specific to a smart textile application). The *TangoHapps IDE (THIDE)* is a development environment for smart textiles built with the *THF*.

The *THF* consists of a *Running Engine* and a *Happ Object Library*. The *Running Engine* provides a way to execute Happs and the *Happ Object Library* is a collection of objects that can be reused by developers using the *THF*. This chapter provides an object model of the *Running Engine* and describes the *Happ Object* taxonomy.

4.1 Running Engine

The *Running Engine* controls the method invocations between *Happ Objects* at runtime. *Runner* is the class that initiates execution of a *Happ* by instantiating the *Application* class and invoking its *execute()* method. The functionality executed by a *Happ* at runtime is distributed among *Happ Objects*.

Happ Objects contain data in the form of *Variables* and code in the form of *Methods*. *Methods* might require parameters (called *Variables*) in order to execute. A *Method* and the set of parameters it requires is called “signature”. A *Happ Object* might invoke another *Happ Object*’s *Method* if the set of parameters provided by the invoking *Method* match the receiving *Method*’s signature.

A *Happ Object* can also emit *Events*. An *Event* is an occurrence that might be handled by an application, such as when the user presses a button on his smart jacket. *Events* can deliver values, that are represented by instances of the *Variable* class. *Method* parameters and *Event* types are represented by the *Variable* class. For example, a temperature sensor might emit an event to notify about the existence of a new temperature sensor reading passing the sensor reading as a parameter. An *Event* can also invoke a *Method*. However, *Events* invoking a *Method* are also constrained

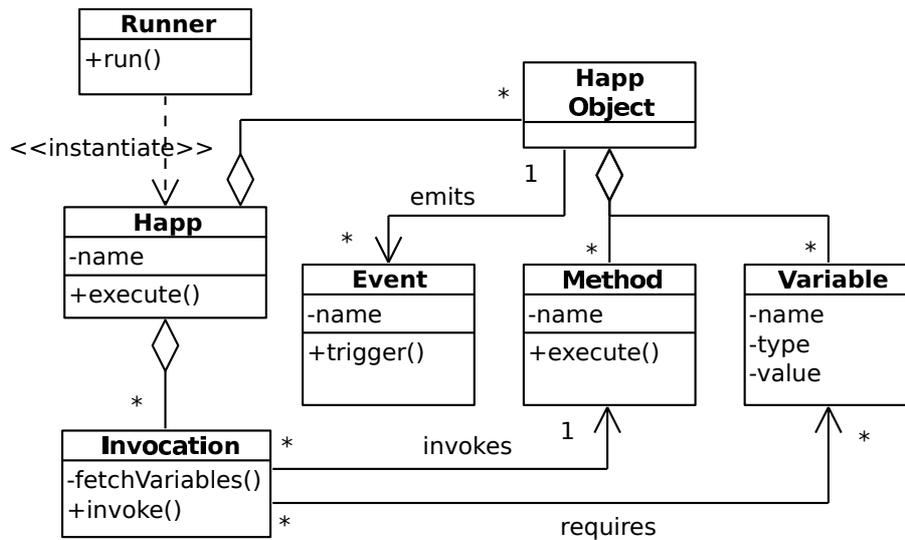


Figure 4.1: Main classes of the *Running Engine*.

by the signature of the *Method* being invoked.

The *THF* supports *Numbers*, *Booleans* and *String* types. *Objects* in *THF* are treated as every other *Type*, which enables *Events* to provide and *Methods* to expect *Objects* as parameters. An *Invocation* references the *Event* that causes a *Method* to be invoked at runtime, the *Method* and the *Variables* that will be passed as a parameter to the *Method*. Figure 4.1 displays the main classes of the *Running Engine*.

4.2 Happ Object Library

A *Happ* consists of a collection of *Happ Objects* that interact with each other. A *Happ Object* is either an *Electronic Device*, a *UI Widget* or a *Programming Construct*.

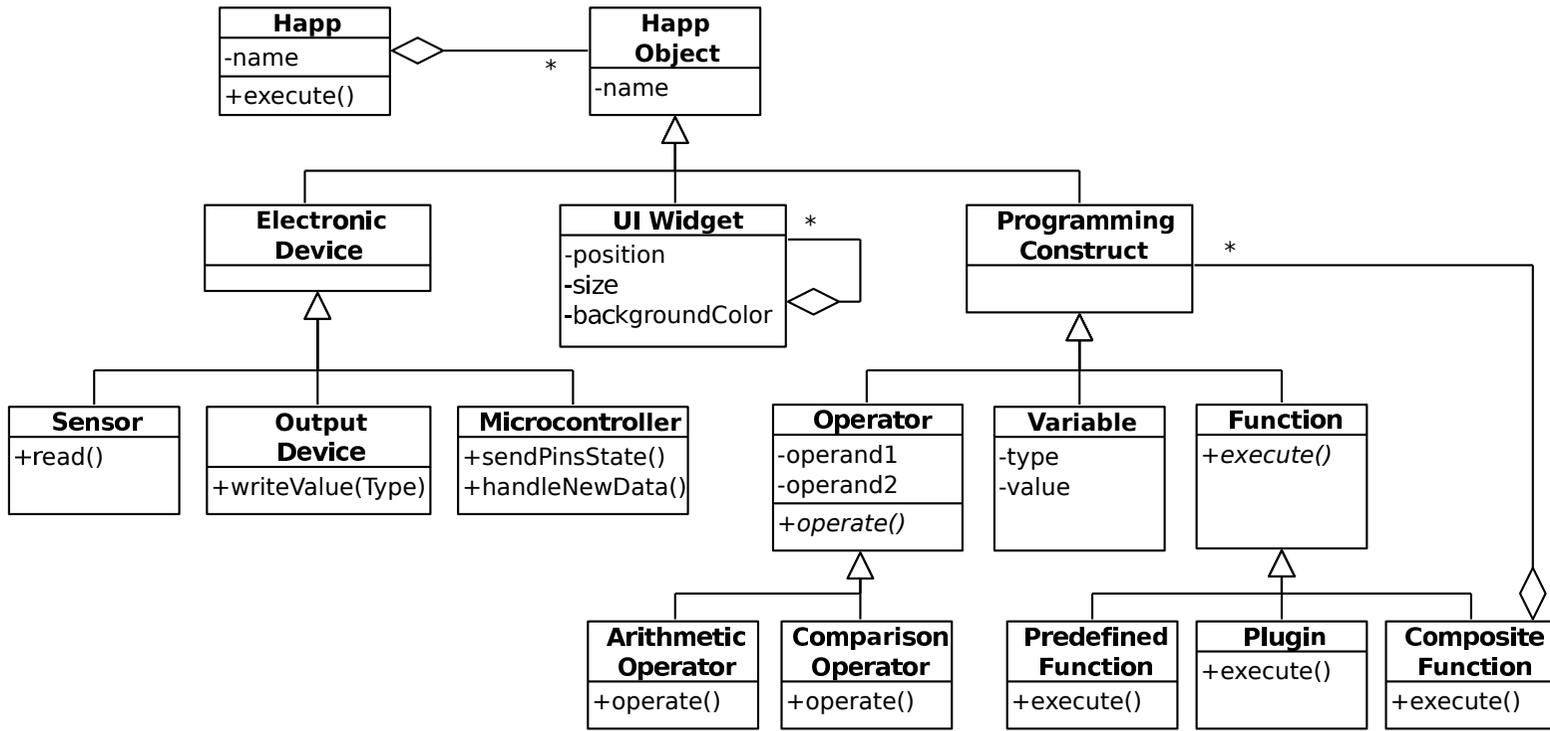


Figure 4.2: Model of the *Happ Object Library*.

An *Electronic Device* is attached to or integrated into the smart textile. It can be a *Sensor*, an *Output Device* or a *Microcontroller*. The communication between a *Happ* and an *Electronic Device* happens through an interface offered by subclasses of *Electronic Device*.

An *UI Widget* represents a widget displayed on the smartphone's screen such as buttons, labels, sliders and switches. Subclasses of *UI Widget* offer an interface for other *Happ Objects* to control the widget. For example, an *UI Widget* can handle user input captured by the widget and display information to users through the widget.

A *Programming Construct* provides software functionality that developers can reuse in their smart textile application (FR1). A *Programming Construct* can be an *Operator*, a *Variable* or a *Function*.

Figure 4.2 shows an UML model of a *Happ* and taxonomy of *Happ Objects*. In the following subsections, we list every *Happ Object* available in the *THF*. A complete description of *Happ Objects* including their *Events*, *Methods* and *Variables* is available in Chapter A in the Appendix.

Electronic Devices

The *THF* supports every sensor, output device and microcontroller from the Arduino Lilypad¹ hardware kit and additional sensors that are also suitable for integration in smart textiles, such as the MPU 6050 Inertial Measurement Unit (IMU). *Sensors* supported in the *THF* are listed next.

- *Buttons* are similar to *UI Widget Buttons*. *Buttons* can be pressed and emit events when pressed and when released.
- *Switches* are on/off switches and like *UI Widget Switches*, they emit events when their state change.
- *Light Sensors* measure the intensity of the light and emit an event when a new sensor reading is available.
- *Temperature Sensors* are similar to *Light Sensors*. *Temperature Sensors* measure temperature and emit an event when a new sensor reading is available.
- *Accelerometers* measure acceleration forces in a 3D space and emit an event when a new sensor reading is available.
- *LSM Compass* represents the LSM303C 3-axis accelerometer and 3-axis compass². The *LSM Compass* emits events when either new accelerometer or compass readings become available.
- *MPU 6050* represents the MPU 6050 3-axis accelerometer and 3-axis gyroscope³. Like the *LSM Compass*, the *MPU 6050* emits events when new accelerometer or gyroscope readings become available.

¹<https://www.sparkfun.com/products/retired/10354>

²<https://www.sparkfun.com/products/13303>

³<http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/>

- *Potentiometer* represents a generic analog textile sensor, including sensors based on the capacitive, resistive and optical principles.

Output Devices supported in the *THF* are:

- *LEDs* stand for Light Emitting Diodes. *LEDs* can be turned on and off and their intensity can be set. These *LEDs* emit light of a single color.
- *Buzzers* emit a sound at a specified frequency. Like *LEDs*, *Buzzers* can be turned on and off and their frequency can be set.
- *Three-Color LED* emit light in different colors. *Three-Color LEDs* can be turned on and off, and the intensities of their red, green and blue components can be set in order to produce different light frequencies.
- *Vibration Boards* produce a vibration at a specified frequency. Like *Buzzers*, they can be turned on and off and their vibration frequency can be set.

The *THF* supports three Arduino Lilypad microcontroller models. All of them operate with voltages that oscillate between 2.7 V and 5.5 V at 8 MHz frequency. The specific capabilities of these microcontrollers are listed below:

- *Lilypads* represent the microcontroller officially called “Lilypad Arduino Main Board”⁴. *Lilypads* contain 14 digital, 6 analog pins and 16 Kb of flash memory. *Lilypads* have no placeholder for a battery and do not include a module for wireless communication; these hardware parts need to be attached externally to the board.
- *Lilypad Simple* represents the microcontroller officially called “Lilypad Arduino Simple”⁵. *Lilypad Simple* microcontrollers have less pins and more memory than *Lilypads*: 9 digital, 4 analog pins and 32 Kb of flash memory. *Lilypad Simple* microcontrollers have a placeholder for a battery but no wireless communication module.
- *BLE-Lilypad* represents a microcontroller developed by the Universität der Künste in Berlin based on the Lilypad Arduino Main Board. The *BLE-Lilypad* has the same features as the *Lilypad* and additionally features a placeholder for battery and an integrated Bluetooth Low Energy module.

UI Widgets

UI Widgets (i.e. reusable elements of a graphical user interface) are used to create the interface shown on the mobile device. *UI Widgets* range from simple elements such as buttons, labels and sliders to more complex elements to access specific capabilities of a smartphone such as its contacts and music lists. *UI Widgets* have *Variables* that enable developers to configure their visual appearance, such as their position and dimensions within the mobile device’s screen and their background color. *UI*

⁴<https://www.arduino.cc/en/Main/ArduinoBoardLilyPad>

⁵<https://www.arduino.cc/en/Main/ArduinoBoardLilyPadSimple>

CHAPTER 4. TANGOHAPPS FRAMEWORK

Widgets can be simple (e.g. a button) or contain other *UI Widgets* (e.g. *Music Player*, which contains *Buttons*, *Labels*, *Switches* and *Sliders*). The subclasses of *UI Widgets* supported in *TangoHapps* are listed next.

- *Buttons* are equivalent to push buttons on mechanical devices. *Buttons* emit events when they start to be pressed and when they are released.
- *Labels* are used to display text. Their text can be set statically or dynamically.
- *Switches* are equivalent to an on/off switch (e.g. a light switch). *Switches* have two possible states *on* and *off* and emit events when their state change.
- *Sliders* have a handle that allows users to select a value through a range of allowed values. *Sliders* emit events when the handle is moved.
- *Touchpads* represent an area on the smartphone where multi-touch gestures are recognized. Multi-touch gestures supported by the *Touchpad* are: *tap*, *double tap*, *long tap*, *pinch* and *pan*. *Touchpads* emit events when a multi-touch gesture is recognized.
- *Image Views* are used to display an image, which is set statically.
- *Music Players* are composite widgets that contain *Buttons*, *Labels*, *Image Views* and *Sliders*. *Music Players* contain functionality to iterate and play music stored on the user's mobile device. Music playlists can be controlled by users through the *Music Player*'s user interface or by other *Happ Objects* through its methods.
- *Contact Books* are composite widgets similar to *Music Players* that contain *Buttons*, *Labels* and *Image Views*. *Contact Books* contain functionality to iterate through the user's contacts, and to make phone calls if the mobile device supports it. In a similar way to *Music Players*, *Contact Books* offer an interface to users as well as to other *Happ Objects* to access its functionality.
- *Monitors* are composite widgets used to display series of values in a plot. This is useful for applications where users need to monitor sensor values over time.

Programming Constructs

Programming Constructs are the main means by which developers reuse, extend and implement behavior in *TangoHapps*. A *Programming Construct* is either an *Operator* (e.g. an arithmetic operator), a *Variable* or a *Function* (e.g. an accelerometer-based human posture classifier).

An *Operator* can be an *Arithmetic*, a *Comparison* or a *Logical Operator*. An *Arithmetic Operator* takes two numeric inputs and produces a numeric output. Supported *Arithmetic Operators* are *Addition*, *Subtraction*, *Multiplication*, *Division* and *Modulo*. A *Comparison Operator* compares two numeric values and provides the result of the comparison as a Boolean *Variable*. Supported *Comparison Operators* are: *Bigger*, *BiggerEqual*, *Smaller*, *SmallerEqual*, *Equal* and *NotEqual*. A *Logical Operator* takes

two Boolean inputs and provides the result of a logical operation as output. A *Logical Operator* is either an *AndOperator* (an AND-conjunction) or an *OrOperator* (an OR-disjunction).

Variables store values and emit events when the stored value has changed so that other objects can react to the change. Supported *Variables* in *TangoHapps* are: *Booleans*, *Numbers*, *Strings* and *Objects*. *Booleans* store Boolean values, *Numbers* store real numbers including integer and floating point numbers and *Strings* store chains of characters.

Functions are executable code procedures, comparable to functions in text-based programming. An example of a *Function* is the *Mean Extractor*. The *Mean Extractor* calculates the mean of a set of values it receives as input. *Functions* can be *Predefined Functions*, *Plugins* or *Composite Functions*. *Predefined Functions* are functions already available in *TangoHapps*. In contrast, *Plugins* are a container for code not originally available in the *THF* but provided by developers. We explain the concept of *Plugins* in detail in Section 6.5. *Predefined Functions* for signal processing are:

- *Windows* take streams of values, buffer them and deliver chunks of values. *Windows* are useful for signal processing algorithms to process chunks of a signal. The amount of samples contained in a chunk and the amount of overlapping samples between two consecutive chunks can be configured by developers.
- *Low-Pass Filters* smooth out a signal by attenuating its high frequencies.
- *High-Pass Filters* are the opposite of *Low-Pass Filters*. *High-Pass Filters* attenuate the low frequencies of a signal and pass its high frequencies.
- *Mean Extractors* compute the mean of a set of values.
- *Deviation Extractors* compute the deviation of a set of values.
- *Peak Detectors* compute the highest peak of a set of values and deliver its value and index within the set of values.
- *Activity Classifiers* determine user physical activity (walking, running, climbing, quiet) based on accelerometer input.
- *Posture Classifiers* determine the user posture (standing, lying down on belly, lying down on the back) based on accelerometer and gyroscope input.

Other *Predefined Functions* are:

- *Timers* generate events after a specified amount of time.
- *Sounds* reproduce audio files over the user's mobile device. *Sounds* differ from *Music Players* in that they do not access the device's music lists, but rather play a sound predefined at development time. *Sounds* are useful for user interfaces.
- *Recorders* offer functionality to store a set of values on the mobile device's hard drive and to reproduce it later.

CHAPTER 4. TANGOHAPPS FRAMEWORK

- *Mappers* scale numeric values within a specified range. They are useful for adapting either sensor input to a specific range or values to the range required by an output device.

Composite Functions group *Programming Constructs* and *Invocations* between them in order to construct complex *Programming Constructs* by aggregating simpler ones. *Composite Functions* were designed as a Composite design pattern [36]. *Composite Functions* enable developers to dynamically define their *Methods* and *Events*. *Methods* and *Events* of a *Composite Function* do not contain any specific behavior, but simply redirect to a *Method* or *Event* of a *Programming Construct* contained within the *Composite Function*.

Chapter 5

System Design

Chapter 3 presented the requirements and analysis object models that serve as a basis for the design of the *THIDE*, which we describe in this chapter. Section 5.1 describes the trade-offs we encountered and decisions we made during the design of the *THIDE*. In particular, we describe how the different alternatives we analyzed would have affected the *THIDE* in terms of the non-functional requirements. Section 5.2 describes the subsystems and architecture of the *THIDE* and section 5.3 describes its hardware/software mapping.

5.1 Design Decisions

When developing a software, developers have to make trade-offs among many design goals that might contradict each other [14]. In this section, we describe the design decisions we made in order to realize the non-functional requirements described in Section 3.1.2.

Hybrid text and visual programming

Visual programming languages have been shown to lower the entrance barrier to users with little programming knowledge [72, 51, 7, 88, 89]. In the *THIDE*, software functionality components, electronic devices and elements composing the user interface of a smartphone are represented visually. However, a visual programming language does not scale well for Happs with many components. It is also inconvenient for experienced software developers. To address NFR1 (*Low entrance barrier*) and NFR2 (*High ceiling*), we decided to design the *THIDE* based on a hybrid visual and a textual programming language.

Prototype-based programming

The *THIDE* should offer software functionality for reuse (FR1) and this functionality should be usable by beginners (NFR1). Prototype-based programming makes it possible to reuse software functionality by cloning and modifying existing object prototypes. For example, given a soccer ball, one can create a tennis ball by changing its color to green and reducing its size to 6.5 cm. This is in contrast to the class-based programming paradigm, where a class has to be defined before an object can be instantiated. Furthermore, the prototype-based programming is considered to be simpler than class-based programming [117, 123]. In order to facilitate reuse of the software functionality provided by the *THF*, we decided that the *THIDE* is designed based on a prototype-based programming approach. The *THIDE* offers a palette of object prototypes that can be cloned and modified to fulfill the requirements of a specific smart textile application.

Flow-based programming

The choice of the visual programming language was mainly driven by NFR1 (*Low entrance barrier*) and NFR2 (*High ceiling*). The alternatives we considered were block-based programming, programming by demonstration and flow-based programming.

Block-based programming represents programming constructs such as if-conditions and for-loops with different colors and shapes. This makes the syntax of the language more explicit. However, developers using a block-based programming language face similar difficulties as with a text-based programming because programs have to be created based on the same programming constructs.

In programming by demonstration, the only way developers program the system is by demonstrating certain behaviors. Therefore, this paradigm is limited to those behaviors that can be demonstrated and achieves a relatively low ceiling.

In flow-based programming, software functions are represented as boxes. In order to specify method calls, boxes are connected to each other. We decided to design the *THIDE* based on a flow-based programming paradigm for two main reasons. First, the flow-based programming paradigm has been shown to lower the entrance barrier to development in several development environments for physical devices [39, 77, 30, 29, 89, 9] and development tools for other domains, such as *Node-RED*, *Max/MSP* and *PureData*. Second, this paradigm can be used together with a text-based programming language in order to increase its ceiling. Components developed over a text-based interface can be represented as boxes and used within the flow-based programming language.

Tablet device for design

We decided to implement the visual programming interface of the *THIDE* on a tablet device based on the following reasons, addressing *Low entrance barrier* (NRF1) requirement:

1. Tablet devices allow for direct manipulation of objects on the touchscreen. Representing software functionality components visually and making them tangible helps make smart textile programming easier to novice.
2. The multi-touch capabilities of tablet devices enable their usage by multiple users simultaneously. This might be beneficial for developers with less experience in programming and electronics to learn by collaborating on the same project.
3. An IDE for smart textiles must make it possible to use the capabilities present in a smartphone (FR3) and must enable the simulation of the smart textile and smartphone (FR4) . Because tablet devices have similar capabilities to smartphones, they allow for a realistic design and simulation of the smartphone application.

Delegation of computations to a smartphone

As we discussed in Section 3.1.2, complex computations performed on the smart textile lead to heavier and bulkier hardware. In order to allow lightweight constructions of smart textiles, we decided to delegate most of the smart textile's runtime computations to the smartphone. We considered two main categories of devices: mobile devices (smartphones, tablet devices) and non-mobile devices (desktop or laptop computers). We decided for a mobile device based on two main reasons:

1. Many mobile devices enable communication over Bluetooth Low Energy, a protocol optimized for low energy consumption (which supports NFR3). The delegation of computations to a non-mobile device would require the transmission of data over longer-range communication technologies, which in general lead to higher power consumption rates.
2. The smart textile must be functional even if it is not reachable over a wireless connection (NFR5). Non-mobile devices might not be reachable from every location (e.g. in the metro). The delegation of computations to a device that cannot be reached would cause the smart textile to be non-functional. Instead, hosting the computations on a mobile device that is carried by the user increases the chances of the device being reachable.

As discussed in Section 3.1, an IDE for smart textiles must support smartphones (FR3). Since the smart textile must communicate with a smartphone, we decided that the same smartphone also performs the complex computations. The choice to

delegate computations to a different device would add a point of failure and could limit performance.

Interpreted language

Interpreted programs can be easier to debug at runtime because interpreters use the same representation of the software as the one used by developers. Compiled languages, instead, transform programs written by the developer into machine code making it harder to relate execution errors to faults in the program. An interpreter can display information at runtime, which is aligned with the non-functional requirement of *Low entrance barrier*. For this reason, Happs developed with the *THIDE* are interpreted.

5.2 Architecture

The software architecture of the *THIDE* is based on a combination of the repository and layered architectural styles [14, 16]. The software components in the *THIDE* and *THF* are organized into layers to create, modify, simulate, deploy and execute a Happ, which is the repository of the architecture. An object model of these software components is provided in Chapters 4 and 6. The repository architectural style decouples the software components in the *THIDE* making them independent from each other.

The *Happ* subsystem is also organized in layers. The top layer of the Happ is the user interface, which consists of the *UI Widgets* displayed on the smartphone and *Electronic Devices* attached to the smart textile. The bottom layer of the Happ consists of a set of *Programming Constructs* that reuse software functionality provided by the *THF*. *Programming Constructs* are executed by *THF's Running Engine*. The Happ can also contain software functionality that was not originally available in the *THF*. This software functionality is contained within *Plugins*, which conform the middle layer of the Happ. *Plugins* are interpreted by the *Plugin Interpreter*.

The *Development* layer of the *THIDE* contains the functionality to develop, debug and deploy applications for smart textiles. For this purpose, the *Development* layer has three main components: the *Editor*, the *Simulator* and the *Deployer*. The *Editor* is the equivalent to a source code editor in a traditional IDE and is the main interface developers use in order to create Happs. The *Simulator* contains the main functionality to simulate and debug the Happ, without having it to deploy to the smart textile. The *Deployer* transforms the Happ's source code into an executable and uploads the executable to the smart textile.

The *Execution* layer contains the main functionality to execute applications developed in the *THIDE*. The Software components in the *Execution Layer* are the *Plugin Interpreter* and the *Firmata* components. The *Plugin Interpreter's* task is to interpret code *Plugins*. The *Firmata Library* executes input and output operations

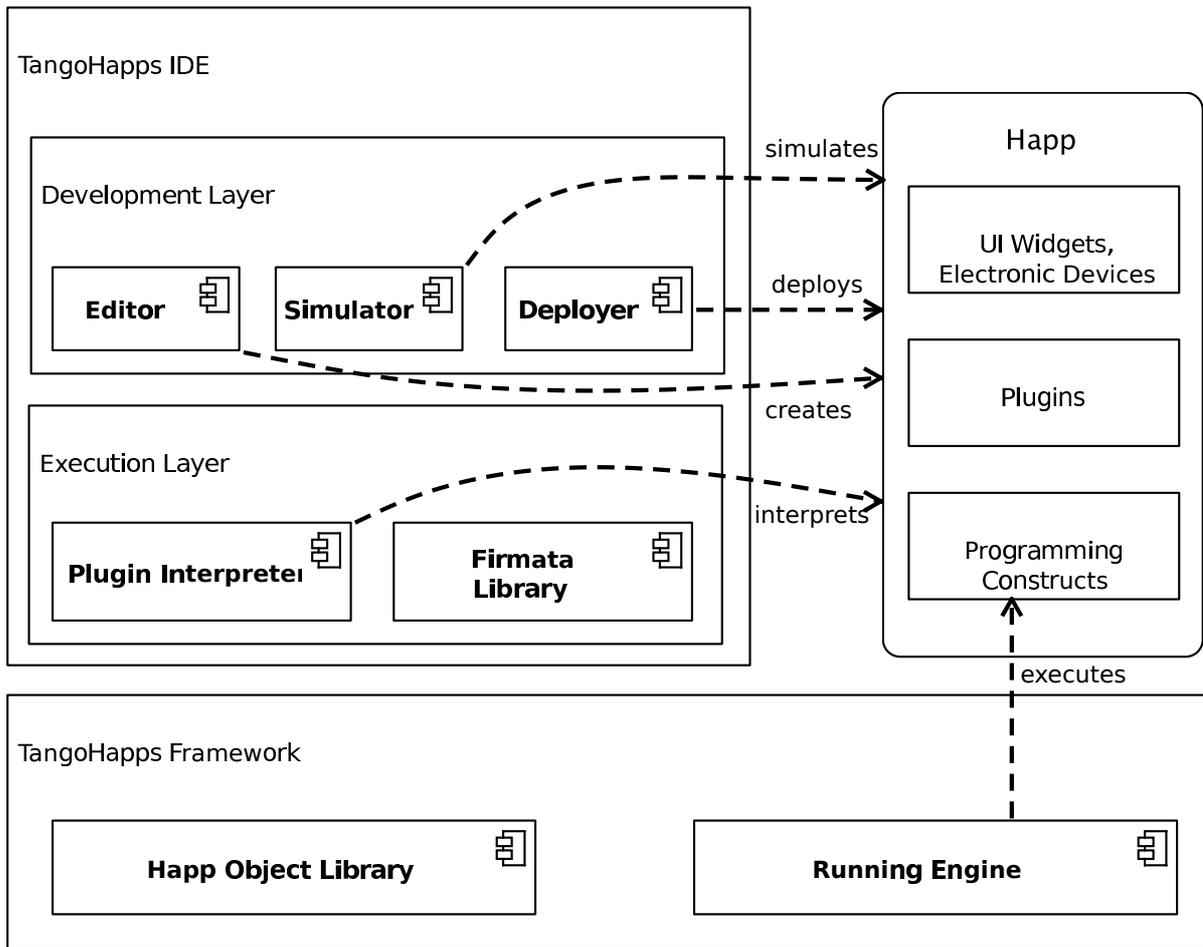


Figure 5.1: Software architecture of the *THIDE* based on the layered and repository architectural styles. The Happ subsystem is the repository. Software components in the *TangoHapps IDE* and *TangoHapps Framework* access the Happ. Accesses to the Happ are represented with dashed lines.

on the smart textile's hardware and offers a service for other components to control the hardware remotely.

The software architecture of *THIDE* is shown in Figure 5.1. The following section in this chapter describes the mapping of software components to hardware devices.

5.3 Hardware/Software Mapping

The *THIDE* consists of two components: *Interactex* and *TextIT*. *Interactex* is a visual programming environment to develop, debug and execute Happs. *Interactex* includes three tools: *Interactex Designer*, *Interactex Client* and *Interactex Firmware* [44]. *Interactex Designer* contains the functionality to develop and debug Happs. Happs created in *Interactex Designer* are deployed into *Interactex Client*, where they are executed. *Interactex Client* controls the hardware on the smart textile through APIs offered by *Interactex Firmware*. *Interactex Firmware* runs on the microcontroller attached to the smart textile. *TextIT* is a hybrid text and visual programming environment used to extend the functionality of *Interactex*. Software components are developed using *TextIT* and imported into *Interactex Designer* where they are linked to a specific hardware setup and reused in the context of an application.

The *THIDE* runs on four different hardware devices. *TextIT* runs on a web browser. Therefore, it executes on any computing device able to host a browser. *Interactex Designer* executes on a multi-touch tablet device. *Interactex Client* executes on a smartphone. The current implementations of *Interactex Designer* and *Interactex Client* are made for iOS. *Interactex Designer* can be installed on an iPad device and *Interactex Client* on an iPhone. *Interactex Firmware* executes on a microcontroller. The current implementation of *Interactex Firmware* supports the Arduino microcontroller family. Figure 5.2 displays how the the software components introduced in previous section are distributed into the hardware devices.

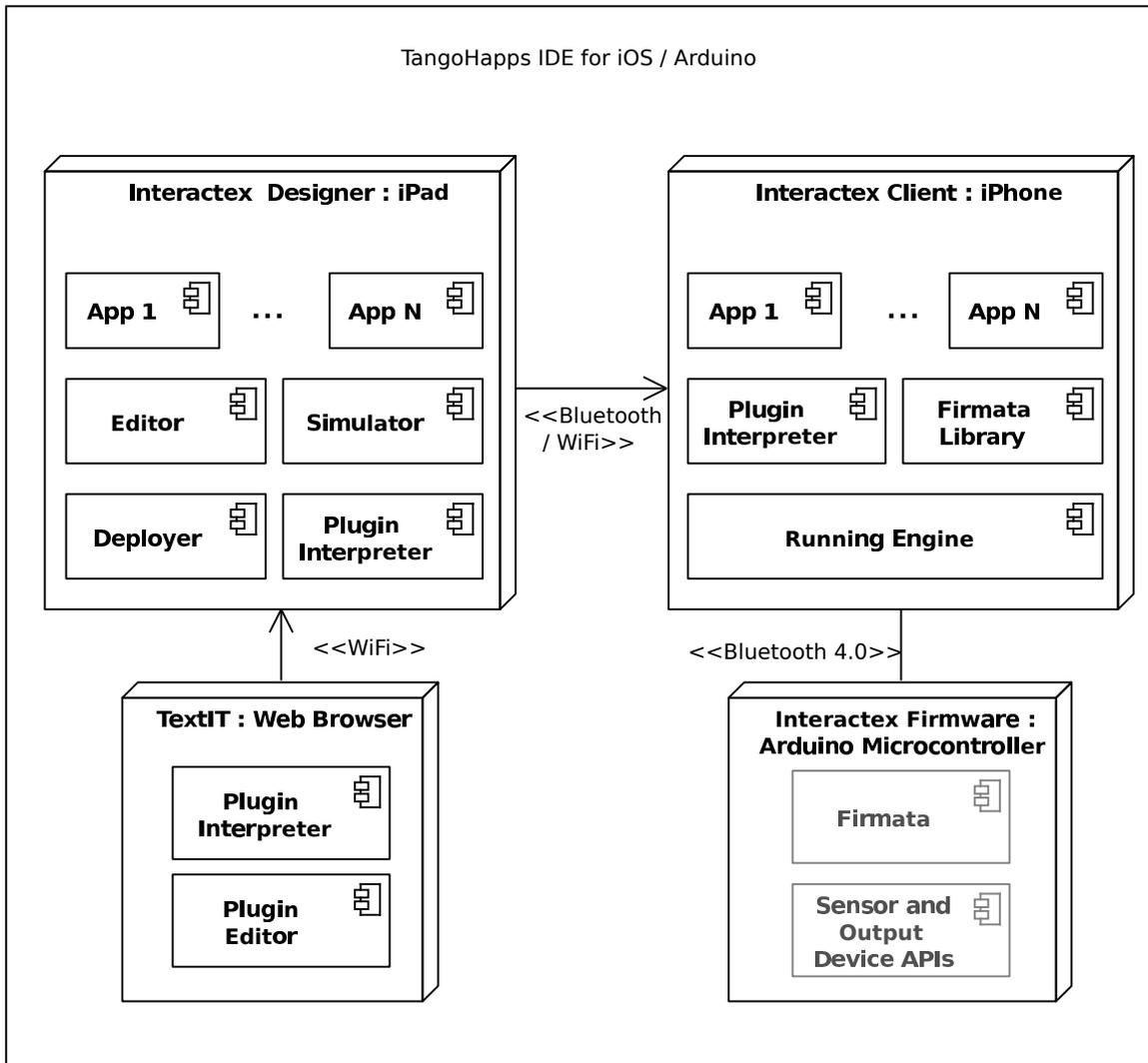


Figure 5.2: Deployment of software components in *THIDE*.

CHAPTER 5. SYSTEM DESIGN

Chapter 6

Object Design

Previous chapter introduced the software components in the *THIDE*. This chapter describes the object model of these components.

6.1 Editor

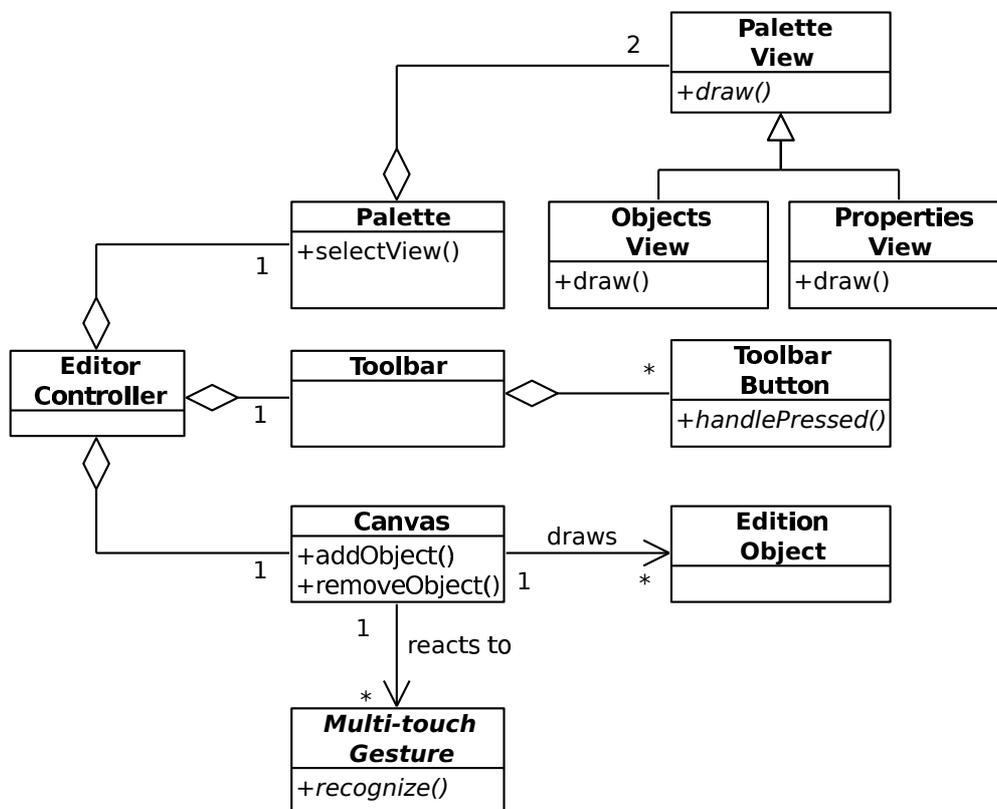


Figure 6.1: Overview of the *Editor*.

The *Editor* offers an interface for developers to design circuits and create applications. The *Editor* consists of three main components: the *Palette*, the *Toolbar* and the *Canvas*. The *Palette* fulfills two tasks. First, it displays a list of *Happ Objects* that developers drag into the *Canvas* in order to reuse their functionality. Second, it offers an interface to developers for modifying the *Variables* of the object that is currently selected in the *Canvas*. The *Toolbar* contains *Toolbar Buttons* that are used to manipulate objects in the *Canvas* and modify the *Editor's* state. The *Canvas* displays *Edition Objects* and enables developers to manipulate them (e.g. move them, delete them, define *Invocations* between them). Developers manipulate *Edition Objects* by means of *Multi-touch Gestures*. The *Multi-touch Gesture* is an abstract class that defines an interface for recognizing multi-touch gestures. Subclasses of *Multi-touch Gesture* implement the behavior for recognizing a gesture based on sequences of user touches. Currently supported multi-touch gestures are taps, double tap, pans, pinches and rotation gestures. An UML model of the *Editor's* main classes is shown in Figure 6.1. We continue this section with a more detailed description of the components in the *Editor* and finally describe how they can be used to design a circuit and develop an application.

6.1.1 Palette

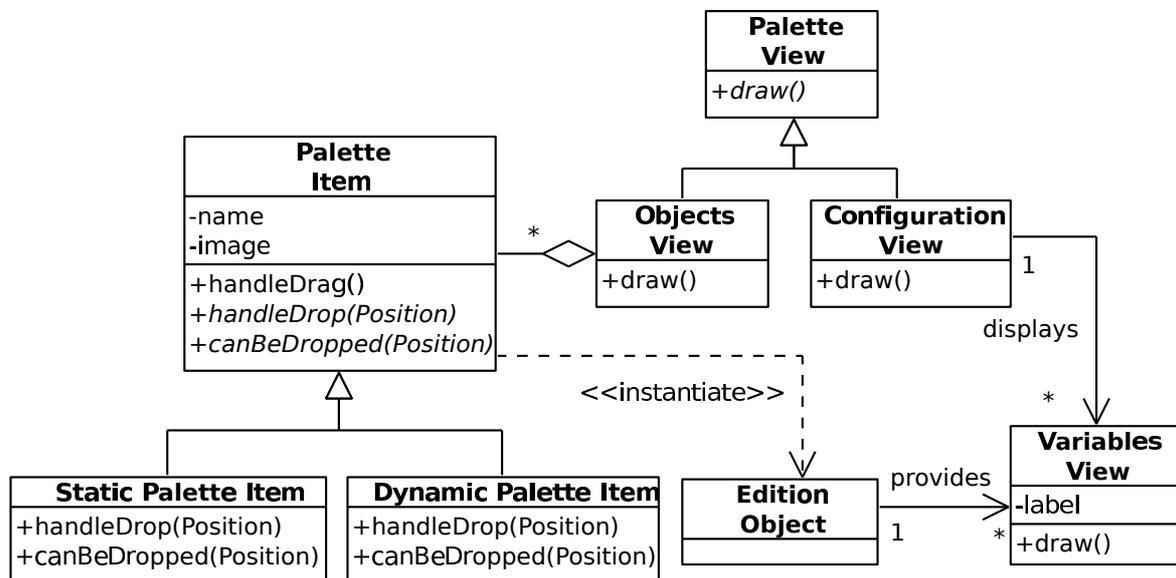


Figure 6.2: Main classes involved in the *Palette*.

The *Palette* enables developers to add objects to a *Project* and edit them. The *Palette* contains two views, the *Objects View* and the *Configuration View*. The *Objects View* displays a list of every object available in the *THIDE* for reuse and the *Configuration View* enables developers to modify an object's configuration.

Objects displayed in the *Objects View* are called *Palette Items*. A *Palette Item* is a visual representation of an *Application Object* that contains an image and a name.

Palette Items can be dragged into the *Canvas*. *Palette Items* can be dropped at specific positions within the canvas. For example, *UI Widgets* can only be dropped on top of the smartphone and *Hardware Devices* can only be dropped on top of a textile. For this reason, each *Palette Item* subclass has a different implementation of the *canBeDropped()* method.

When dropped, each *Palette Item* instantiates a different *Edition Object*. *Palette Items* that instantiate *Edition Objects* available in the *THIDE* by default are called *Static Palette Items*. We have described the concept of *Composite Functions* in Section 4.1. A *Composite Function* is a *Programming Construct* that has been created by aggregation of simpler *Programming Constructs*. A *Composite Function* can be added to the *Palette* for later reuse in the same or in a different *Project*.

Palette Items that were originally available in the *THIDE* are called *Static Palette Items*. *Static Palette Items* are programmed to instantiate a specific *Editable Object*. In contrast, *Dynamic Palette Items* instantiate a *Composite Function* that has been added to the *Palette* at runtime.

It is unlikely that the default configuration of an *Application Object* fulfills the requirements of a specific use case. *Application Objects* can expose their *Variables* in order for developers to modify the default behavior of an *Application Object*. *Variables* exposed by an *Application Object* are modified through the *Variables View*. Each *Edition Object* generates a different instance of a *Variables View* containing a user interface with controls to modify an *Application Object*'s exposed *Variables*. A complete list of every *Application Object*'s *Variables* is available in Chapter A in the Appendix. An UML model of the main classes of the *Palette* and the relationships between them is provided in Figure 6.2.

6.1.2 Toolbar

Toolbar Buttons available in the *Toolbar* are the *Connection Button*, the *Copy Button*, the *Remove Button* and the *Hardware Button*. Each *Toolbar Button* sets the *Editor* into a different state. The *Editor* reacts differently to user input depending on its current state. We have designed the *Editor*'s response to user input as a State pattern [36]. Every subclass of *Tool State* conforms to the *Tool State* interface by implementing the *handleGesture()* method. Each *Tool State* handles a *Multi-touch Gesture* differently. The *Editor*'s state is modified by setting its reference to the current instance of *Tool State*. Figure 6.3 displays the relationship between the *Toolbar* and the *Editor* in a class diagram.

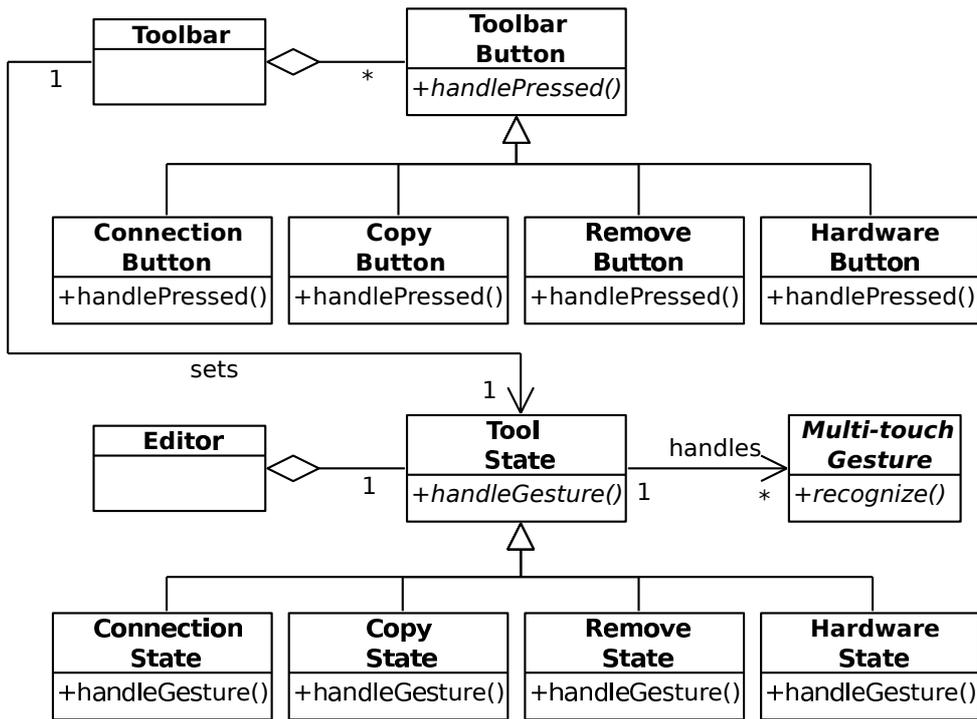


Figure 6.3: Main classes involved in the *Toolbar*.

6.1.3 Canvas

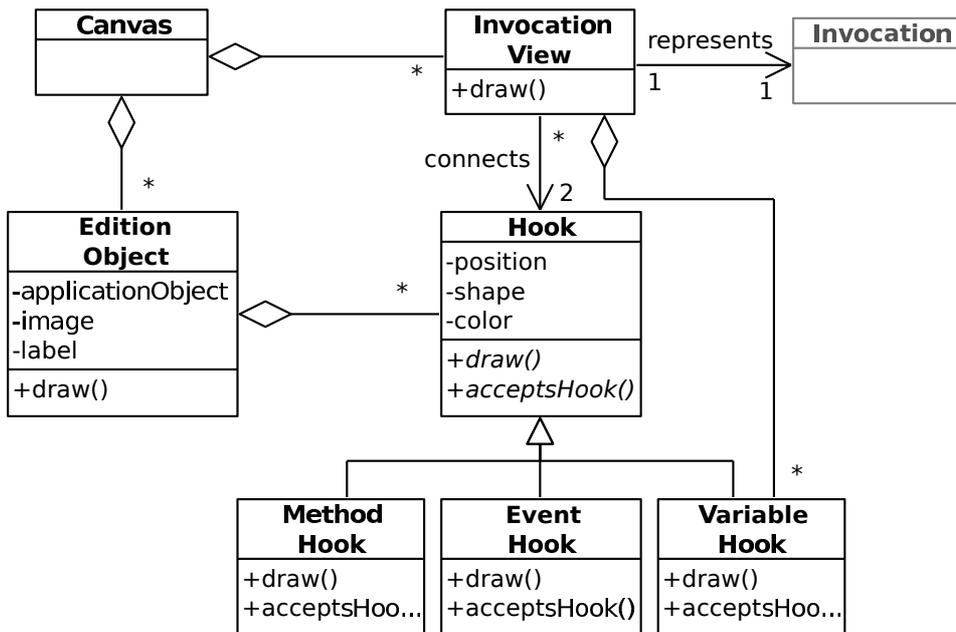


Figure 6.4: Main classes involved in the *Canvas*.

The *Canvas* is where the visual programming in *Interactex Designer* takes place. Applications in *Interactex Designer* are created by adding *Edition Objects* into the *Canvas* and by defining *Invocations* between them. *Edition Objects* are representations of *Application Objects* that expose the *Application Objects' Events, Methods* and *Variables* in order for developers to define how *Events* should be handled, when *Methods* should be invoked and how *Variables* should be set at runtime. *Edition Objects* are shown as an image with a label on screen. *Edition Objects* have the so called *Hooks*. *Hooks* are visual representations of *Methods, Events* and *Variables* that developers use to define method invocations. Connecting an object's *Event Hook* to another object's *Method Hook* causes an *Invocation* to be created that will invoke the specified *Method* when an *Event* triggers at runtime.

Invocation Views are a visual representation of an *Invocation*. *Invocation Views* draw lines between objects in the *Canvas* and display *Variable Hooks*. *Variable Hooks* fulfill two purposes. First, they display the types of parameters being passed in an *Invocation*. Second, *Variable Hooks* enable developers to pass an *Application Object's Variables* as parameters into any *Invocation*. For example, an *Invocation* between the *Button's buttonPressed()* event and the *LED's setIntensity(Number)* would not valid because the event does not provide a parameter of type *Number*, as expected by the *setIntensity(Number)* method. In this case, the *Invocation View* will still be shown with an icon indicating that the parameters provided by the event do not comply with the parameters expected by the method. The different icons used by *Invocation Views* are shown in Section 7.1.3. In this case, the *Variable Hook* of a third object with a *Variable* of *Number* type might be dragged into the *Invocation View*. At runtime, when the *buttonPressed()* event triggers, the *Running Engine* will fetch the third object's *Variable* and pass it in the *setIntensity(Number)* method call. Figure 6.4 displays a model of the main classes involved in the *Canvas*.

6.1.4 Circuit Layout

In order to create circuits, developers lay out *Electronic Devices* on top of the *Smart Textile* and draw *Connections* between them. *Connections* connect two or more *Pins*. An *Electronic Device's Pin*, however, can have more than one *Connection*. *Connections* can be straight lines between two *Pins* or a collection of line segments. In order to split a *Connection* into two or more line segments, developers add *Routing Nodes* to a *Connection*. *Routing Nodes* are fixed positions defining the edges of the segments composing the *Connection*.

Not every *Pin* may be connected to every other *Pin*. A set of *Constraints* limit the amount of *Pins* a *Pin* can be connected to. The *Allowed Pin Constraint* enforces a *Pin* to be connected to a specific type of *Pins*. The *Allowed Device Constraint* enforces a *Pin* to be connected to *Pins* of a specific type of *Electronic Device*. The *Requires Resistor Constraint* enforces that a *Pin* is connected to another *Pin* through a *Resistor*. A *Pin* can have multiple *Constraints* at the same time. An UML model of the main classes involved in the lay out of electronic circuits and the relationships between them is provided in Figure 6.5.

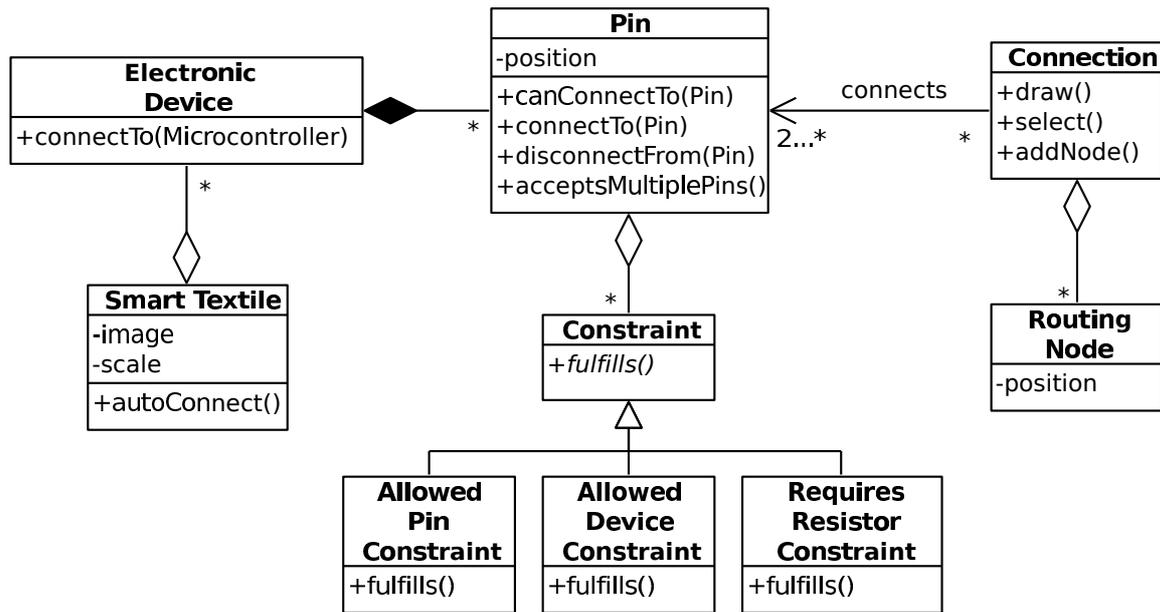


Figure 6.5: Main classes involved in the *Editor*'s functionality to create circuit layouts.

Next, we describe the concept of pin constraints with an example of a *Button* connected to a digital pin of the Arduino *Lilypad*. The *Button* has two *Pins*: a *Power Pin* and a *Digital Pin*. The *Power Pin* has an *Allowed Pin Constraint* that enforces it to be connected to other *Power Pins*. The *Digital Pin* has three constraints: a *Constraint* to enforce the usage of a *Resistor*, a *Constraint* to enforce its connection to a *Digital Pin*, and a *Constraint* to enforce its connection to a *Microcontroller*. All three *Constraints* are satisfied because the *Digital Pin* is connected to the *Lilypad*'s *Digital Pin 9* and to the *Lilypad*'s *Ground Pin* through a *Resistor*. Figure 6.6 shows the described circuit layout created with *Interactex Designer*.

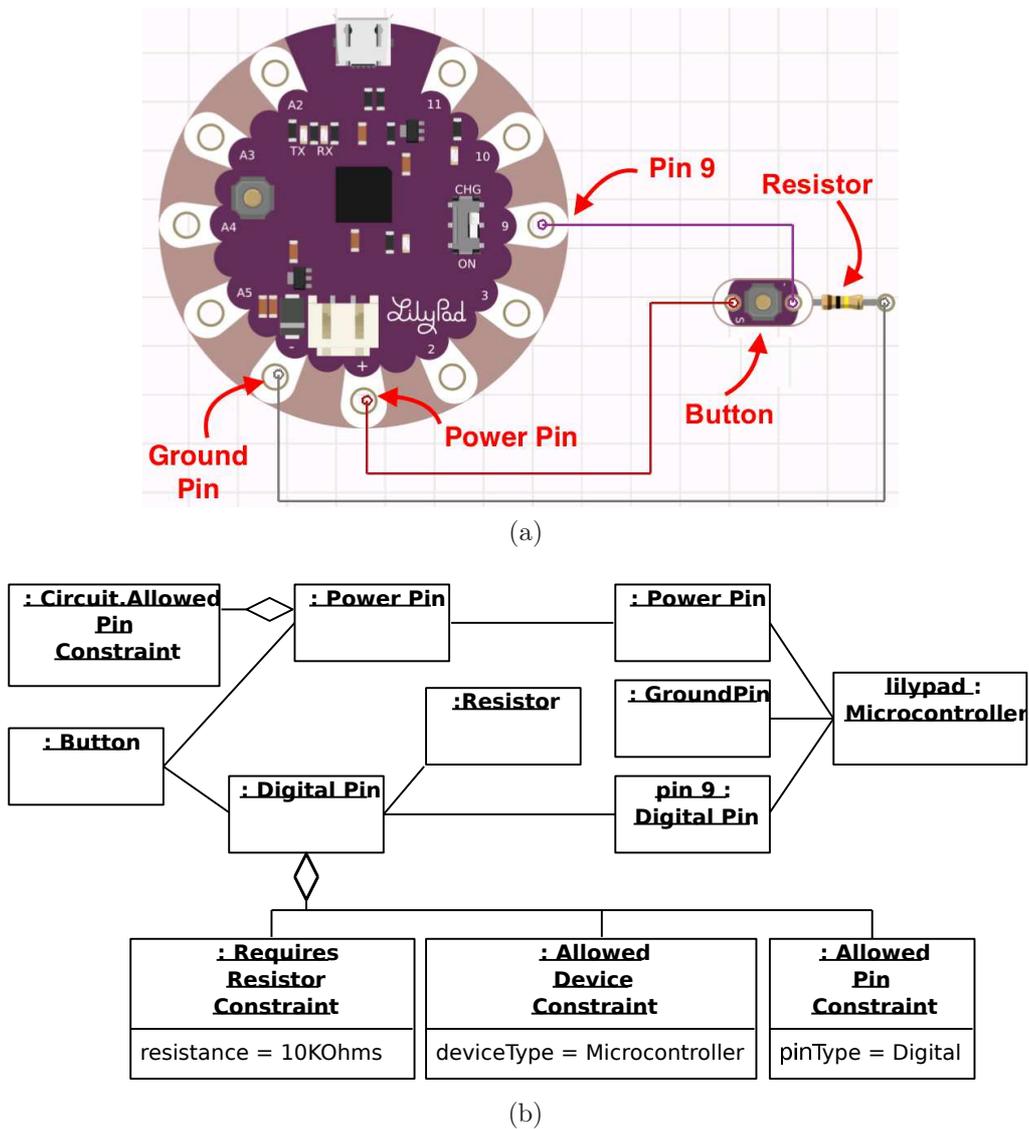


Figure 6.6: Layout (a) and object diagram (b) of an electric circuit that consists of a *Button* connected to an *Arduino LilyPad*.

6.2 Simulator

The *Simulator* is responsible for executing an *Application* in *Interactex Designer*. This enables developers to test *Applications* without having to deploy them to the smart textile. The *Simulator Controller* class is the Facade of the *Simulator*. The *Simulator Controller* offers an interface to other components for starting and stopping a simulation.

Applications in the *THIDE* are an aggregation of *Application Objects*. Therefore, in order to test an *Application*, developers test the different states of each *Application Object*. The functionality to simulate applications is distributed along the subclasses of the *Simulation Object* class. Each subclass of *Simulation Object* is a visual representation of an *Application Object*. *Simulation Objects* display runtime information about the *Application Object* they represent and react to *User Gestures* by modifying an *Application Objects*'s state. Section A.4 in the Appendix lists the behavior of each *Simulation Object* subclass.

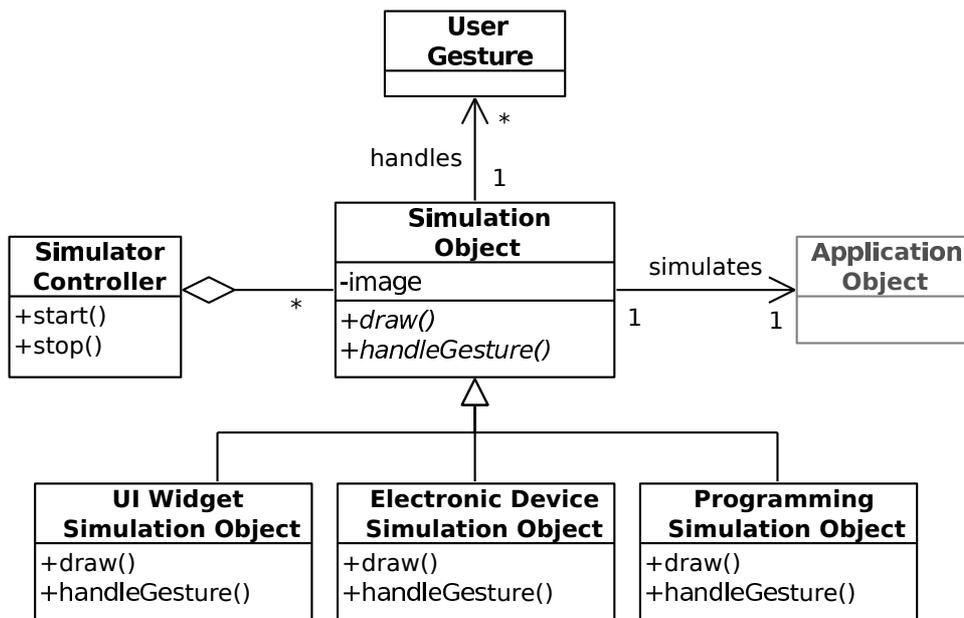
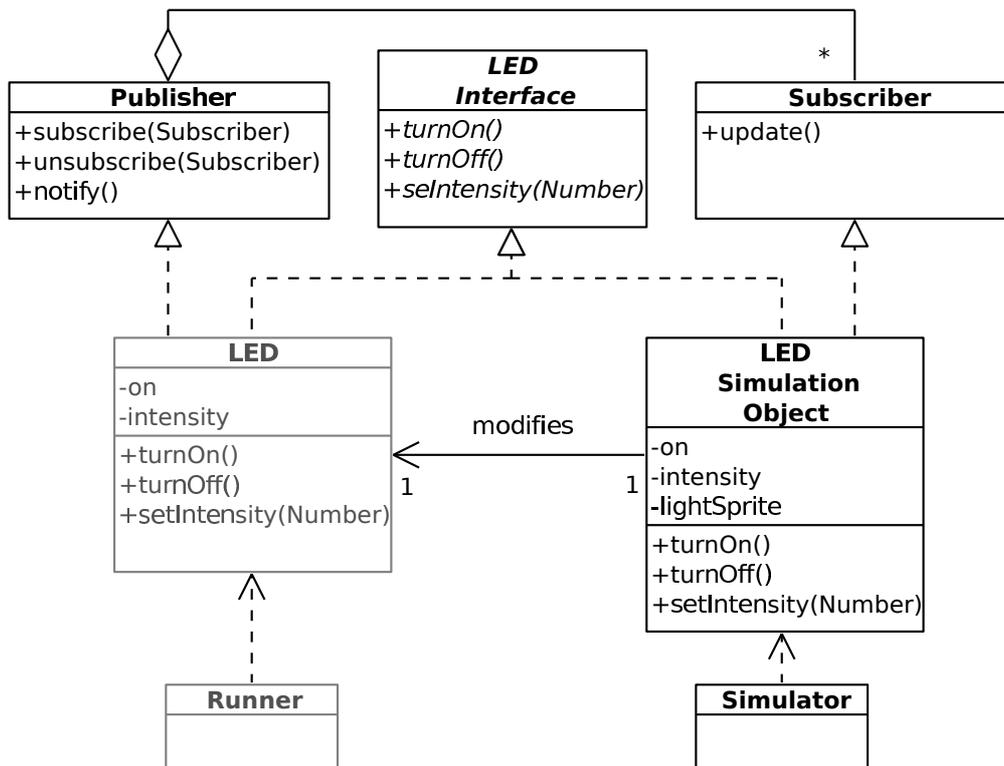


Figure 6.7: Main classes in the *Simulator*.

Simulation Objects are designed following the Decorator and Observer design patterns [36]. The Decorator design pattern enables adding behavior to *Application Objects* without modifying them. In agreement with the Decorator pattern, *Application Objects* and *Simulation Objects* share a common interface. Having *Application Objects* and *Simulation Objects* share a common interface enables the *Simulator Controller* to interact with *Simulation Objects* in the same way as the *Runner* class from the *Running Engine* component would interact with an *Application Object*. *Simulation Objects* handle user input during simulation time and delegate the state of the *Application Object* accordingly. The Observer design pattern enables *Simulation Objects* to observe the state of *Application Objects*. *Simulation Objects* update their

Figure 6.8: Usage of the Decorator pattern in the *Simulator*.

visual representation to match the state of the *Application Objects* they represent.

Figure 6.8 illustrates the application of the Decorator and Observer patterns to extend the functionality of the *LED* for simulation purposes. The *LED* and *LED Simulation Object* share a common interface, called “*LED Interface*”. The *LED Simulation Object* handles user taps by invoking the *LED*’s *turnOn()* method. When the *LED* has been turned on, it notifies its observers about the state change. When notified about the *LED*’s state change, the *LED Simulation Object* displays a sprite of a light beam on top of the *LED Simulation Object*’s image. Note that the *LED Simulation Object* could directly respond to user taps by displaying and hiding the sprite. However, *Application Objects* might change their state independently from user input applied to a *Simulation Object*. Having the *Simulation Objects* respond to changes in *Application Objects*’ state ensures that *Simulation Objects* are updated independently of how the *Application Object*’s update happened.

6.3 Deployer

The *Deployer* executes on *Interactex Designer* and is responsible for two main tasks. First, the *Deployer* converts a *Project* into an executable *Application*. For each *Edition Object* in the *Project*, the *Deployer* instantiates an *Application Object*. If *Invocations* between *Edition Objects* have been defined in the *Project*, the references between them

have to be mapped to references between the corresponding *Application Objects* in the *Application*. After every *Application Object* has been created, the *Deployer* converts references between *Edition Objects* into references between *Application Objects*. In order to resolve a reference in constant $O(1)$ time, the *Deployer* uses the *Reference Lookup Table* that contains a reference to an *Application Object* for each *Edition Object*.

Second, the *Deployer* is responsible for serializing an application and uploading it to *Interactex Client*. The main class responsible for serializing and uploading applications is the *Deployer Controller*. The *Deployer Controller* is also the Facade of the *Deployer* and offers a single method to other software components to *deploy* an application. The *deploy* method first converts a *Project* into an executable *Application*, as described earlier in this section. Once the instance of the *Application* has been created, the *Deployer Controller* passes it over to the *Application Uploader*.

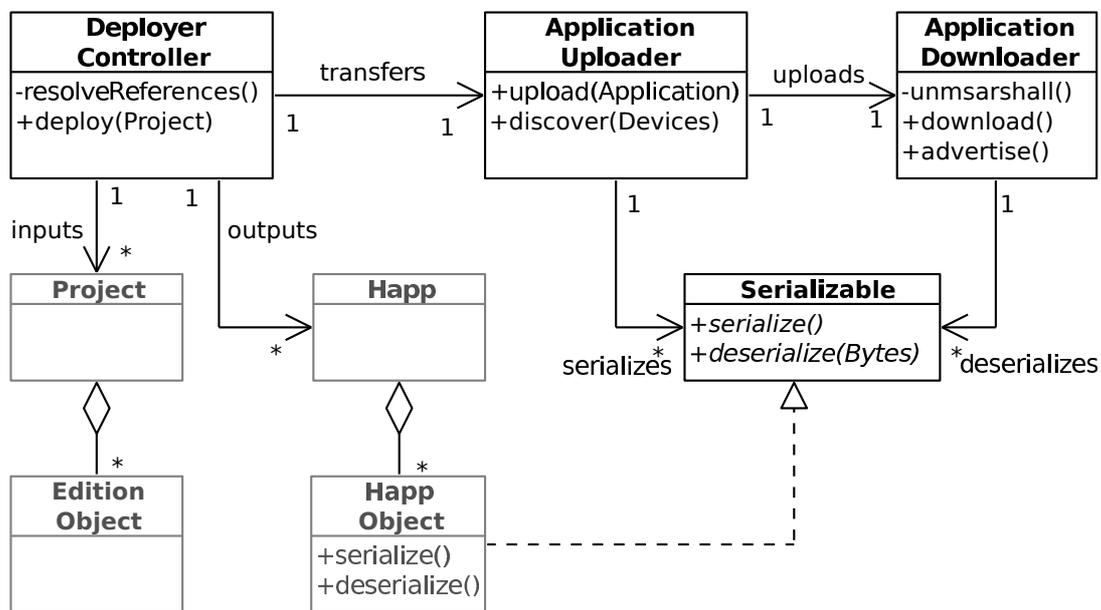


Figure 6.9: Main classes of the *Deployer*.

Every *Application Object* in the *THIDE* implements the *Serializable* protocol. The *Serializable* protocol contains a method to *serialize* and another to *deserialize* the *Application Object*. The *serialize* method is an instance method that returns a byte stream representing the instance. The *deserialize* method is a class method that creates an instance based on a byte stream. The *Application Uploader* first serializes an *Application* by invoking the *serialize* method of each *Application Object* and then transfers the *Application* to a smartphone running *Interactex Client*. The *Application Uploader* also discovers nearby smartphones running an instance of *Interactex Client*, serializes *Applications* and transfers a serialized *Application* to the *Application Downloader*.

The *Application Downloader* runs in *Interactex Client* and is responsible for downloading a byte stream and deserializing it into an *Application*. In order to deserialize an *Application*, the *Deployer Controller* invokes the *deserialize()* method of every

Application Object.

6.4 Plugin Editor

The *Plugin Editor* offers an interface to create code plugins in *TextIT* that can be imported into *Interactex*. In order to create a plugin, developers drag and drop *TextIT* Objects from a palette into the *Canvas*. There are two main categories of objects, the *Control Objects* and the *Visualization Objects*. *Control Objects* execute code that modifies input data and *Visualization Objects* display input data without modifying it. With exception of the *Library Manager*, every object in *TextIT* has input and output *Ports* used to send and receive data to other objects. Developers define the flow of data between *TextIT* Objects by creating *Connections*. *Connections* connect the output *Port* of a *TextIT* Object to the input *Port* of another *TextIT* Object.

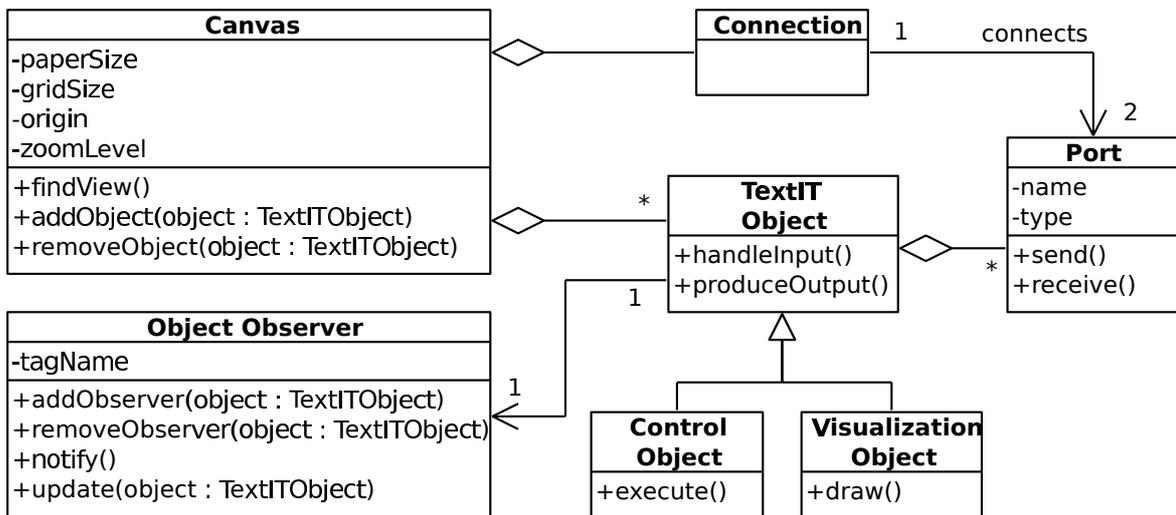


Figure 6.10: Main classes in the *Plugin Editor*.

The *Plugin Editor* follows the Observer pattern to handle the flow of data between *TextIT* Objects. When a *Connection* between two *TextIT* Objects is defined, the receiving *TextIT* Object subscribes to the receiving object's *Object Observer*. Figure 6.10 displays the main classes of the *Plugin Editor* and their relationship.

Next, we describe every *Control Object* in *TextIT*.

- The *Start Object* initiates execution of an application. The *Start Object* does not have input ports and has a single output port that triggers an event when the *Start Object's* button is clicked.
- The *Data Source* accesses files containing data in JSON format. The *Data Source* contains an input port that can be triggered by other objects to load the data file and an output port that delivers the data organized as a keys/values dictionary.

- The *Conditional Statement* is equivalent to an if/else condition in text-based programming. It is similar to the *Code Editor* in that it also offers a view with syntax highlighting where developers can edit and debug code. However, it differs from the *Code Editor* in that its output is restricted to a single value of Boolean type.
- The *Iterator* iterates through an array or list of data. The *Iterator* accepts an input of JavaScript types List or Array and has two output ports. The *Iterator*'s "result" output port delivers elements of the input array or list sequentially, together with their index within the array or list. The "finished" output port is triggered after the last element of the list or array has been reached.
- The *Delayer* is used to delay the internal execution of the object it connects to. The delay time is specified by developers. The *Delayer* is useful to simulate sensor sampling rates and to deliver data at a convenient speed to *Visualization Objects*.
- The *Code Editor* enables code-based programming and features syntax highlighting and debugging functionality. The *Code Editor* has a single input port, which accepts any type of data. Output ports are generated dynamically by the *Code Editor* after execution of the code.
- The *Library Manager* enables developers to specify which libraries will be needed at runtime. Libraries are JavaScript files containing JavaScript functions. When loaded, every function of the imported library becomes available to every other object in the canvas. The *Library Manager* has no input or output ports.

TextIT's *Visualization Objects* are:

- *Labels* display a value and have an input port providing the value.
- The *Line Chart* displays a series of 2D points in a line chart. The *Line Chart* has an input port that receives an array or pairs of values representing a point. Developers can decide whether data is fed as single array or as a series of value pairs. Alternatively, developers can also configure the *Line Chart* to use "time" as one of the axis, and supply only single values.
- The *Pie Chart* displays data as a pie chart. The *Pie Chart* has an input port that accepts key-value pairs. The *Pie Chart* can be used to compare values (e.g. the accuracies of two algorithms).
- The *Scatter Chart* is similar to the *Line Chart* in terms of input and output ports. However, the *Scatter Chart* displays the input values as non-connected points.
- *Gauge Charts* display a numeric value that is known to be within a range. The value is provided by the *Gauge Chart*'s single input port.

- *3D Viewers* render an object in a 3D space. Object rotation values are passed to the *3D Viewer* in the form of quaternions. The *3D Viewer* might be helpful to understand body postures and limb movements.
- *JSON Viewers* provide a visualization of data in the JSON format. The *JSON Viewer* receives a dictionary over its single input port and displays it as nested key/value pairs. Because data between *TextIT Objects* is exchanged in a JSON format, the *JSON Viewer* might be useful for debugging *TextIT* applications.
- The *Table Viewer* displays data in a tabular format. The *Table Viewer* has a single port that accepts 2D arrays and an output port that triggers after the table data has been rendered on screen.
- *Video Players* are used to play video files that are loaded from the developer's filesystem. *Video Players* have an input port to control the video (i.e. start, stop) and an output port that emits events to communicate the state of the player (i.e. started, stopped). Furthermore, while playing, the *Video Player* sends the elapsed time through an output port called *tick*. *Video Players* are useful to relate sensor values and algorithm results to situations on the real world. For example, displaying a slow-motion video of a user jumping next to the motion data collected during the jump might help developers relate the specific phases of the jump (jump preparation, take off, height peak, landing) to the signal.

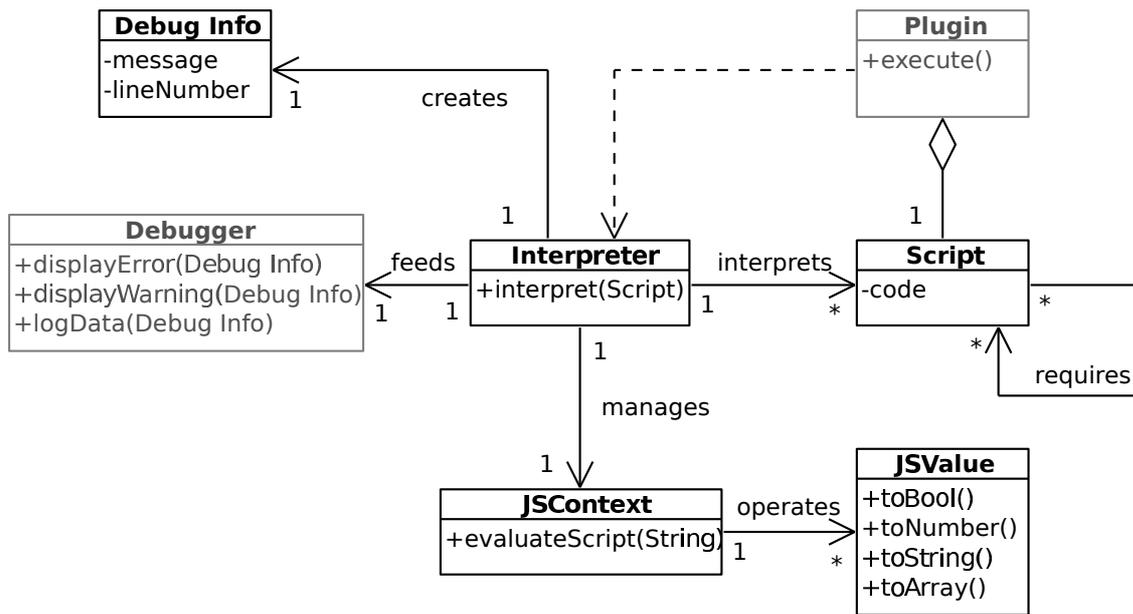
Every *Visualization Object* has an output port that triggers an event when the last data sample has been rendered.

6.5 Plugin Interpreter

The *Plugin Interpreter* executes JavaScript plugins developed in *TextIT*. The *Interpreter* class is the Facade of the *Plugin Interpreter* [36] and offers a service to other components for interpreting *Plugins*. *Plugins* have code, which is contained within the *Script* class. A *Script* might reuse code contained in other *Scripts*. The *Plugin* class is a subclass of *Happ Object*. This allows *Plugins* to be executed by the *Running Engine* in the same way any other *Happ Object*. When a *Plugin's* *execute()* method is invoked by another *Happ Object*, the *Plugin* uses the *Interpreter* to execute the code contained in its *Script* instance. The *Interpreter* delegates the execution of *Scripts* to the *JSContext*.

The *JSContext* is a class provided by Apple's *JavaScriptCore* framework that represents a JavaScript execution environment. The data used by the *JSContext* class is stored in instances of the *JSValue* class. An instance of the *JSValue* class represents a JavaScript variable. Types of variables supported by the *JavaScriptCore* framework include numbers, arrays, objects and functions.

The *JSContext* produces execution information when it executes JavaScript code. The *Interpreter* parses the execution results provided by the *JSContext* and creates

Figure 6.11: Main classes of the *Plugin Interpreter*.

an instance of the *Debug Info* class for each error, warning and output message. The *Debug Info* contains the message represented in a *String* and the line number where the message was originated. The *Interpreter* passes instances of *Debug Info* over to the *Debugger*. The *Debugger* displays syntax errors, warnings and output messages in order for developers to identify issues in the code syntax and undesired runtime behavior. The main classes of the *Plugin Interpreter* and their relationships are shown in Figure 6.11.

6.6 Firmata Library

The *Firmata Library* is the closest component to the *Hardware Layer* and acts as an adapter between the *Execution Layer* and the *Hardware Layer*. The *Firmata Library* transforms high-level operations with hardware devices into instructions that the microcontroller attached to the smart textile is able to execute. For example, an operation to turn an LED on will be transformed by the *Firmata Library* into an instruction to execute a digital output command on the microcontroller pin connected to the LED.

The communication between the mobile device and the microcontroller attached to the smart textile conforms to the Firmata protocol¹. Firmata is a generic protocol that enables the communication between a microcontroller and a host computer. The *Firmata* class is the Facade of the component and offers a set of high-level methods to send messages to the microcontroller and to handle messages received from it. Internally, *Firmata* converts high-level method calls into sequences of bytes and parses

¹<http://www.firmata.org>

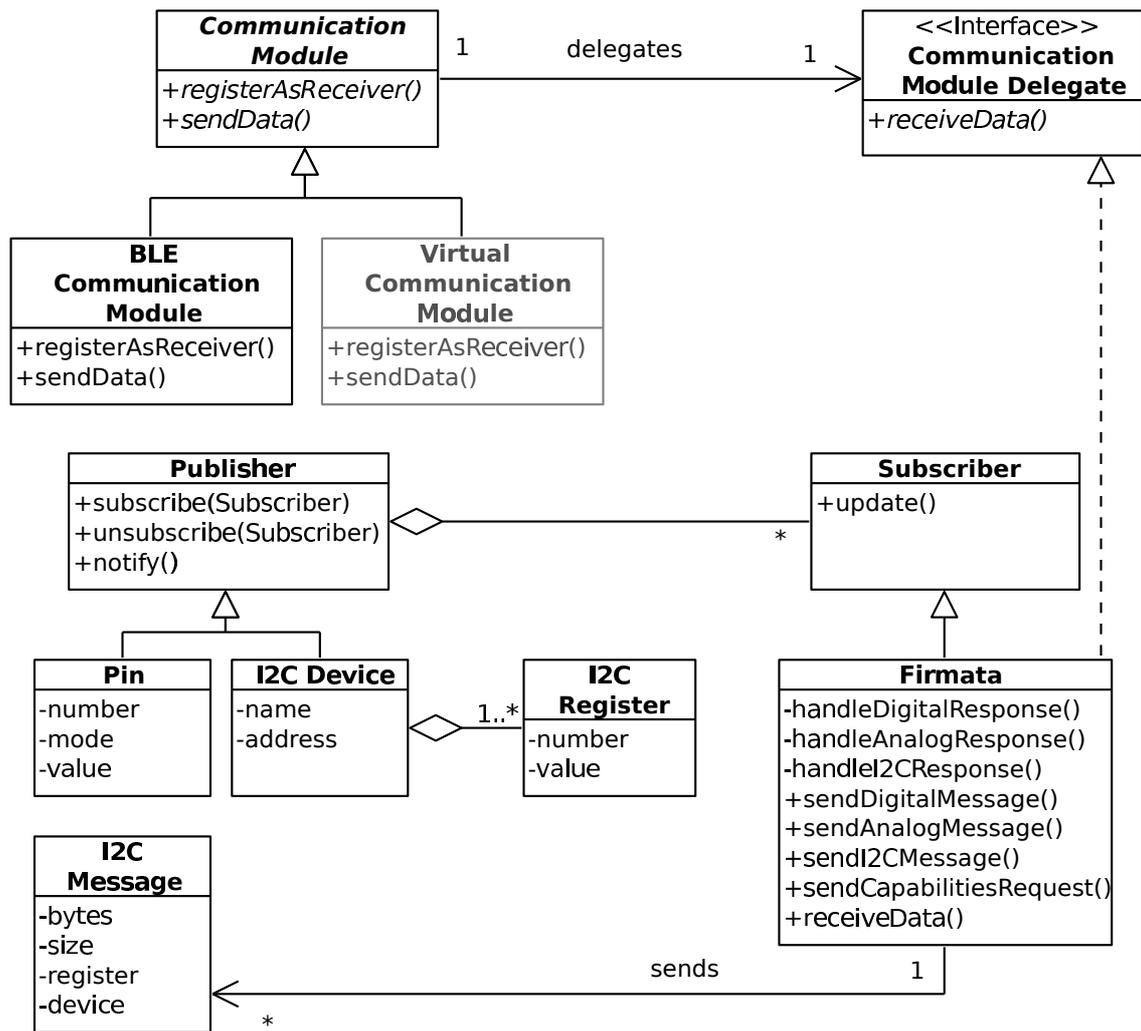


Figure 6.12: Main classes of the *Firmata Library*. The *Virtual Communication Module* belongs to the *Simulator*.

sequences of bytes in order to invoke the appropriate handler method.

The *Communication Module* defines an interface for sending and receiving data. There are currently two implementations of the *Communication Module*. The *BLE Communication Module* sends data over the device’s Bluetooth Low Energy interface. The *Virtual Communication Module* belongs to the *Simulator* software component and is used to display data on a debugging interface in order for developers to keep track of messages sent to the microcontroller.

The *Firmata Library* follows the Observer [36] and the Delegation patterns for sending data to the microcontroller and handling received data. The Observer pattern is used by the *Firmata* class, which observes for changes in *Pins* and *I2C Devices* and synchronizes their state with the actual hardware. The Delegation pattern is used between the *Communication Module* and the *Firmata* class. The *Communication Module* delegates data to *Firmata* without knowing the details of its implementation besides the fact that it implements the *Communication Module Delegate* interface.

CHAPTER 6. OBJECT DESIGN

The *Communication Module Delegate* interface enforces the implementation of the *receiveData()* method, which handles data received over the current *Communication Module*. The use of the Observer and Delegation patterns decouples the *Pin*, *I2C Device* and the *Communication Module* classes from the *Firmata* class. Furthermore, the usage of these design patterns improves maintainability of the *Firmata* class and reusability of the classes it communicates with. An UML model of the *Firmata Library* is shown in Figure 6.12.

Chapter 7

User Interface

In Chapters 5 and 6 we described system and object design of the *THIDE*. The *THIDE* implemented for iOS and Arduino consists of two main tools: *Interactex* and *TextIT*. This chapter describes every view of *Interactex* and *TextIT* and the navigation between them. Furthermore, at the end of this chapter, we present a step by step description of how a smart textile application is developed in the *THIDE*.

7.1 Interactex Designer

Interactex Designer has three main views. The first view the user sees when the application has been opened is the *Project Selection Screen*. The *Project Selection Screen* enables developers to start a new project and select already existing projects to continue working on them. Developers use the *Editor Screen* to develop applications for smart textiles and the *Simulator Screen* to test applications. Usually, developers open a project and then navigate back and forth between *Editor Screen*, where they add new functionality, and the *Simulator Screen*, where they test the latest developed functionality.

7.1.1 Project Selection Screen

The *Project Selection Screen* displays the projects stored in the iPad's hard drive. Projects can be arranged on a grid or list fashion. When projects are arranged as a grid fashion, a screenshot of each project's last state and the project's name are displayed. When projects are displayed as a list, only their name and creation date are shown. The navigation bar at the top of the *Project Selection Screen* contains two buttons: *Edit* and *New*. The *Edit* button turns displayed projects into *editable mode*. In *editable mode*, projects can be renamed, rearranged, duplicated and removed. The *New* button starts an empty project. A project is selected by tapping on its screenshot. Selecting an existing project or starting a new project lead to the *Editor Screen*. Figure 7.1 shows a screenshot of the *Project Selection Screen*.

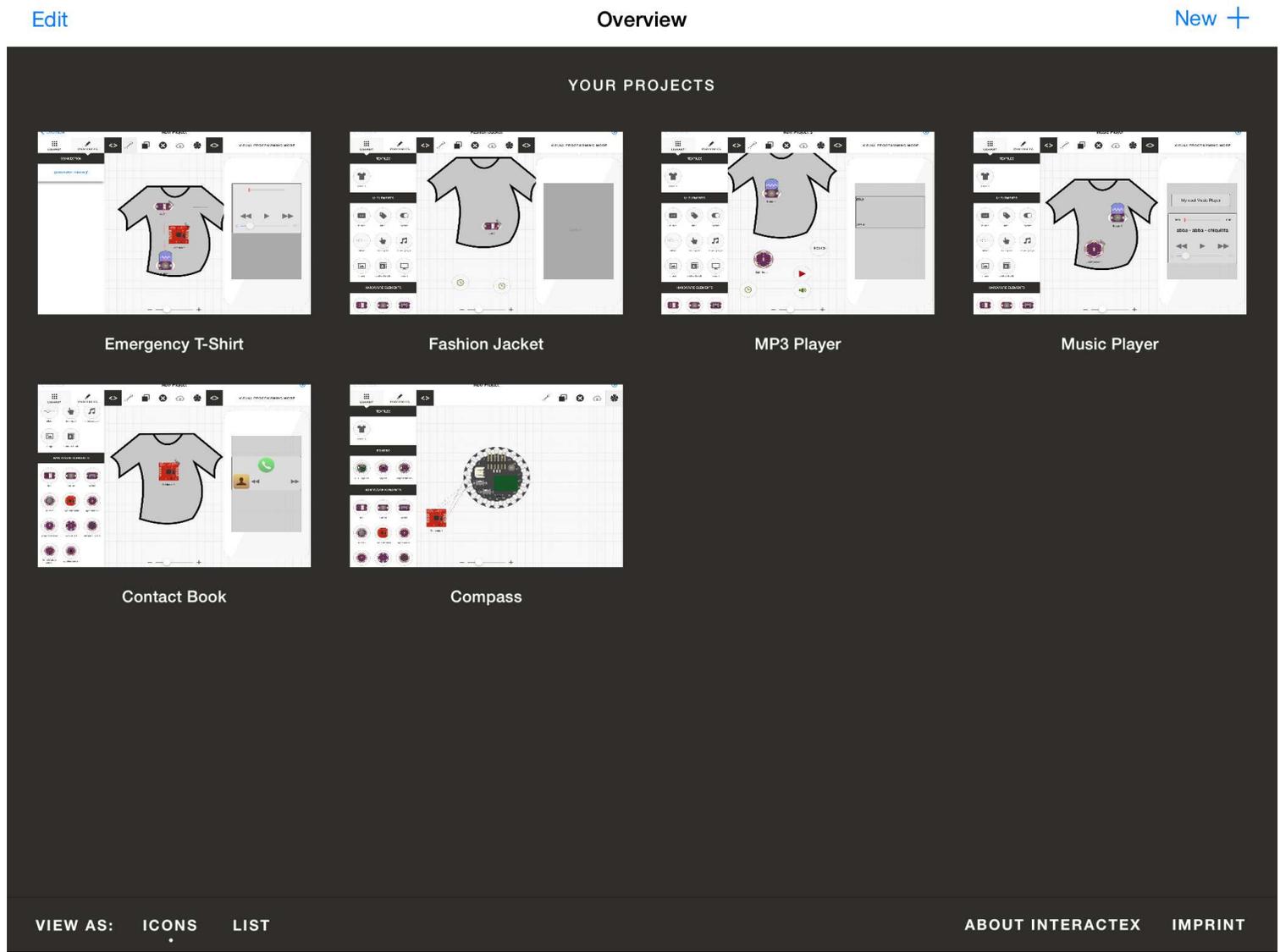


Figure 7.1: *Interactex Designer's Project Selection Screen.*

7.1.2 Editor Screen

The *Editor Screen* is the main interface for developers to create an *Application* for smart textiles. The main components of the *Editor Screen* are the *Canvas*, the *Palette* and the *Toolbar*, which we have already introduced in Section 6.1. In order to create an *Application*, developers drag elements from the *Palette* into the *Canvas*.

The *Palette* has two main tabulations labelled *Library* and *Properties*. The *Library* tabulation displays the *Objects View* and the *Properties* tabulation displays the *Configuration View*. Figures A.1 and A.2 in the Appendix display the *Objects View* and Figure A.3 displays the properties of a *Label* and *Timer* objects.

The *Toolbar* contains five buttons to edit and manipulate objects in the *Canvas* and to perform specific actions on the current application. Table 7.1 lists the different buttons available in the *Toolbar* and describes their functions. The *Simulation Button* is used to start and stop the simulation of the current application. Starting a simulation hides the *Palette* and every connection between objects. Figure 7.2 shows an overview of *Interactex Designer's Editor Screen*.

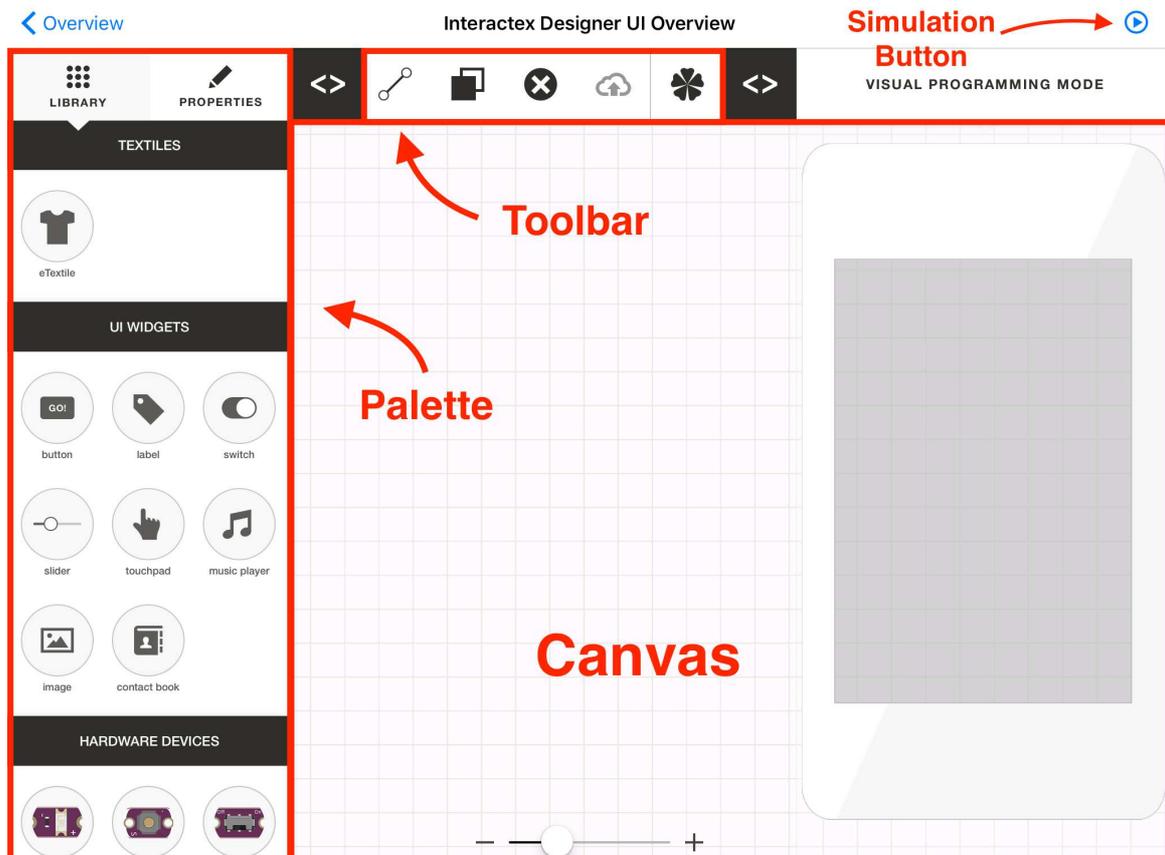


Figure 7.2: *Interactex Designer* UI overview.

Image	Name	Description
	Connect	Sets the <i>Editor</i> in <i>Connection</i> mode so that lines can be drawn between objects in order to create <i>Invocations</i> .
	Duplicate	Sets the <i>Editor</i> in <i>Duplicate</i> mode causing the creation of copies of objects dragged in the <i>Canvas</i> .
	Delete	Sets the <i>Editor</i> in <i>Delete</i> mode. During <i>Delete</i> mode, user taps on objects on the <i>Canvas</i> causes objects to be deleted.
	Circuit Layout	Switches <i>Editor</i> to the circuit layout view.
	Deploy	Deploys the project into <i>Interactex Client</i> . The <i>Deploy</i> toolbar button becomes enabled when an instance of the <i>Interactex Client</i> is found nearby.

Table 7.1: *Toolbar Buttons in Interactex Designer.*

7.1.3 Canvas

The *Canvas* is where the visual programming in *Interactex* takes place. The *Canvas* displays *Edition Objects* and *Invocations* between them. Figure 7.3 shows the *Invocations* between a *Button* and a *LED*. Developers draw lines between an *Edition Object's Event Hook* and another *Edition Object's Method Hook* in order to define a new *Invocation*. *Hooks* display different images depending on the type of parameters provided by an *Event* or required by a *Method*. *Hook* images also indicate whether the parameters provided by the *Event* comply with those required by the *Method* (filled shapes) or not (hollow shapes). Table 7.2 displays the different images used by *Hooks*.

Type	Compliant Parameters	Non-compliant Parameters
Boolean		
Numeric		
String		
Object		

Table 7.2: Images of *Hooks* in *Interactex Designer*. Filled shapes indicate that a parameter provided by an *Event* comply those expected by a *Method*. Hollow shapes indicate that the *Event* does not provide a parameter of the type expected by the *Method*.

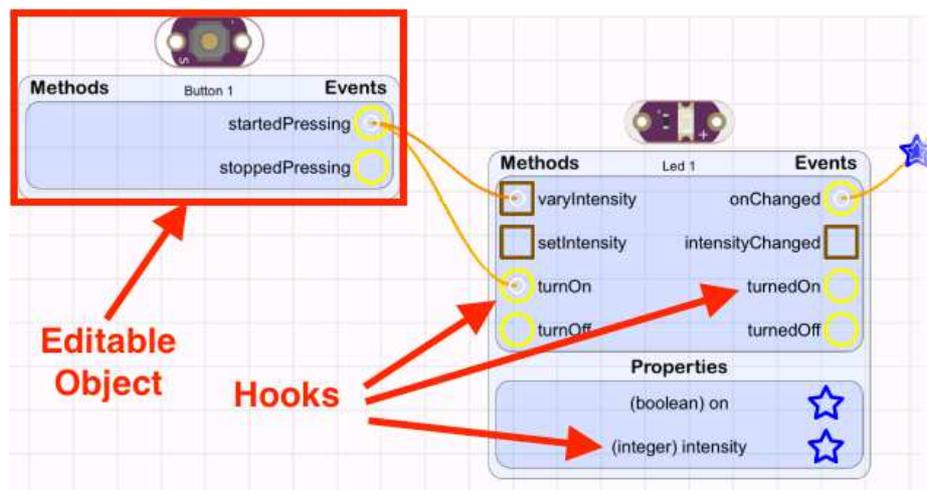


Figure 7.3: *Invocations* between *Events* of a *Button* and *Methods* of an *LED*.

7.1.4 Circuit Layout

The *Canvas* is also where developers create the circuit layout of a smart textile. Figure 7.4 shows the circuit layout of the CTS Gauntlet, described in Section 1.1. Circuit layouts are created by drawing connections between *Hardware Devices' Pins*. *Hardware Devices' Pins* that can be assigned to only one *Microcontroller Pin* (e.g. the power pin of a *Button*) are wired automatically when the *Hardware Device* is added to the *Canvas*. The *Canvas* can be zoomed in and out to facilitate the accurate drawing of connections.

Different types of connections are drawn in different colors. Connections to a *Power Pin* are drawn in red; connections to a *Ground Pin* are drawn in gray and *Connections* between *Data Pins* are drawn in purple by default and can be configured by developers over the *Properties View*.

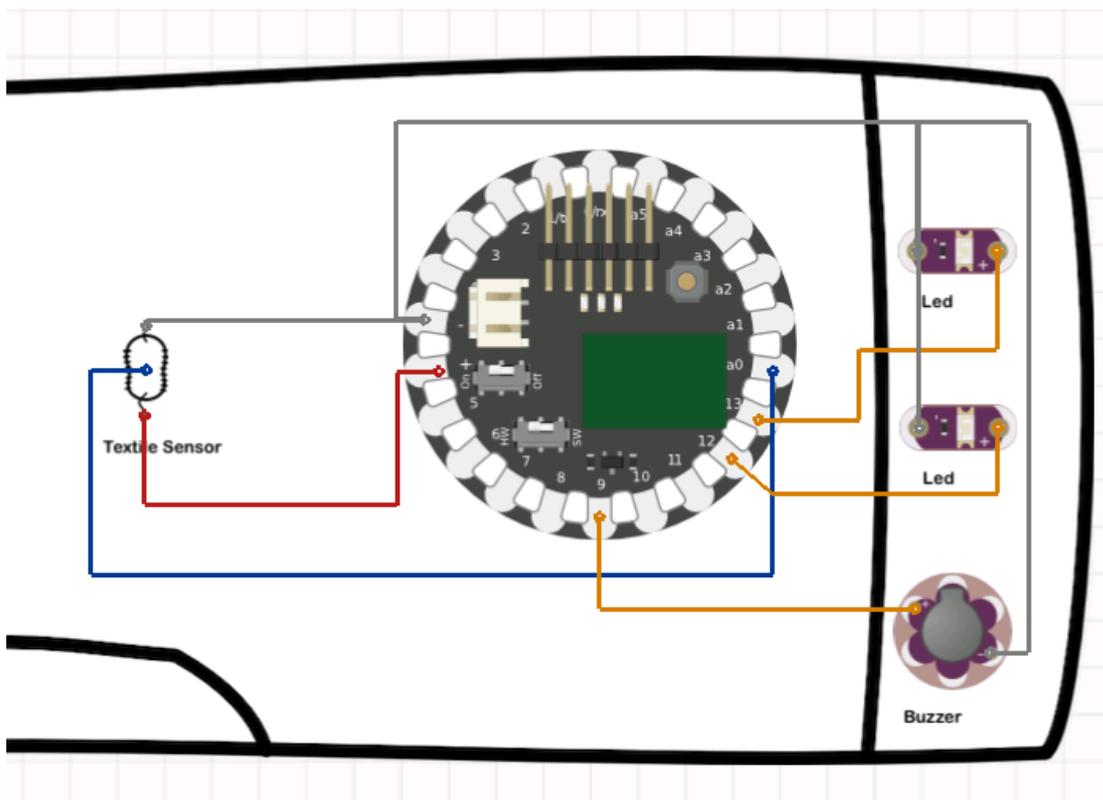


Figure 7.4: Circuit layout of the *CTS-Gauntlet* developed in *Interactex Designer*.

7.1.5 Simulator Screen

The *Simulator Screen* displays an interactive representation of the smart textile and smartphone. Multi-touch gestures and iPad sensors are used to simulate sensor values. Touching (covering) a light sensor causes the light intensity to decrease. The

analog textile sensor displays a handle to set its value. Accelerometers and compasses attached to a smart textile respond to acceleration and compass readings measured by the tablet device. Behavior of output devices is represented with animations, sounds and images. Vibration motors shake and emit a noise with an intensity and sound frequency proportional to their vibration intensity. LEDs display a semitransparent image of light that gets brighter the higher the LED's intensity is. Sensors and variables display their values on the *Canvas* for debugging purposes. Figure 7.5 displays different objects in *Simulator Screen*.

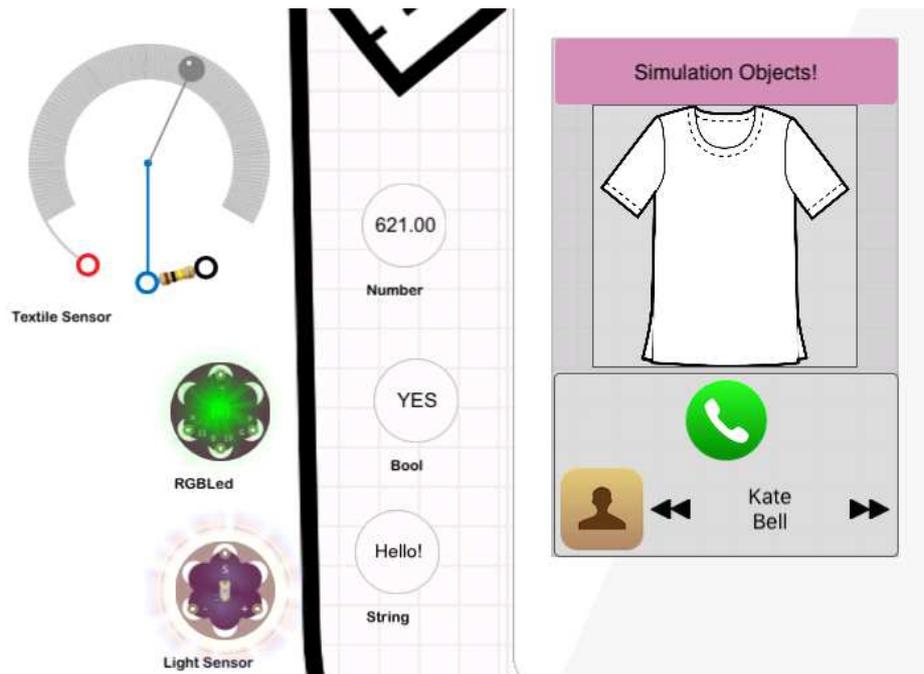


Figure 7.5: Simulation of different objects in *Interactex Designer*.

7.2 Interactex Client

Interactex Client has four screens. The *Download Screen* is used to download applications from *Interactex Designer*. The *User Applications* and *Default Applications Screens* are used to choose and open an application. The *Application Screen* displays the application.

7.2.1 Download Screen

The *Download Screen* is used to download applications from *Interactex Designer*. When the user enters the *Download View*, *Interactex Client* begins advertising itself to nearby instances of *Interactex Designer* using Bluetooth or Wi-Fi. The *Download*

Screen displays a *status* label to inform users about the connection state. The *status* label displays the following connection states: “Connecting”, “Connection Request Sent”, “Connected”, “Downloading Project”, “Download Finished”. Once an application download started, the *Download Screen* displays the application’s name and download progress in addition to the *status* label. Applications that finished downloading appear in the *User Application Screen*.

7.2.2 User Applications Screen

The *User Application Screen* displays the projects downloaded from the *Interactex Designer* arranged into a grid or list, configurable by users. Projects are represented with an icon and a name. If no project has been downloaded into Interactex Client, the grid or list are empty and instead a label is shown to users with the message: “No projects available.”. The *User Application Screen* displays a navigation bar at the top of the screen with two buttons: *Edit* and *+*. The *Edit* button enables users to change applications’ names and to delete them. The *+* button is used to navigate to the *Download View*. Tapping on an application’s icon leads to the *Application View*. A screenshot of the *User Application Screen* is shown in Figure 7.6 a).

7.2.3 Default Applications Screen

Interactex Client offers nine default applications for testing the communication between the smartphone and the smart textile and proper functioning of *Interactex Firmware*. The default applications are:

- Digital Output. Two *Buttons* on the smartphone’s interface are used to turn on and off an *LED*.
- Digital Input. A *Label* displays the state of a *Button* attached to the textile.
- Buzzer. Two *Buttons* and a *Slider* are used to turn on and off a *Buzzer* and to control its vibration frequency.
- Analog Input. A *Label* displays the value of an analog sensor (e.g. *Temperature Sensor*).
- Three Color LED. Three *Sliders* enable the user to control the color of a *Three-Color LED*.
- LSM303. Three *Labels* display the accelerometer values along x y and z-axes delivered by the *LSM303* sensor.
- Accelerometer. Three *Labels* display the accelerometer values along x y and z-axes delivered by the *Accelerometer* sensor.

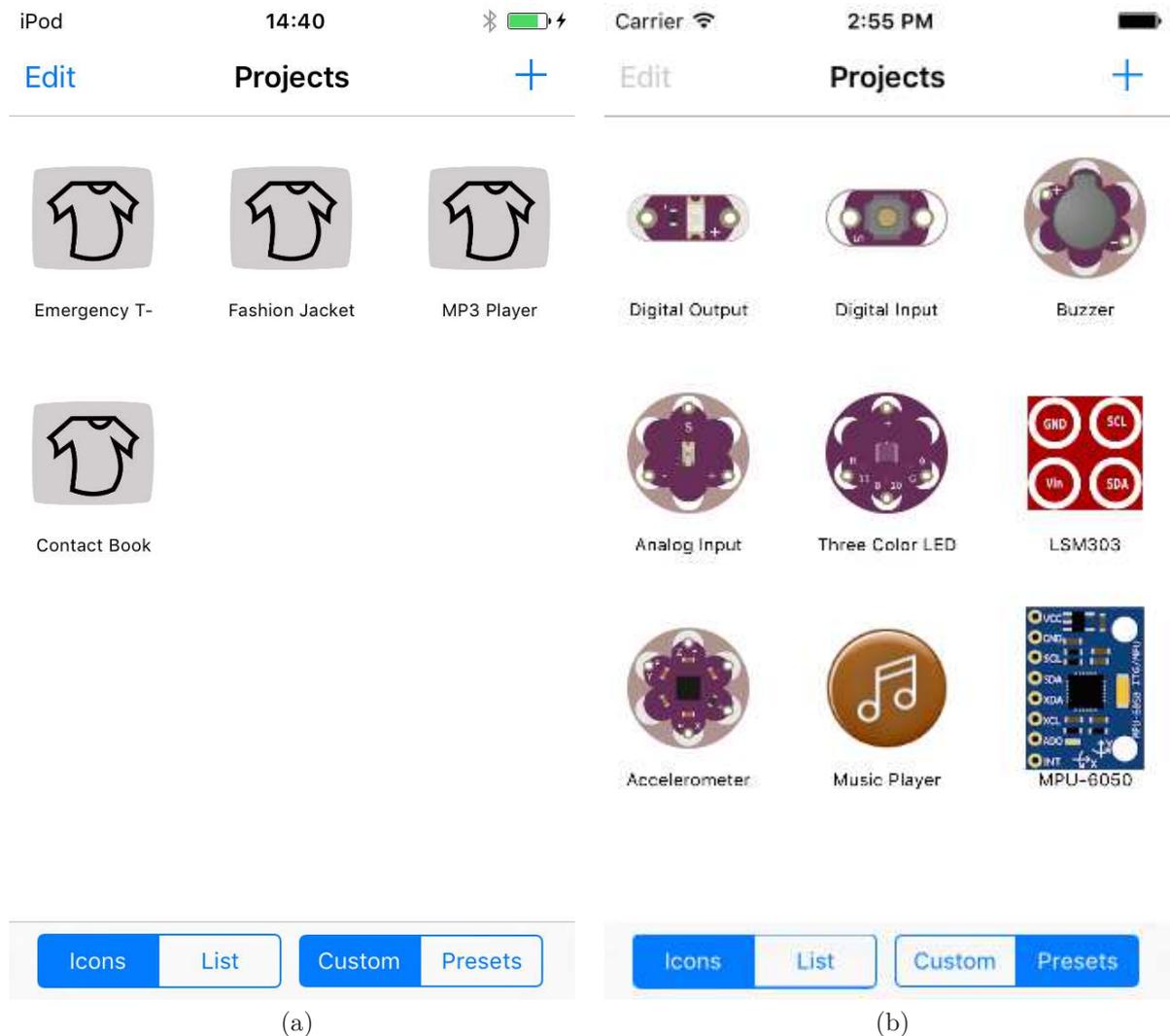


Figure 7.6: *Interactex Client's User Application Screen (a) and Default Application Screen (b).*

- Music Player. A *Button* attached to the smart textile is used to start and stop the music on the smartphone.
- MPU 6050. Three *Labels* display the linear acceleration values along x y and z-axes delivered by the *MPU 6050* sensor.

The default applications provide information to developers about how to connect the electronic devices to the microcontroller. The *Default Application Screen* is displayed in Figure 7.6 b).

7.2.4 Application Screen

The *Application Screen* displays the smartphone’s user interface developed in *Interactex Designer*. The *smartphone’s* user interface in *Interactex Client* have a high fidelity to the design created in *Interactex Designer* because both tools use the same *UI Widgets* available in Apple’s *UIKit* framework. In addition to the user interface developed in *Interactex Designer*, the *Application Screen* displays a navigation bar at the top of the view containing two buttons. The *Back* button is used to navigate back to the project selection screen and the *Connect* button connects the smartphone with the smart textile over Bluetooth Low Energy and starts the execution of the application.

7.3 TextIT

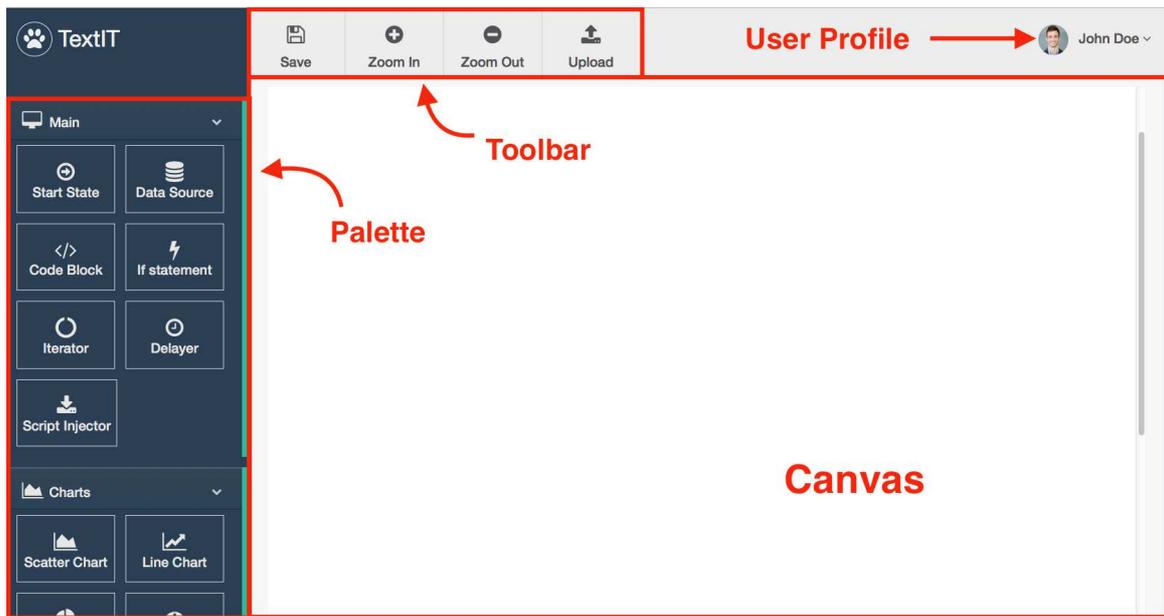
TextIT’s user interface was designed to be consistent with *Interactex Designer’s* user interface. *TextIT* also has a *Canvas*, a *Palette* and a *Toolbar*. An annotated screenshot of *TextIT’s* user interface is shown in Figure 7.7. The *Palette* contains *Palette Items* for every *Control* and *Visualization Object* available in *TextIT*. A full list of *TextIT’s* *Control* and *Visualization Objects* is provided in Section A.5 in the Appendix. *TextIT* *Objects* are added to the *Canvas* by clicking on the appropriate *Palette Item*.

Toolbar Buttons are used to save the state of the current project, zoom the *Canvas* in and out and upload the current project to *Interactex Designer*. The *upload* button only becomes available when a nearby instance of *Interactex Designer* is detected. Clicking on the upload button sends the code in every *Code Editor* object to *Interactex Designer*.

Figure 7.8 displays the main elements of *TextIT* objects. *TextIT* objects are displayed as a box with a name. With exception of the *Library Manager*, every object has input and output *Ports*. Input *Ports* are displayed as a green circle above the element’s box, and output *Ports* are displayed as red dots below the element’s box. Furthermore, every element has an icon, an *edit* button to open the *Edit Window* and a cross to remove the element. The *Edit Window* enables developers to configure the behavior of the object. For example, the *Edit Window* of the *Scatter Chart* enables developers to define the data series and axis labels.

7.4 Usage Example

In this section, we demonstrate the usage of *Interactex* by describing how *WaveCap* is developed, which we have described in Table 1.1. We then extend the *Interactex* application with a *TextIT* plugin that estimates the jogging speed based on motion data. The speed estimation is used to adapt the volume of the music player so that the faster the user runs, the louder the music is played.

Figure 7.7: Overview of *TextIT*'s user interface.

7.4.1 Development of an Interactex Application

Figure 7.9 shows the *WaveCap* project implemented in *Interactex Designer*. The application consists of a textile, four *Hardware Devices* and eight *UI Widgets*. We used the following *Hardware Devices*: one *Textile Speaker*, two *Textile Sensors* and a *Switch* and the following *UI Widgets*: 5 *Labels*, two *Switches* and a *Slider*. The behavior of the application is determined by six *Invocations* between objects. The *Textile Speaker* can be turned on and off with a smartphone switch or with the hardware on/off switch. The *Frequency* textile sensor is coupled to the *Textile Speaker*'s frequency. The *Volume* textile sensor sets the value of a *Slider* on the smartphone. When the *Slider*'s value changes, the *Speaker Radio*'s volume is set. This makes it possible to set the volume by either pulling the strings attached to the hood or through the smartphone. Another smartphone switch sets the radio sender to AM or FM. Values passed in an *Invocation* are represented with a shape - yellow circles represent *Boolean* types and a brown squares represent *Numeric* types.

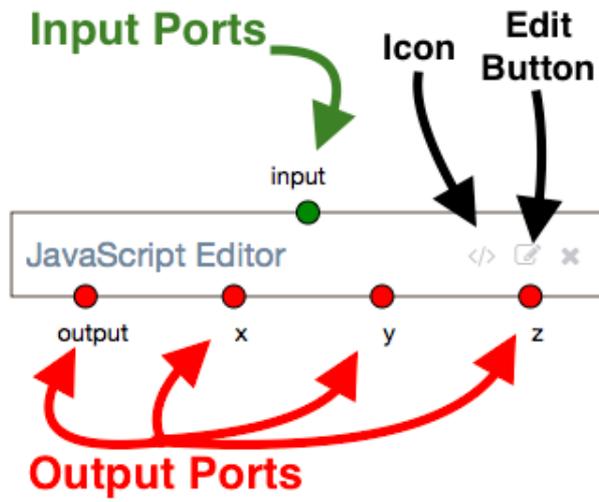


Figure 7.8: Visual components of a *TextIT Object*.

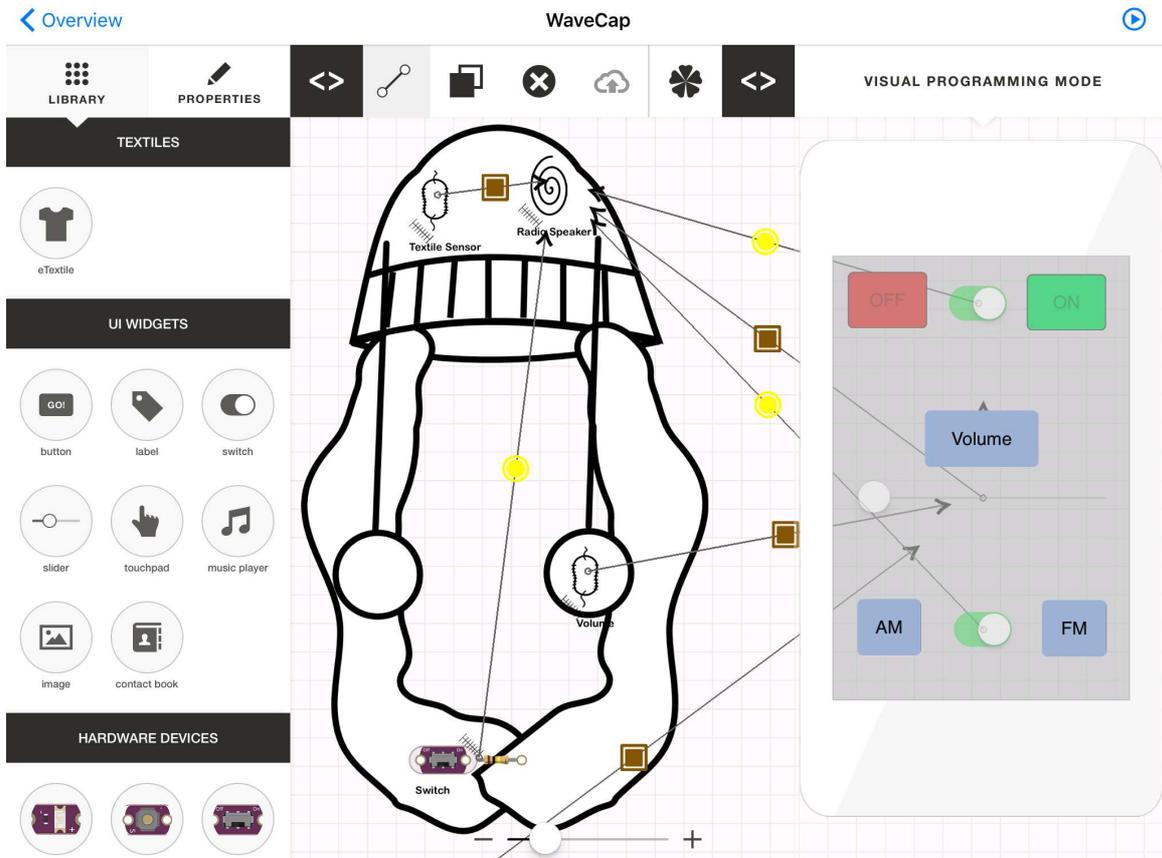


Figure 7.9: *WaveCap* implemented in *Interactex Designer*.

7.4.2 Development of a TextIT plugin

In order to develop an algorithm that estimates jogging speed, we start by displaying accelerometer data of a user jogging. A file containing the accelerometer data stored in JSON format can be imported into *TextIT* with the *Data Loader TextIT* object. After loading the file containing the jogging data, the *Data Loader* parses it and initializes the output ports named “*output*”, “*startTime*”, “*endTime*”, “*frequency*” and “*data*”. These ports correspond to keys in the JSON file. The *data* port provides the sensor signal. When the *Data Loader*’s *output* port is connected to the *Line Chart*’s data port, the *Line Chart* draws every sample in the signal as a line chart. The plot produced by the *Line Chart* is shown in Section A.5.2 in the Appendix.

The *Data Loader* is connected to a *Code Editor*, where the jogging speed is calculated. To estimate the user’s jogging speed, we filter the acceleration signal and compute the norm of the deviation vector. A *TextIT* project that loads the data, calculates and displays the jogging speed in a *Gauge Chart* is shown in Figure 7.10. The *Code Editor* contains the code shown in Figure 7.11.

After the plugin has been developed, it is uploaded to *Interactex* by simply clicking on the *Upload Button* on *TextIT*’s toolbar. Once imported, the plugin appears at the bottom of *Interactex*’s *Palette* with the name “*customComponent*”. We renamed the plugin to “*runningSpeed*” over the *Properties View* in the *Palette*. The *MPU 6050* accelerometer and gyroscope delivers linear acceleration values to the *Window* object. The *Window* object is configured over its *Properties View* to gather 50 acceleration samples. Once the 50th sample is collected, the *Window* emits the *filled* event, which triggers *runningSpeed*’s *execute()* Method passing in all 50 samples. Once the speed has been computed, *runningSpeed* triggers its *executionFinished()* event, which invokes the *setValue()* method of the *Mapper*. The *Mapper* maps the value provided by *runningSpeed* to a range between 0 and 1 and invokes the *Slider*’s *setValue()* method, passing in the normalized speed. The speaker’s volume is automatically adapted when the *Slider*’s value changes due to the previously defined *Invocation* between the *Slider*’s *valueChanged()* event and *Textile Speaker*’s *setVolume()* method.

Next, we enter *Hardware Mode*, add the *BLE-Lilypad* microcontroller to the *Wave-Cap* and draw connections between every *Hardware Component* and the appropriate pin on the *BLE-Lilypad*. Figure 7.13 displays *WaveCap*’s circuit.

The project can be simulated to ensure the application behaves as expected. The *Recorder Happ Object* can be used to record and replay jogging data. The *Textile Speaker* will shake to indicate it has been turned on. The *Textile Speaker*’s volume property can be displayed on a label to test whether the volume changes as expected. Finally, the project is uploaded to *Interactex Client* by tapping the *Deploy Button* on the toolbar. Once opened from the *User Applications Screen*, the project starts.

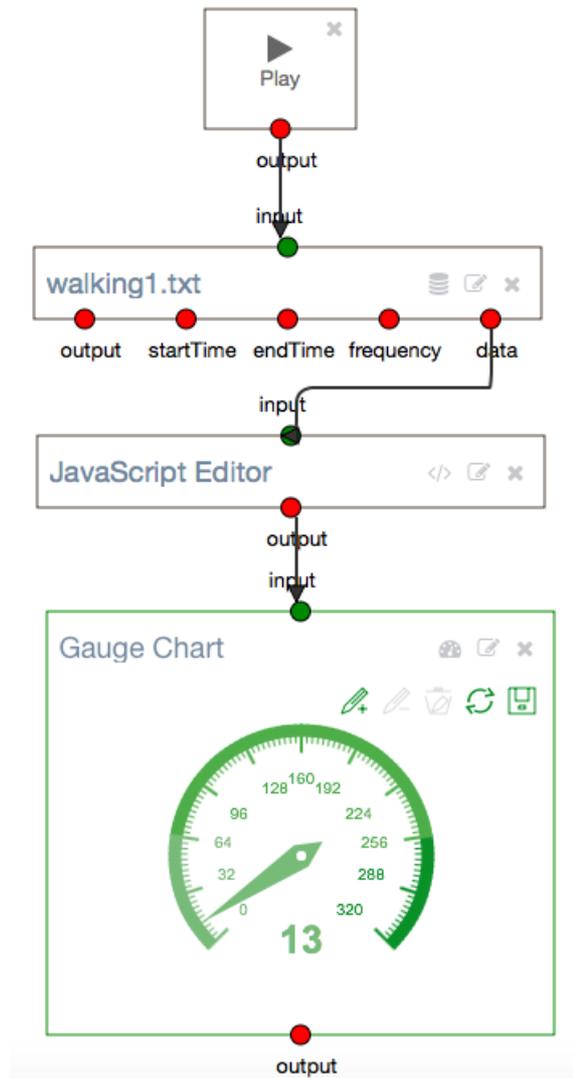


Figure 7.10: Development of an algorithm to detect jogging speed in *TextIT*.

```

1  var samples = data.input;
2
3  //low-pass filter the signal
4  var lowPassedSignal = Filter.lowPassFilter(samples);
5
6  //calculate deviation
7  var mean = FeatureExtractor.mean(lowPassedSignal);
8  var deviation = FeatureExtractor.deviation(lowPassedSignal, mean);
9
10 //calculate norm
11 var x = deviation.x;
12 var y = deviation.y;
13 var z = deviation.z;
14
15 var norm = sqrt(x*x + y*y + z*z);
16
17 //return norm
18 return norm;
19

```

Figure 7.11: *TextIT* code to estimate jogging speed for the *WaveCap* application.

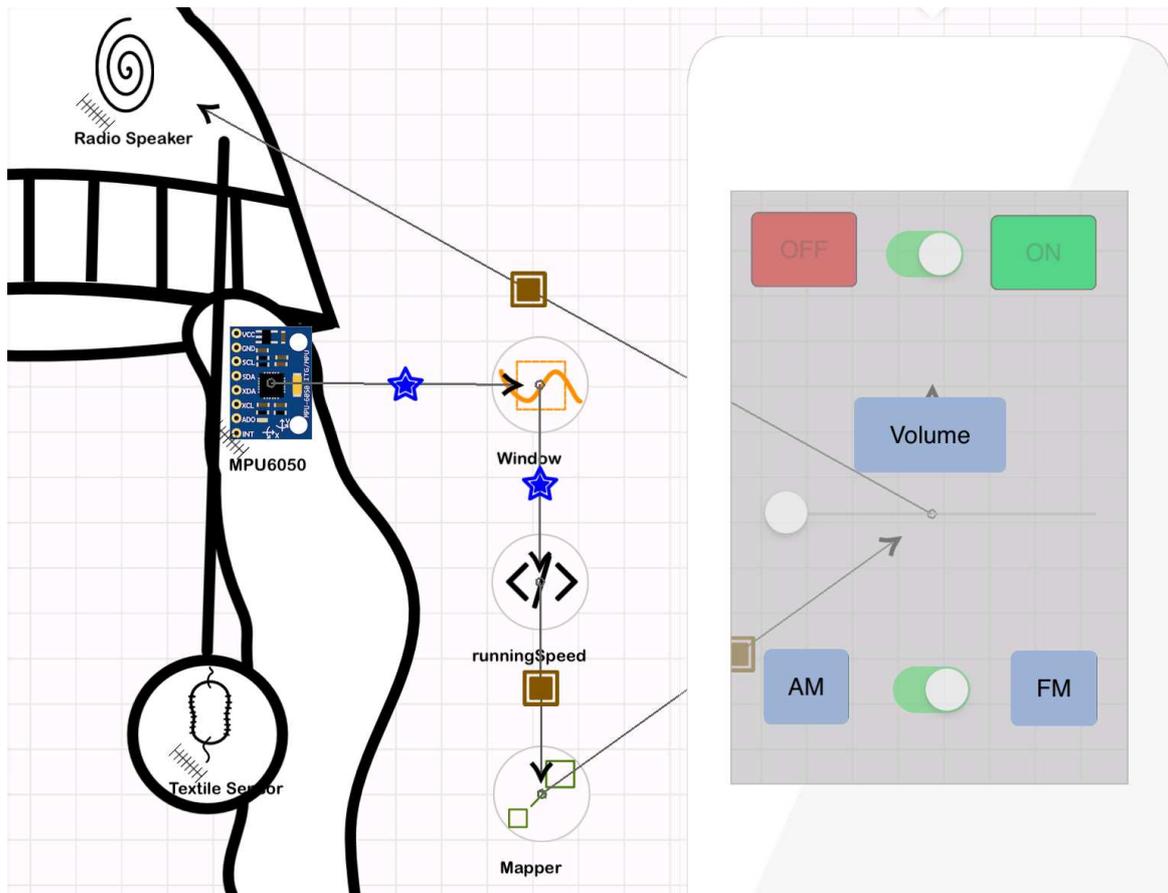


Figure 7.12: Usage of a *TextIT* plugin in *Interactex Designer* to control smartphone's volume based on estimated jogging speed.

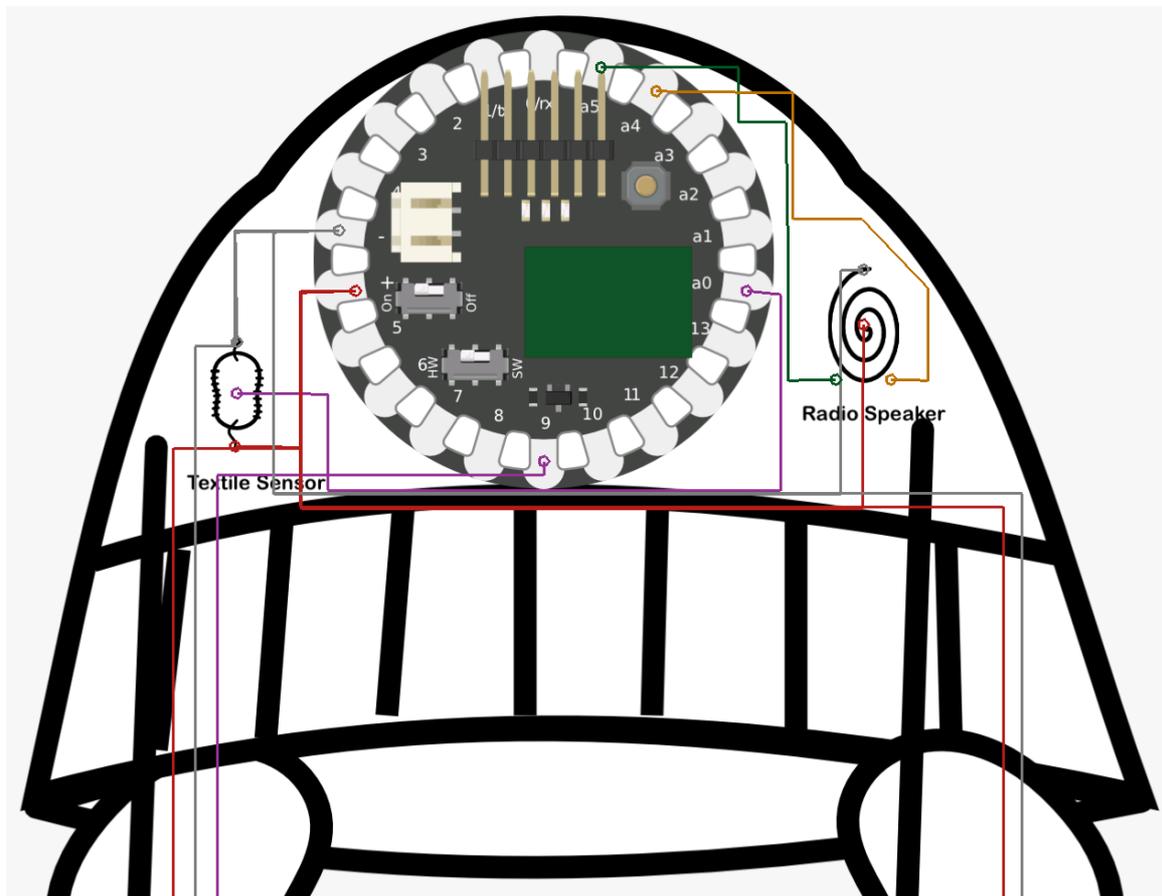


Figure 7.13: Circuit layout of *WaveCap* developed in *Interactex Designer*. *Hardware Devices* from left to right: a *Textile Sensor*, a *BLE-LilyPad* and a *Textile Speaker* (renamed to “Radio Speaker”).

Chapter 8

Applications

The *THF* and *THIDE* were developed iteratively based on a series of smart textile applications. In order to ensure a wide breadth of coverage of use cases, we have chosen applications from different domains. In this chapter, we describe two of the smart textile applications.

The *first* smart textile is the KneeHapp Bandage, a smart bandage that tracks the rehabilitation progress after a knee injury. The KneeHapp Bandage was developed at the Technical University of Munich during several student courses. The first prototype of the KneeHapp Bandage was developed and evaluated by a team of seven undergraduate students during the iOS lab course in the summer term 2014. The prototype was refined and extended with an insole with integrated pressure sensors by another team of eight students who took part in the iOS lab course winter term 2015. The insole is used to measure performance of jump-based exercises. For example, it can estimate the duration of a one-leg hop. During the Ferienakademie 2015, a team of six students developed a sock with integrated textile pressure sensors. The goal of the sock was to replace the insole in order to improve user comfort. JASS 2016 was a winter school held in St. Petersburg (Russia) and had a duration of a week. During JASS 2016, a team of four students developed an additional interface for the KneeHapp Bandage on the Apple Watch. Furthermore, five students added support for further exercises and improved KneeHapp's algorithm accuracy as part of their Bachelor's and Master's thesis at the TUM. We are currently collaborating with Intel in order to port the KneeHapp Bandage to the Intel Curie microcontroller. The goals of the Curie-based version of KneeHapp are to reduce the size of the electronics, lower the price of the bandage and improve battery lifetime by performing more computations on the bandage and reducing the amount of data transferred from the bandage to the smartphone.

The development of the KneeHapp Bandage was done in close collaboration with orthopedic surgeons from the Sportsclinic Cologne, led by Prof. Jürgen Höher. We first visited the Sportsclinic Cologne at the beginning of the project with the goal to understand the application domain and elicit the core requirements for KneeHapp. Two other meetings with the Sportsclinic Cologne took place after we had developed a working version of KneeHapp. The goal of these subsequent meetings were to assess the bandage's usability, wearability and accuracy and to gather orthopedists' insights and ideas for further improvement. In this chapter we refer to the development and

in the next chapter to the evaluation of KneeHapp.

The *second* smart textile we present in this chapter is the Custodian Jacket, a jacket that monitors and supports technicians while performing maintenance activities in a supercomputer center. The Custodian Jacket was also developed in collaboration with real customers and end users. The customer of the Custodian Jacket was Siemens IT, a division of Siemens in charge of providing the digital infrastructure required for the company to function. One of the tasks of Siemens IT is the correct functioning of a supercomputer center built in the south of Munich that executes services used by the entire company. Some of these services are critical and could cause high losses to the company if not repaired on time. The goals of the Custodian Jacket were to improve technician's performance and safety during maintenance activities at the supercomputer center.

The Custodian Jacket was developed following an iterative development process in a period of 16 months. We held monthly meetings with Siemens where we reviewed the work done during the previous iteration and planned the goals for the next iteration. The first prototype of the Custodian Jacket was developed during the Ferienakademie 2013 by a team of 16 students in two weeks. The first prototype of the Custodian Jacket combined hardware from the .NET Gadgeteer and Arduino hardware kits. Every sensor and output device in the first prototype was wired to a central microcontroller provided by the .NET Gadgeteer kit sitting in the jacket's pocket. After the first prototype, we developed and evaluated three successive versions of the Custodian Jacket. Every new iteration had as a goal to make the Jacket more robust, comfortable, reduce its energy consumption and improve accuracy of the algorithms. The last version of the Custodian Jacket had two Arduino Lilypad microcontroller units that communicated wirelessly over Bluetooth Low Energy with a smartphone. This made it possible to eliminate the wired connections between sensors and a central microcontroller. Furthermore, hardware devices were connected with conductive thread to the microcontroller unit in order to reduce weight and risk to get the cables entangled.

In this chapter, we first describe how we developed each smart textile application without *TangoHapps* and then demonstrate how each application can be realized with *TangoHapps*.

8.1 Application 1: KneeHapp Bandage

Tear of Anterior Cruciate Ligament (ACL) is a common knee injury that occurs mostly among athletes [40]. In the US, the number of ACL injuries was estimated to be between 80,000 and 200,000 per year [35, 105]. The rehabilitation after an ACL injury can last as long as a year and often includes physical therapy, strength exercises and frequent visits to physiotherapists and doctors. Successful recovery of an ACL injury relies on the appropriate design of rehabilitation exercises performed by patients daily with the goal of recovering full range of motion, strength and coordination.

Currently, patients sustaining an ACL injury perform the rehabilitation exercises mostly unsupervised and lack quantitative ways to measure the quality and track

performance of their exercising. Orthopedists also lack tools to assess patients' rehabilitation progress and still have to rely on subjective observations such as how much a patient's leg shakes during a squat due to muscular instability. Furthermore, orthopedists and patients meet at time intervals as long as three months and the treatment is decided upon observations during these meetings without consideration of the patient's recovery progress in the periods between visits.

8.1.1 Problem

We interviewed two orthopedists who perform ACL surgeries on a daily basis and recorded and shadowed six patients after ACL-reconstruction surgery during their doctor visits. During these visits, patients had to perform a series of rehabilitation exercises. Based on our interviews and observations, we identified the problems described below.

Range of Motion (RoM)

After a knee surgery, the patient loses range of motion of the knee, mainly due to pain and swelling. Recovering a certain range of motion is required before patients can continue with the rehabilitation program. Patients can walk without crutches only after being able to fully extend their injured leg and can start pedaling a bicycle only after being able to bend their knees for at least 90 degrees. Therefore, it is important for patients to recover the flexibility on their injured leg as early as possible. The range of motion is measured every day at the beginning of the rehabilitation.

Orthopedists measure the range of motion of a patient's knee using a mechanical device called goniometer. Goniometers are placed at the knee's center of rotation and opened such that their arms are in line with the patient's upper and lower leg. The alignment of the goniometer to the leg is difficult due to muscle and fat in the leg. Goniometer measurements do not always correlate among different specialists [90]. Besides, the usage of a goniometer requires a second individual to take the measurement.

One-Leg Squat

After the ACL injury, patients begin to lose strength in their injured leg. Therefore, the second phase of the rehabilitation focuses on muscle building. One of the exercises orthopedists teach patients for this purpose is one-leg squat and different variations of it. During a one-leg squat, patients stand on the injured leg, bend it as much as they can and then go back up into their initial position without deviating their knees from the line between the ankles and hips. The more patients bend their leg during a one-leg squat, the more difficult the exercise becomes. Therefore, it is desirable to know how much patients bend their leg during the squat. However, orthopedists lack of convenient ways to measure the degree of flexion and count only the amount of repetitions while observing the quality of the movement.

During a squat, many patients are not able to stabilize the leg and deviate from the optimum axis and rotate their knees inwards (called medial collapse). Bending the knee inwards indicates a weakness of the hip and leg due to pain or lack of strength and might result in new knee injuries. Despite the importance of preventing medial collapses, orthopedists and orthopedists lack ways to quantify the degree of medial collapse during a squat. To the best of our knowledge, there are currently no solutions for quantifying the degree of medial collapse during a squat. The shaking of the leg during a squat is another important parameter indicating muscular instability. This is observed subjectively by orthopedists.

One-Leg Hop, Side Hops and Running in Eights

At the end of the rehabilitation phase, orthopedists should assess whether patients are ready to start doing sports again. In order to do this, orthopedists observe the patient's injured leg's performance on different exercises and compare it to their performance on the patient's healthy leg. While orthopedist's have not reached a common agreement on the exercises for the assessment, commonly accepted exercises are one-leg hops, side hops and running in eights.

One-leg hops is an exercise in which patients should jump forward as far as possible using only one leg and land stably. Orthopedists measure the distance of the hop by placing a meter on the ground next to the area where the patient jumps.

During *side-hops*, patients are asked to hop side-wise over a distance of 20 cm using only one leg as many times as possible during a period of 15 to 60 seconds of time. In order for patients to concentrate on the exercise, orthopedists count the hops. Orthopedists say that counting the hops is cumbersome and that they sometimes lose count.

Running in eights is another exercise used by orthopedists to assess patient's rehabilitation progress. Performance of the *running in eights* exercise is determined by the time it takes patients to complete the figure of eight. In the current praxis, orthopedists use a timer to measure how long it takes patients to complete the exercise.

8.1.2 KneeHapp Bandage

KneeHapp is a compression bandage that patients wear after the knee surgery, with two Inertial Measurement Units (IMUs) inserted into pockets at the upper and lower leg. The bandage is shown in Figure 8.1 a). Other research such as [4, 2, 129] and commercial products such as CoreHab¹ use leg bands which are fastened to the upper and lower part of the leg. Being a single piece, the sensors in the bandage are always placed in the same relative distance to each other avoiding the risk of inconsistent measurements caused by a misplacement of the sensors. This cannot be achieved using the leg bands because they consist of two pieces that are placed independently on the upper and lower leg. Leg bands should be fastened tight so that they don't slide while patients perform several repetitions of a rehabilitation exercise. Fastening

¹<http://www.corehab.it/en/>

8.1. APPLICATION 1: KNEEHAPP BANDAGE

a leg band tight might cause discomfort, especially as muscles need to tense and relax several times during an exercise session. Because the compression bandage has a bigger contact surface to the patient's leg, it is unlikely to slide during exercising. KneeHapp measures the quality of the different rehabilitation exercises described in previous section, gives feedback to patients and shares the measurements with orthopedists.



Figure 8.1: a) KneeHapp Bandage. b) KneeHapp sock.

Use Case 1: Range of Motion (RoM)

KneeHapp calculates the angle of flexion of the leg by computing the difference between the Yaw values delivered by upper and lower IMUs. KneeHapp's coordinate system is shown in Figure 8.2. By convention, the angle of flexion should be equal to zero when the leg is relaxed on a flat surface. Because bones are not perfect straight lines and because of additional muscle and swelling, a direct angle computation does not usually yield zero degrees even when the leg is relaxed on a flat surface. Therefore, a calibration is performed to determine the alignment of the IMUs to the leg. KneeHapp supports two calibration techniques. The first one determines the sensor alignment while patients extend their leg on a flat surface. However, because most patients are not able to fully extend their leg after the surgery, we designed the following alternative calibration approach that uses the healthy leg to estimate the offset angle:

1. Wear the bandage on the healthy leg and measure the orientation of the IMU on a flat surface.
2. Place leg above any object and measure the angle of flexion.
3. Wear the bandage on the injured leg, place the leg above the same object and measure the angle of flexion.

Because the angle measurements when using the object should be equal on both legs, after these calibration steps KneeHapp knows the sensor alignment on the injured leg. Other approaches we considered were pose calibrations, a static calibration using a wedge and functional calibration approaches. Pose calibration approaches compare the orientation measured by motion sensors to expected measurements while the subject performs a specific pose [85]. However, simple pose-based calibration is

not suitable for rehabilitation right after the surgery because of the difference in the range of motion of both knees. Ayoade et al. use a static calibration technique that requires patients to place their knee on a wedge with a predefined degree of flexion [4]. The accuracy of this approach will depend on the length of each individual’s leg and requires a wedge that patients might not have in their homes. Functional calibration approaches require the individual to perform a series of movements used to infer the alignment of the motion sensors to the body [33, 32]. These movements should be performed by a specialist, which makes them unsuitable to home-based applications with in- expert users [85]. Our calibration approach is suitable for the home and does not require additional equipment such as cameras or other individuals to take the measurements.

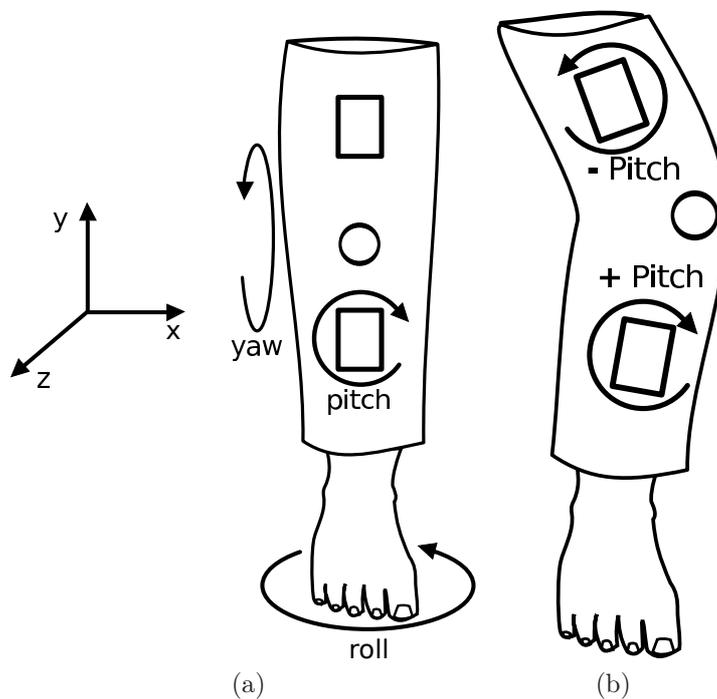


Figure 8.2: a) KneeHapp coordinate system. b) Illustration of the medial collapse during a squat. and how it affects the pitch.

Use Case 2: One-Leg Squat

Using the orientation vectors provided by both accelerometers, KneeHapp tracks the angle of flexion of the leg during the squat. The technique used to calculate the angle of flexion during a squat is the same as the one used to calculate the angle of flexion in *Use Case 1: Range of Motion*. However, the calibration for a one-leg squat is done while the patient is standing straight. The iPad application triggers a visual and auditive feedback when the minimal angle of the squat has been achieved.

During the squat, medial collapse causes upper and lower leg to bend in opposite directions. In our coordinate system, the pitch rotation component indicates how much the IMUs have rotated due to medial collapse. Figure 8.2 b) shows that the

upper and lower IMUs rotate in opposite directions, which is reflected in the Pitch rotation component delivered by the IMUs. Therefore, in order to determine the degree of medial collapse during a squat, we add up the difference of the Pitch rotation angles of upper and lower IMUs to the difference of the Pitch stored during the calibration. We negate the upper sensor's pitch because a smaller value indicates a higher medial collapse. The iPad application provides auditive feedback when the amount of medial collapse is bigger than four degrees in order for users to correct their deviation.

KneeHapp also determines the degree of shaking of the leg. In order to measure the degree of shaking of the leg, we compute the standard deviation of the linear acceleration produced by the upper IMU along the x-axis. We chose to use only the upper IMU to measure the shaking because squats challenge mostly the muscles on the upper and cause shaking due to muscular instability.

Use Case 3: One-Leg Hop

In order to quantify patient's performance during one-leg hops, orthopedists measure the distance of their hops on the injured leg and compare it to the same measurement on their healthy leg. According to the two orthopedists we interviewed, the duration of the hop can be an equivalent measurement of a patient's performance during this exercise. Therefore we chose the calculation of the duration of one-leg hops as performance parameter in this exercise.

To measure the duration of one-leg hops, we studied two possible solutions based on different types of sensors. The first one uses KneeHapp's accelerometers and the second uses a sock with integrated textile pressure sensors that can be attached to the bandage. The highest peak in the signal represents the moment where the patient is at the highest height during a hop. The lowest negative peak corresponds to moment where the patient landed. Figure 8.3 shows the linear acceleration along the y and z-axes of the upper leg IMU recorded during a one-leg hop with annotations for each phase of the hop. An average one-leg hop has a duration of 0.23 seconds. A sampling rate of 100 Hz enabled us to detect the different phases of the one-leg hops. KneeHapp uses a peak detection algorithm to detect highest and lowest peaks and then measures the difference between both peaks. Since the highest peak in the signal does not correspond to the moment when the patient jumped off the ground, the computed difference is scaled by a constant factor.

The second approach uses textile pressure sensors attached to the sole of a sock. The smart sock is shown in Figure 8.1 and the raw pressure sensor signal recorded by the sock during a one-leg hop is shown in Figure 8.4. The signal produced by the textile pressure sock contains two peaks that correspond to the take-off and landing phases of the hop, when higher pressure is applied to the sole. The duration of the hop is also estimated by measuring the distance between both peaks. However, in this case, the peaks indicate the exact time when the foot left and came back in contact to the ground after a hop.

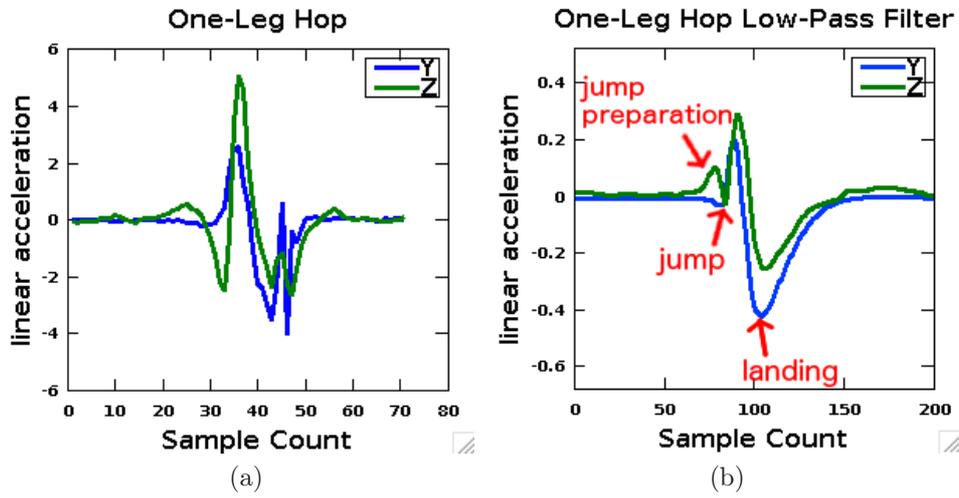


Figure 8.3: One-leg hop linear acceleration on y and z-axis raw (a) and filtered (b).

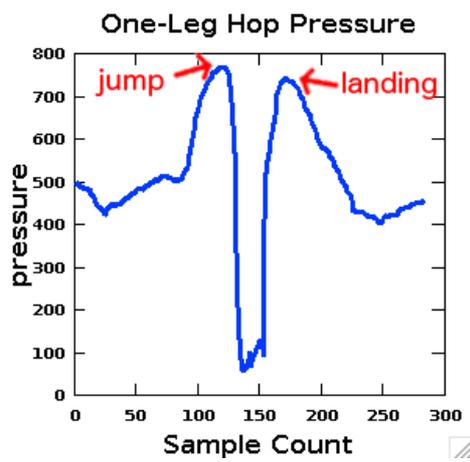


Figure 8.4: One-leg hop average pressure.

Use Case 4: Side Hops

For the side hops we also use an approach based on the morphology of the signal produced by accelerometers. Figure 8.5 a) shows the linear acceleration on the y-axis for seven side hops. In order to eliminate noise in the signal, we apply several iterations of a low-pass filter until the signal's standard deviation becomes smaller than a specific threshold. We call this threshold low-pass filter threshold. We found the optimal low-pass filter threshold to vary depending on the algorithm used for counting the number of hops. We tested the RC low-pass filter with a time constant $c = 0.25$ and a weighted moving average filter with coefficients $[1/4, 1/2, 1/4]$. After filtering the signal, we use a peak detection algorithm to count the high and low peaks in the signal. A naive peak detection algorithm counts also local maxima as hops in the signal, resulting in more side hops being counted. To overcome this issue, we analyze the surroundings of a peak. If the peak is larger enough than its surroundings, it is counted as a hop. We call this factor peak threshold. We found that the peak threshold leading to most accurate results depends on the filtering algorithm used. Figure 8.5 b) shows the same hops after applying three iterations of the RC filter and running the peak detection algorithm.

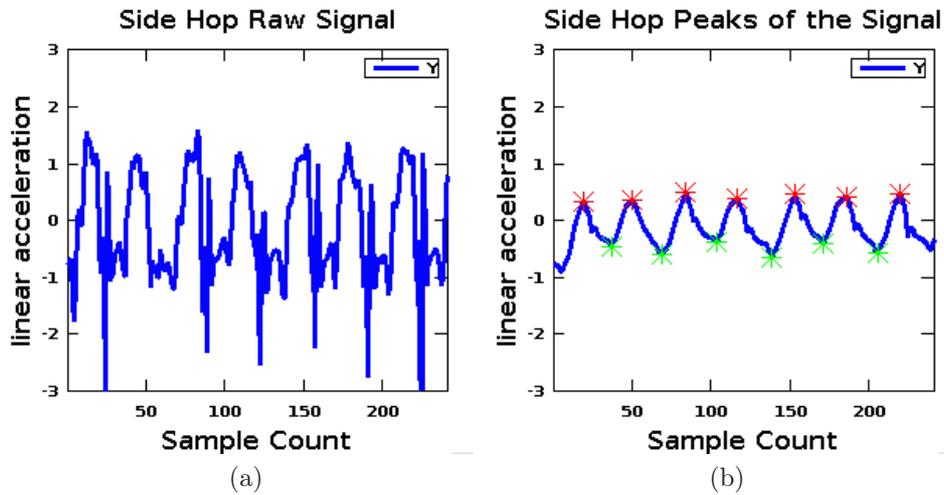


Figure 8.5: Linear acceleration of seven side hops. raw (a) and filtered (b).

Use Case 5: Running in Eights

KneeHapp determines whether patients are standing or running by comparing the standard deviation of the magnitude of the linear acceleration vector of the upper IMU with a specific threshold. We use windows of 50 samples with a sampling rate of 100 Hz and an overlapping factor of 50%. This solution removes the requirement for additional devices and humans who measure the time. Furthermore, this use case removes external factors that affect the measured performance and makes the measured performance dependent solely on patient actions.

8.1.3 Implementation with TangoHapps

In this section we describe how the KneeHapp Bandage can be implemented with *TangoHapps*. The KneeHapp Bandage requires the usage of an *MPU 6050* IMU on the upper leg and another one on the lower leg. The design of the KneeHapp Bandage in *TangoHapps* uses the *BLE-Lilypad* microcontroller instead of the Arduino Mini used in the version of KneeHapp implemented without *TangoHapps*. The *BLE-Lilypad* fuses accelerometer and gyroscope signals and sends rotation data over Bluetooth Low Energy to the smartphone. The *BLE-Lilypad* is equivalent to the Arduino Mini in terms of binary code it is able to execute. Figure 8.6 displays the design of the circuit created with *TangoHapps*. This circuit is inserted in the lower pocket of the KneeHapp Bandage. Another identical circuit is inserted in the upper leg’s pocket.

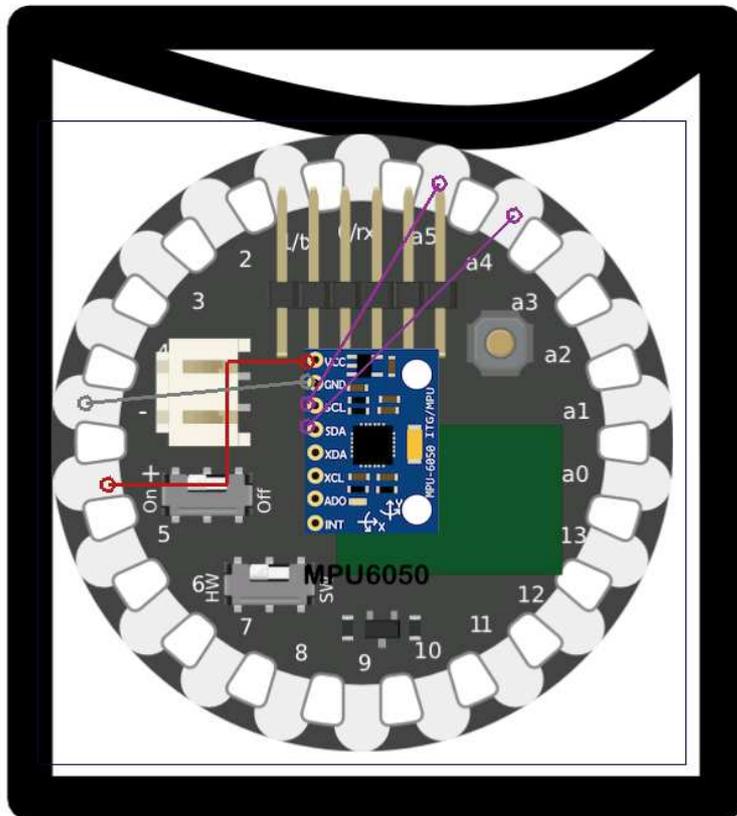


Figure 8.6: KneeHapp’s circuit design created with *TangoHapps*.

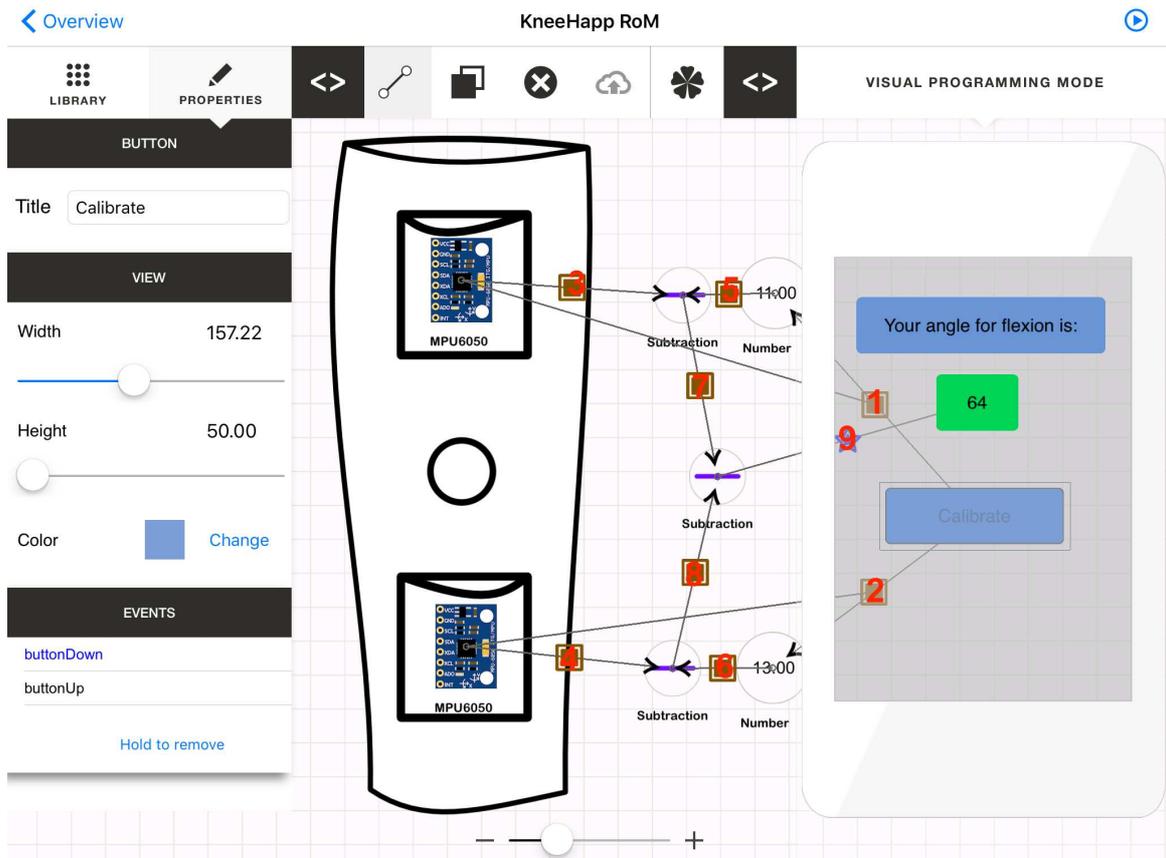


Figure 8.7: KneeHapp’s *Range of Motion* use case implemented with *TangoHapps*.

Use Case 1: Range of Motion

A *TangoHapp*’s application that computes the range of motion would require the following *Happ Objects*:

- Three *Subtraction Arithmetic Operators*. We name the *Subtraction* operators: “*Upper Sensor Subtraction*”, “*Lower Sensor Subtraction*” and “*Sensor Subtraction*”.
- Two *Number* variables. We name the *Number Variables*: “*Upper Sensor Number*” and “*Lower Sensor Number*”.
- A *Button UI Widget*. We name the *Button* “*Calibrate Button*”.
- A *Label UI Widget*.

Furthermore, the application should define the following *Invocations* between objects:

1. The *Calibrate Button*’s *buttonPressed()* *Event* triggers the *Upper Sensor Number Variable*’s *setValue()* *Method* passing the upper *MPU 6050*’s *Yaw Variable* as a parameter.

2. The *Calibrate Button*'s *buttonPressed()* *Event* triggers the *Lower Sensor Number Variable*'s *setValue()* *Method* passing the lower *MPU 6050*'s *Yaw Variable* as a parameter.
3. The upper *MPU 6050 yawChanged()* *Event* triggers the *Upper Sensor Subtraction*'s *setOperand1()* *Method* passing the measured yaw as a parameter.
4. The lower *MPU 6050 yawChanged()* *Event* triggers the *Lower Sensor Subtraction*'s *setOperand1()* *Method* passing the measured yaw as a parameter.
5. The *Upper Sensor Number*'s *valueChanged()* *Event* triggers the *Upper Sensor Subtraction*'s *setOperand2()* *Method* passing the value stored in the variable of the subtraction as a parameter.
6. The *Lower Sensor Number*'s *valueChanged()* *Event* triggers the *Lower Sensor Subtraction*'s *setOperand2()* *Method* passing the value stored in the variable of the subtraction as a parameter.
7. The *Upper Sensor Subtraction*'s *computed()* *Event* triggers the *Sensor Subtraction*'s *setOperand1()* *Method* passing the result of the subtraction as a parameter.
8. The *Lower Sensor Subtraction*'s *computed()* *Event* triggers the *Sensor Subtraction*'s *setOperand2()* *Method* passing the result of the subtraction as a parameter.
9. The *Sensor Subtraction*'s *computed()* *Event* triggers the *Label*'s *setText()* *Method* passing the result of the subtraction as a parameter.

In order to do the sensor calibration, patients should extend their leg on a flat surface and press the *Calibrate Button*. When the user presses the *Calibrate Button*, the application stores the current Yaw value of upper and lower IMUs into the *Upper Sensor Number* and *Lower Sensor Number* variables. After the initial calibration phase, the values stored in both variables are used as a reference to compute how much the IMUs have offset from their initial position. Therefore, the current Yaw values of upper and lower IMUs are subtracted from the values stored in the variables. Finally, in order to measure the angle of flexion of the leg, the application computes the difference of upper and lower yaw offsets and displays it through a *Label*. A sample application that computes the range of motion of a patient's knee is displayed in Figure 8.7.

Use Case 2: One-Leg Squat

An application that computes the angle of flexion and medial collapse of the leg during a squat can be created in *TangoHapps* with the same *Happ Objects* and *Invocations* used in *Use Case 1: Range of Motion* to compute an angle using two IMUs. However, in order to compute the degree of the medial collapse, the *MPU 6050*'s Pitch rotation component should be used. Furthermore, the application should indicate to users when the desired angle of flexion during a squat has been reached. The functionality

8.1. APPLICATION 1: KNEEHAPP BANDAGE

to determine whether an angle is bigger than a threshold can be realized with the *Bigger Comparison Operator*. The *Bigger Comparison Operator's* *setOperand1()* and *setOperand2()* *Methods* should be invoked with the angle and with a *Variable* containing the threshold. Next, the *Bigger Comparison Operator's* *conditionIsTrue()* event should invoke the *Sound's* *play()* *Method*. This functionality is used to inform the user about a specific squat angle being reached and about a medial collapse bigger than four degrees. A simple modification to the above function to use the value provided by a *Slider UI Widget* in the comparison, would enable users to configure the angle for receiving sound notifications at runtime. A screenshot of an *Interactex Designer* application that implements this functionality is shown in Figure 8.8.

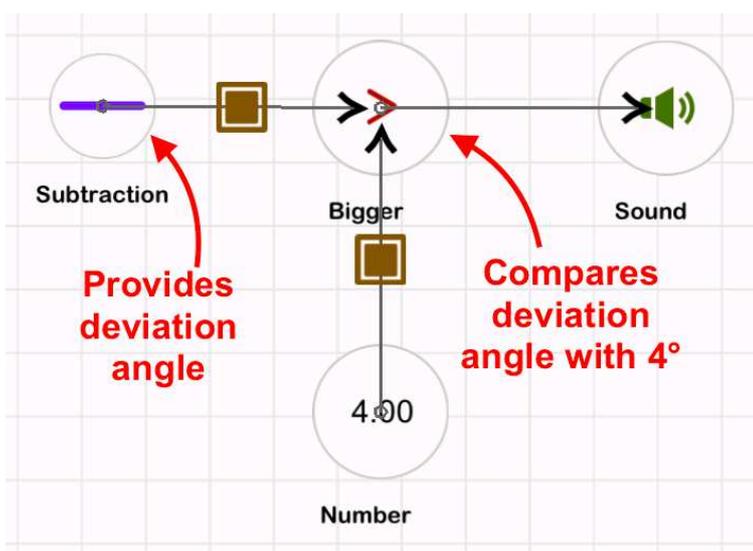


Figure 8.8: KneeHapp’s functionality to produce auditive feedback when the user’s knee deviates more than 4 degrees during a *One-Leg Squat*.

The application should also display live feedback about the shaking of his/her leg. This can be achieved with the *Monitor UI Widget*. The upper *MPU-6050's* *xChanged()* *Event* should trigger the *Monitor's* *addValue1()* object. This causes every linear acceleration value along the x-axis to be added to a line chart rendered by the *Monitor UI Widget*.

Additionally, the degree of shaking of the patient’s leg can be computed using the *Deviation Extractor Function* and displayed on a *Label*. This functionality can be realized with a *Window Function*, a *Deviation Extractor Function*, a *Label UI Widget* and the following *Invocations*:

1. The upper *MPU 6050's* *xChanged()* *Event* triggers the *Window's* *addValue()* *Method* passing in the linear acceleration along the x-axis.
2. The *Window's* *filled()* *Event* triggers the *Deviation Extractor's* *addValue()* *Method* passing in the buffered acceleration values.
3. The *Deviation Extractor's* *featureExtracted()* *Event* triggers the *Label's* *setText()* *Method* passing in the computed deviation.

CHAPTER 8. APPLICATIONS

A sample application that supports the one-leg squat exercise is displayed in Figure 8.9.

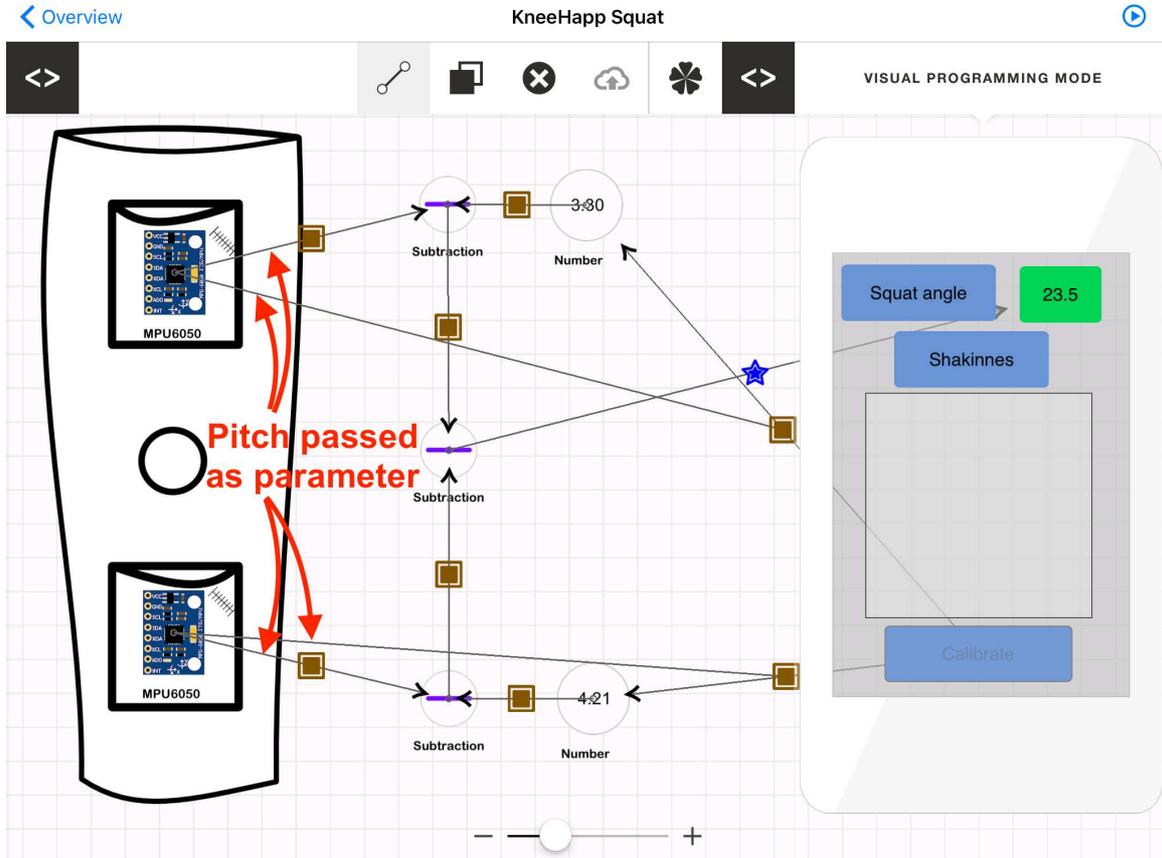


Figure 8.9: KneeHapp's *One-Leg Squat* use case implemented with *TangoHapps*.

Use Case 3: One-Leg Hop

The algorithm to compute hop durations can be implemented in *TangoHapps* using the following *Happ Objects*:

1. A *Window*.
2. Two *Peak Detectors*. We name them *Lower* and *Upper Peak Detector*.
3. A *Subtraction Arithmetic Operator*.
4. A *Multiplication Arithmetic Operator*.
5. A *Variable*.
6. Two *Buttons*. We name them “*Start*” and “*Stop*” Buttons.

Furthermore, the following *Invocations* should be defined:

1. The *Start Button*’s *buttonDown()* *Event* triggers the upper *MPU 6050*’s *start()* *Method*.
2. The *Stop Button*’s *buttonDown()* *Event* triggers the *MPU 6050*’s *stop()* *Method*.
3. The *MPU 6050*’s *yChanged()* *Event* triggers the *Window*’s *addValue()* *Method* passing in the linear acceleration along the y-axis.
4. The *Stop Button*’s *buttonDown()* *Event* triggers the *Upper PeakDetector*’s *compute()* *Method* passing in the array of linear acceleration values stored in the *data* property of the *Window*.
5. The *Upper PeakDetector*’s *indexExtracted()* *Event* triggers the *Lower PeakDetector*’s *setRangeStart()* *Method*.
6. The *Stop Button* should also set the values of the *Lower Peak Detector* in a similar way as in step 4. For this purpose, the *Stop Button*’s *buttonDown()* *Event* triggers the *Lower PeakDetector*’s *compute()* *Method* passing in the array of linear acceleration values stored in the *data* property of the *Window*.
7. The *Upper PeakDetector*’s *indexExtracted()* *Event* triggers the *Subtraction*’s *setOperand1()* *Method* passing in the index of the peak detected.
8. The *Lower PeakDetector*’s *indexExtracted()* *Event* triggers the *Subtraction*’s *setOperand2()* *Method* passing in the index of the peak detected.
9. The *Subtraction*’s *computed* *Event* triggers the *Multiplication*’s *setOperand1()* *Method* passing in the result of the subtraction.
10. The *Number*’s *valueChanged()* *Event* triggers the *Multiplication*’s *setOperand2()* *Method* passing in the fixed value of 0.0136. The constant 0.0136 is used to scale the measured jump duration and convert the result of the subtraction from samples to seconds.

11. The *Multiplication Arithmetic Operator's computed() Event* triggers the *Label's setText() Method* passing in the result of the multiplication.

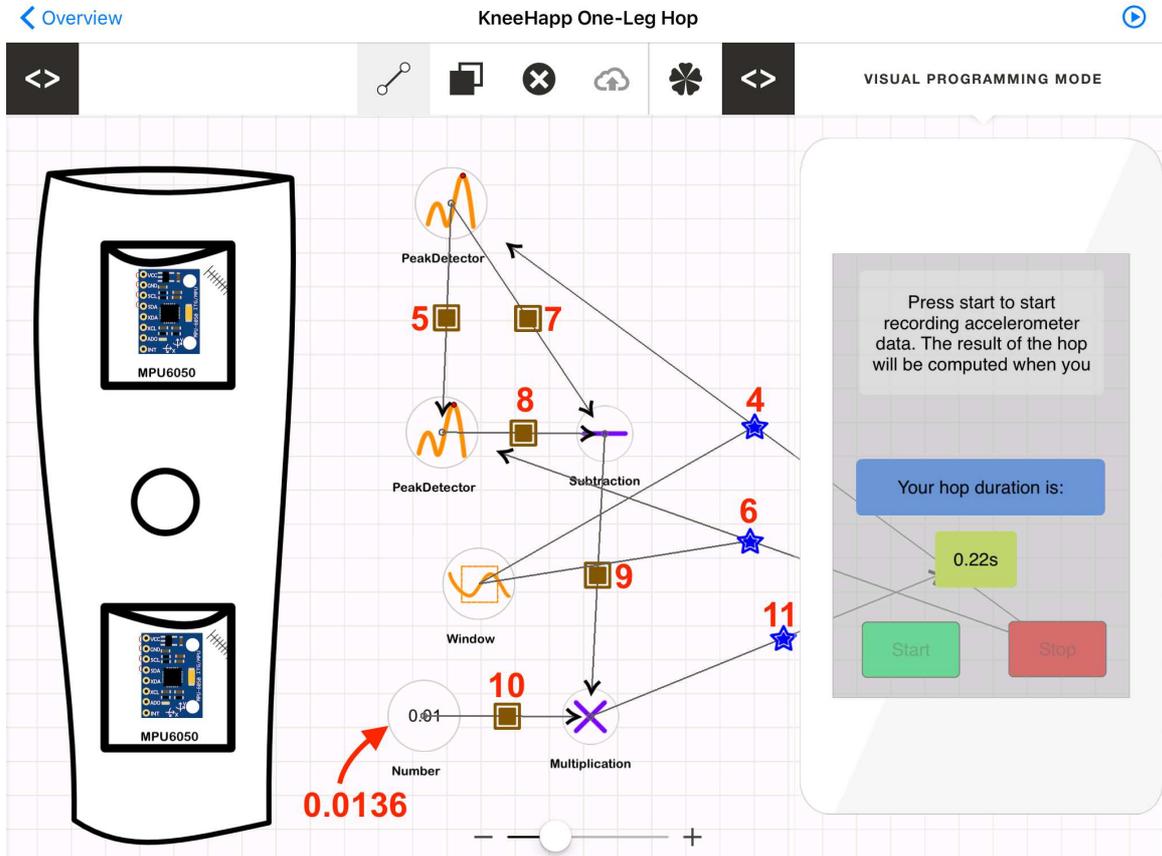


Figure 8.10: KneeHapp’s *One-Leg Hop* use case implemented with *TangoHapps*.

The *Invocations* described above are necessary to create an application that estimates the duration of a hop as described in Section 8.1.2. A screenshot of an application containing every object and *Invocation* mentioned so far is displayed in Figure 8.10. It should be noted that the *Invocations* between the *Start* and *Stop Buttons* and the upper *MPU 6050* are not displayed in Figure 8.10 for clarity purposes. The *Start* and *Stop Buttons* are controlled by users and cause the upper *MPU 6050* to start and stop recording values. The *Window's size Variable* is set to a very large number so that the *Window* has enough buffer space to store the accelerometer samples produced during the hop. The *Stop Button* first causes the *Window* to stop storing values and then triggers the execution of the algorithm that uses the data in the *Window* to estimate the duration of the hop. The *Upper Peak Detector* is configured to detect high peaks and the *Lower Peak Detector* is configured to detect low peaks over their *upper peak Variables*. Since the data passed to both *Peak Detectors* is the accelerometer signal of an entire hop, the *Upper Peak Detector* will detect the highest and the second *Lower Peak Detector* the lowest peak in the signal. An additional optimization is done by setting the *Lower Peak's* starting range index to the *Upper Peak's* index. This reduces the number of computations in the algorithm

by limiting the search range of the lowest peak. The *Subtraction Operator* is used to measure the index difference between the peaks detected. Because accelerometer The time difference between The final hop duration in seconds is computed by scaling the difference indexes. For this purpose, the result of the subtraction is multiplied by the constant 1.36 by using the *Multiplication Operator*.

Use Case 4: Side Hops

An algorithm to count the number of side hops based on an accelerometer signal can be implemented with a *TextIT* plugin. The plugin identifies and returns the number of peaks in an array of input values. The algorithm iterates over every sample in the input signal. Once an upper peak is detected, the algorithm starts looking for lower peaks. Once a lower peak is detected, the algorithm starts looking for upper peaks again. This process repeats. A counter is increased every time an upper peak is detected. The function receives an additional parameter called “*delta*”. The *delta* parameter is a threshold for peaks to be counted. This threshold is necessary to avoid local maxima in the signal. The pseudo-code contained in the *TextIT* plugin is shown in Section A.5.3 in the Appendix.

An *Interactex* application that uses the *TextIT* plugin to count peaks requires additionally the usage of the *Low-Pass Filter*, and a *Number Variable* where the *delta* parameter needed by the *TextIT* plugin is stored. The *Low-Pass Filter* receives the signal directly from the upper IMU and filters it. The filtered signal is passed over to the *TextIT* plugin. The *Low-Pass Filter*’s filtering *factor* and *delta* parameter are configurable by developers. We configure the *Low-Pass Filter*’s *factor Variable* to 0.5 and *delta* parameter to 60. The UI is controlled in a similar way as the *One-Leg Hop* application described earlier in this chapter. The *Start* and *Stop* buttons cause the upper IMU to start and stop receiving data. The result returned by the *TextIT* plugin is displayed in a label in the smartphone. A screenshot of an *Interactex* application that computes the side hops is shown in Figure 8.11. The *TextIT* plugin has been renamed to “sideHops”.

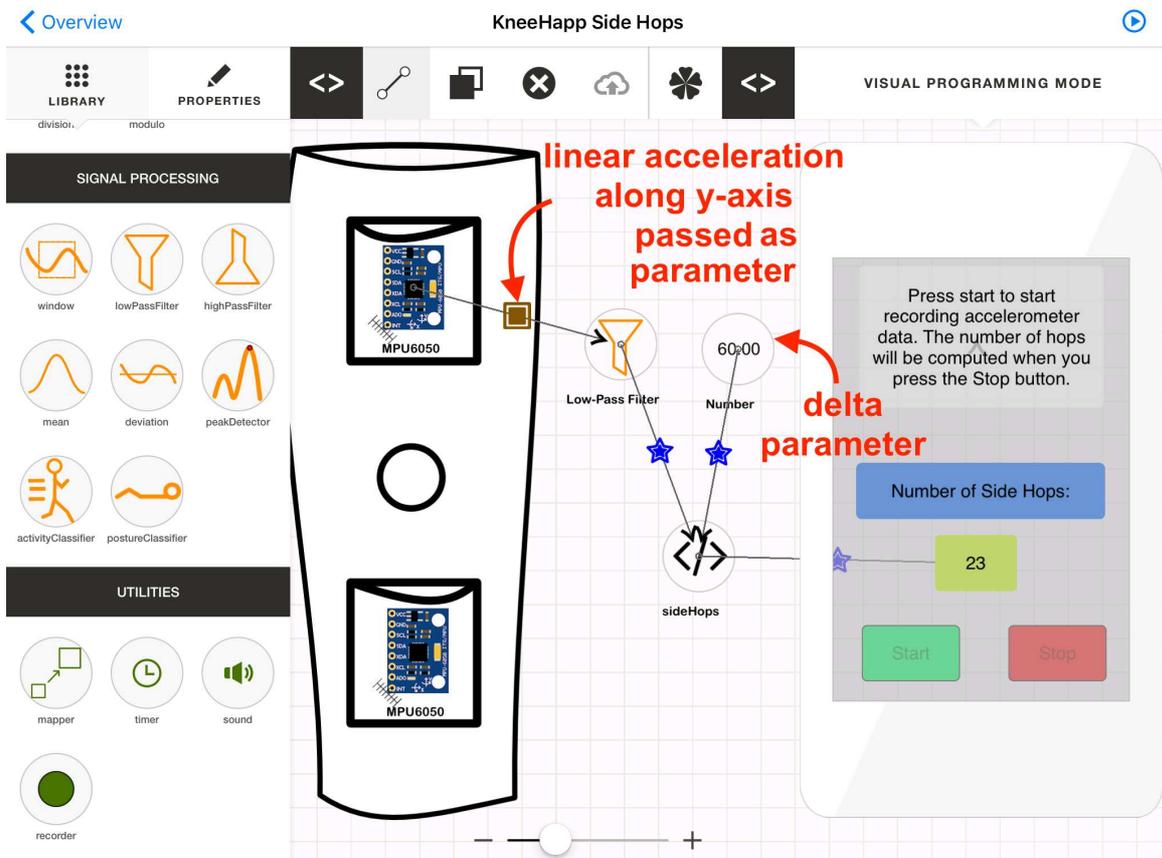


Figure 8.11: KneeHapp’s *Side Hops* use case implemented with *TangoHapps*.

Use Case 5: Running in Eights

An application that starts and stops a timer when it detects the user has started / stopped running can be implemented with *TangoHapps* using the *Activity Classifier* and *Timer Happ Objects*. The *Activity Classifier* requires an array of filtered linear acceleration data. The application should define the following *Invocations* in order to classify user activity based on linear acceleration:

1. The *MPU 6050's yChanged() Event* invokes the *Window's addValue() Method* passing in the linear acceleration along the y-axis as a parameter.
2. The *Window's filled() Event* invokes the *Low-Pass Filter's addValues() Method* passing in the array of buffered linear acceleration samples.
3. The *Low-Pass Filter's filtered() Event* invokes the *Activity Classifier's classifySamples() Method* passing in the filtered array of linear acceleration samples.

We set the *Window's size Variable* to 50 and the *overlapping Variable* to 0.5. The *Activity Classifier* will emit the *running() Event* after the patient started running and the *notMoving() Event* after the patient stopped running. In order to measure the duration of the run, these *Events* are used to start and stop a *Timer*. We call this *Timer* “*Running Timer*”. The *Timer* in *TangoHapps* emits an *Event* after the time specified by developers has elapsed. In this application, the timer should be stopped when the user stopped running. This can be achieved by setting the *Timer's time Variable* to a high value (to ensure it will not finish before the patient completed the run) and by defining two *Invocations*. First, the *Activity Classifier's running() Event* invokes the *Timer's start() Method*. Second, the *Activity Classifier's notMoving() Event* invokes the *Timer's stop() Method*.

Finally, the *Running Timer's* elapsed time can be displayed on a *Label* on the smartphone. In order to continuously update the *Label* as the time passes, a second *Timer* that will trigger the *Label* updates should be used. We call the second *Timer* “*Update Timer*”. The *Update Timer* is started and stopped by the same *Events* as the *Running Timer*. A *Timer* in *TangoHapps* can be configured to trigger periodically by setting its *repeat Variable* to *true*. We set the *Update Timer's repeat Variable* to *true* and *frequency Variable* to 0.05 seconds so that the *Label* is updated every 0.05 seconds. Next, we define an *Invocation* between the *Update Timer's triggered() Event* and the *Label's setText() Method* passing in the *Running Timer's time elapsed* variable as a parameter.

8.2 Application 2: Custodian Jacket

Siemens is a multinational company with more than 340.000 employees worldwide. In order to store employees' data and provide services needed for the functioning of the company, Siemens built a supercomputer center in the south of Munich. The supercomputer center contains 154 racks that store over 1230 servers. Some of the data and services stored in the supercomputer center are highly critical to the organization. Siemens might suffer major losses if a service became unavailable for a few hours or if employees data was lost. Furthermore, some maintenance operations done at the supercomputer center might be dangerous, such as dealing with high voltages. Therefore, the main goals of this project were to prevent costly human mistakes and to increase technician's safety while performing maintenance activities in the supercomputer center.

8.2.1 Problem

In order to gain understanding into the application domain, we conducted several interviews with security managers at Siemens during a period of 14 months of time. We identified three main problems, which we address as “use cases” in the next sections.

Error prevention

Racks contain up to 46 “height units”. A height unit is a slot where a server can be inserted. A server might occupy one or several height units. A height unit has a standard height of approximately 4.5 cm. Racks have a label attached next to each height unit with a numeric identifier. In order to identify a server, technicians find in their worksheets the height unit where the server is installed. Technicians operate servers from the front and backside. Labels might not be visible on the backside of the rack due to the presence of cables, as shown in Figure 8.12 b). Technicians have pulled a working server by mistake in the past, causing a large amount of losses to the organization.

In order to help technicians identify servers that need to be replaced or repaired, Siemens considered installing an LED for each height unit in a rack. The LED would be remotely controlled and be turned on to indicate a faulty server. However, this approach would require the installation and maintenance of more than 6000 LEDs. Instead, Siemens is more interested in a solution with less maintenance costs.

Technician's Safety

In order to repair the faulty servers as fast as possible, a pool of technicians is ready to access the supercomputer room at any day and any time of the week (even at 3 am on a Sunday). At specific times of the day, technicians are likely to have to enter the supercomputer center alone. In order to perform a maintenance operation, technicians perform hazardous operations such as dealing with high voltages, lifting

heavy equipment, climbing ladders and crunching into tight spaces. The closest person that can assist technicians in case of an accident is the gatekeeper, who sits in a different building. Siemens is interested in a solution to monitor and assist technicians in case of accidents.

Emergency Situation

Siemens installed speakers inside the supercomputer center in order to notify technicians about emergency situations such as the presence of fire in the building. Each rack in the supercomputer center is equipped with fans on both the front and back side in order to keep a constant air flow that cools the servers' temperature down. These fans produce a considerable amount of background noise. Therefore, the volume of the speakers had to be loud enough so that it would be heard from different positions within the supercomputer center. Siemens had to remove the speakers after realizing that the vibrations produced by the speakers damaged the nearby hard drives.

8.2.2 Custodian Jacket

The Custodian Jacket is a smart jacket worn by technicians during maintenance activities in a supercomputer center. The main purpose of the Custodian Jacket is to reduce the chances of human mistakes, which might cause high losses to the company and to increase technician's safety while operating in the supercomputer center. We have developed four versions of the Custodian Jacket. The first version was developed in a 2-week summer school by a group of 16 students and was based on the .NET Gadgeteer hardware family. The other three versions of the Custodian Jacket were developed in collaboration between the TUM and Siemens during a one-year project involving three doctoral students and five managers at Siemens. These later versions were based on the Arduino microcontroller family. An image of the last version of the Custodian Jacket is shown in Figure 8.12 a).

In this chapter, we focus on the last version of the Custodian Jacket (version 4). The last version of the Custodian Jacket has two 6-axis Accelerometer and Gyroscope IMU on the chest and sleeve, a proximity sensor on the sleeve and capacitive textile sensors on the shoulders and chest. The textile sensor on the shoulders are used to detect whether a technician is wearing the jacket. The textile sensor on the chest is used as a button. Furthermore, the jacket provides visual, haptic and auditive output to users with an LED and vibration motor on the wrist and a buzzer at the chest. Hardware devices are connected with conductive thread to custom made BLE-Lilypad attached to the chest. The BLE-Lilypad communicates over BLE with a smartphone. In the rest of the section, we describe three use cases that address the three problems we identified in the previous section.

Use Case 1: Error prevention

The Custodian Jacket has an ultrasonic proximity sensor attached to the sleeve. This sensor measures the distance between the user's sleeve and the ground. By comparing



Figure 8.12: a) Custodian Jacket - fourth version. b) Backside of a rack in a super-computer center. Some height unit labels are hidden by cables.

the distance to the ground, the jacket is able to know what server the user's arm is in front of. Technicians swipe their arms from top to bottom of a rack passing through every server. The jacket produces a light outputs to indicate whether the current server should be repaired or not.

The ultrasonic proximity sensor emits an ultrasonic beam and measures the time it takes the beam to return to the sensor. In order to measure the distance to the ground accurately, the sensor should be placed straight such that the beam forms a 90 degree angle with ground. The Custodian Jacket uses a 6-axis accelerometer and gyroscope in order to detect whether the proximity sensor points straight to the ground. An additional LED on the sleeve gives technicians feedback about the proper orientation of the proximity sensor. A green LED indicates that the proximity readings are valid and that the technician's hand is in front of the server that needs to be repaired. A blue LED indicates the proximity readings are valid the technician's hand is not in front of a server that needs to be manipulated. A red LED indicates that the sensor is not pointing straight down, hence proximity readings are not valid. Figure 8.12 displays a technician swiping through servers on a rack to locate a specific server.

Use Case 2: Technician’s Safety

The Custodian Jacket monitors technician’s physical activity in order to determine whether a technician might need assistance. The IMU on the torso is used to determine whether he/she is walking, standing, running or lying down on the ground. If the technician is found to be lying down on the ground for more than 20 seconds, the jacket sends an emergency signal to the gatekeeper. In order to avoid sending emergency signals when the user has taken off the jacket, the application determines whether the user is wearing the jacket with the capacitive sensor on the shoulder.

The activity recognition is done based on the linear acceleration, which is delivered by the IMU attached to the Custodian Jacket’s torso. The linear acceleration is sampled with a frequency of 100 Hz. Then, the linear acceleration along the y-axis is buffered in windows of 50 samples with an overlapping factor of 50%. We apply a low-pass filter and extract the signal deviation. We determine whether the user is standing, walking, or running based on the signal deviation. The linear acceleration along all the y-axis recorded while the user was standing a) and walking b) are shown in Figure 8.13. The distribution of the different activities in a 3D space is shown in Figure 8.14.

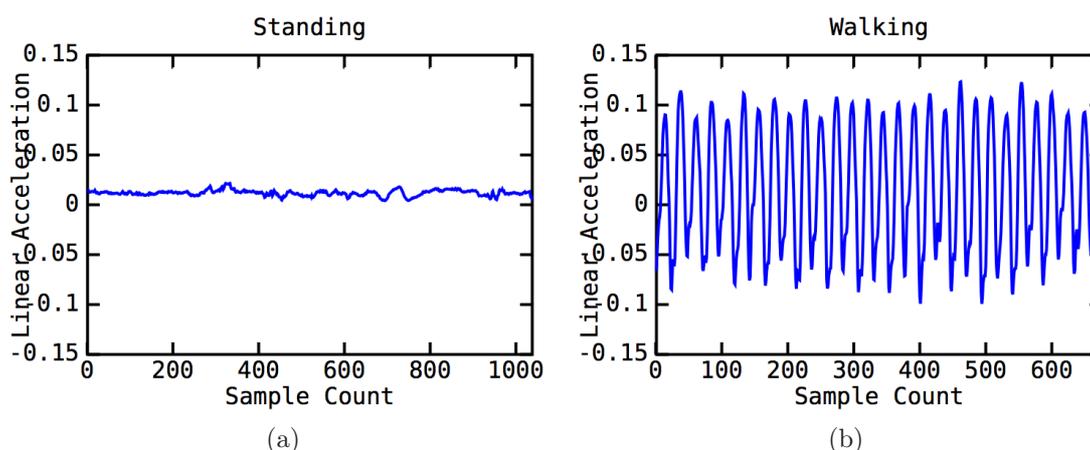


Figure 8.13: Linear acceleration along y-axis while user is standing (a) and walking (b).

Use Case 3: Emergency Situation

The Custodian Jacket is equipped with a speaker and a vibration motor in order to propagate emergency events to the technician. Emergency events are sent out by the gatekeeper and have to be acknowledged by technicians by holding a capacitive button on their chest for three seconds. Furthermore, technicians can request for assistance by pressing the capacitive button on their chest, also for three seconds. In that case, an emergency signal is sent to the gatekeeper as described in “*Use Case 2: Technician’s Safety*”.

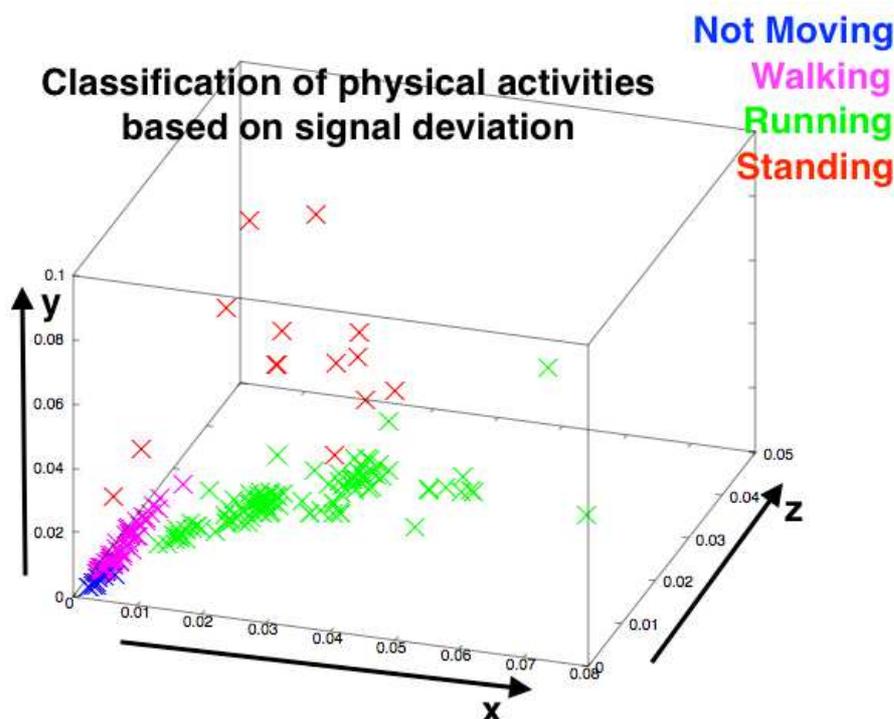


Figure 8.14: Classification of physical activities based on IMU data. Axes correspond to the deviation of the linear acceleration along the x, y and z-axes.

8.2.3 Implementation with TangoHapps

In this section, we describe in detail how the three use cases of the Custodian Jacket are implemented with *TangoHapps*. The Custodian Jacket can be designed using a *Proximity* sensor, a *Three-Color LED*, two *MPU 6050* IMUs, a Button and a *Textile Sensor*. The Custodian Jacket contains two isolated circuits, one on the torso and another one on the sleeve. Each circuit is controlled by a BLE-LilyPad microcontroller that communicates with the smartphone wirelessly over Bluetooth Low Energy. Figure 8.15 displays both circuits designed with *Interactex Designer*.

Use Case 1: Error prevention

An *Interactex* application that provides visual output to users based on proximity sensor and IMU readings can be realized with the following *Happ Objects*:

1. Three *Bigger Comparison Operators*.
2. Three *Smaller Comparison Operator*.
3. Three *AND Logical Operators*.
4. Three *Number* variables. We name them “Tolerance”, “255” and “0”.
5. A *Slider UI Widget*.

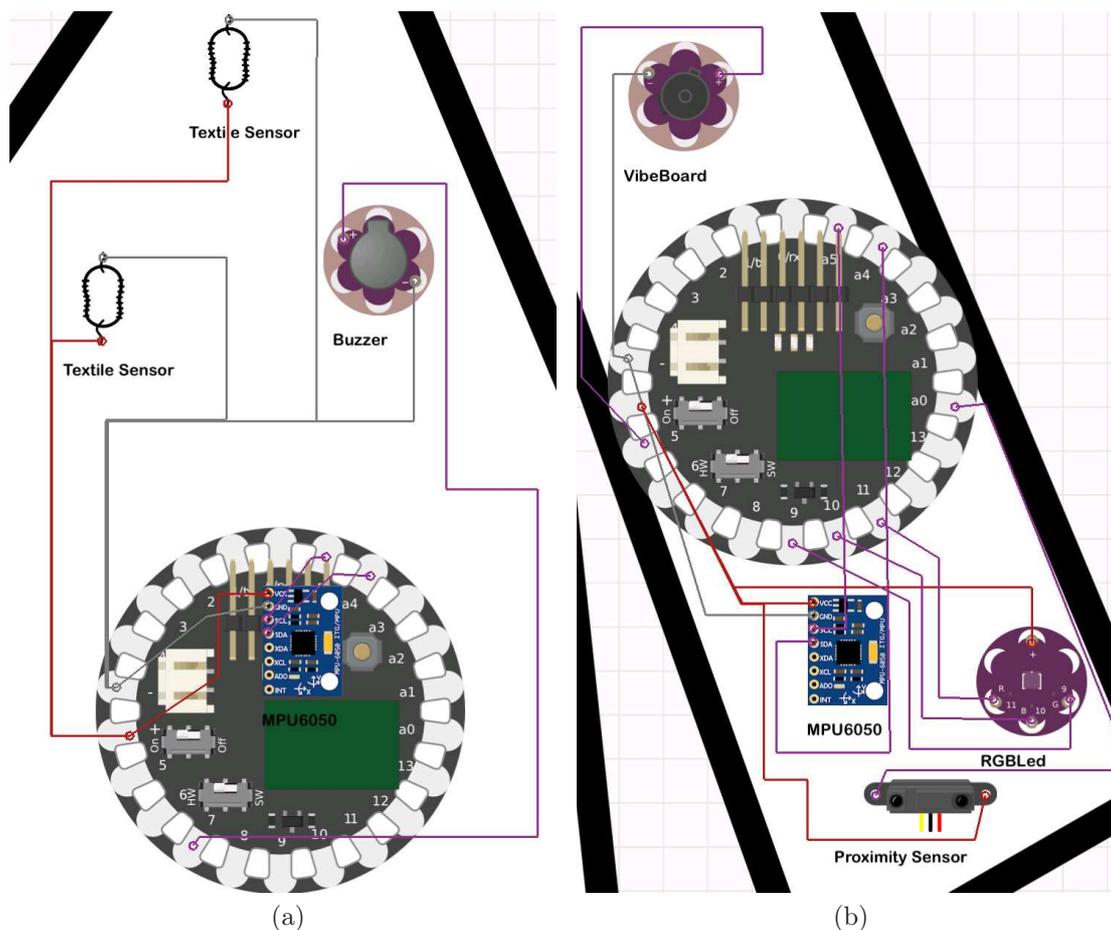


Figure 8.15: Circuit Design of the Custodian Jacket. a) Jacket’s torso including *Buzzer*, *Textile Sensor*, *Button*, *MPU 6050* and *BLE-Lilypad*. b) Jacket’s Sleeve including *MPU 6050*, *Proximity Sensor*, *Three-Color LED* and *BLE-Lilypad*.

We will describe how this application can be developed with *TangoHapps* in three steps. *First*, we create an interface for users of the Custodian Jacket to set the height unit they are looking for over their smartphones. The height unit is set over a slider. In order for the application to compare the proximity values delivered by the proximity sensor to the height unit input by the users, the application should calculate the distance between the height unit to the ground. This is achieved by first multiplying the number of the height unit by 4.5 cm and then adding a distance of 3.0 cm which is the distance to the ground of the first height unit in the rack.

Second, we develop a function that checks whether the value we calculated in the first step is within a range. This is needed in order to detect whether the user’s arm is in front of a specific server. This function can be created as follows:

1. We store the middle value of the range in a *Variable Happ Object*. We call this *Variable: MidValue*.
2. We store a constant in another *Tolerance Variable*. This function will check for values in the range: $[MidValue - Tolerance \dots MidValue + Tolerance]$. *MidValue*

and *Tolerance* are configurable by developers according to specific applications.

3. We use the *Addition* and *Subtraction* operators to compute the lower and upper limits of the range. We set the *MidValue* variable as first operand and *Tolerance* variable as second operand using the operators' *setOperand1()* and *setOperand2()* *Methods*.
4. We use the *BiggerEqual Comparison Operator* to compare the reading from the *Proximity* sensor to the lower limit of the range. The *Subtraction Arithmetic Operator's computed()* *Event* provides the range's upper limit.
5. Similarly, we use the *SmallerEqual Comparison Operator* to compare the proximity to the upper limit of the range.
6. We use the *AND Logical Operator* to ensure that comparisons from steps 4. and 5. are true. We use its *Methods setOperand1()* and *setOperand2()* to the *SmallerEqual* and *BiggerEqual* operators' *conditionChanged()* *Event*.
7. The *AND* operator will emit the *conditionIsTrue()* *Event* when the value is within the specified range and the *conditionIsFalse()* will be triggered when the value is not in the range.

Because height units have a height of 4.5 cm, a tolerance of 2.25 cm would lead to a range that covers the entire height unit. The same function can be applied to check whether the IMU on the sleeve is oriented properly. An IMU that is held parallel to the ground would shield a yaw and pitch of 0 degrees. Therefore, the *MidValue Variable* would be set to 0 and the *Tolerance Variable* to 2. Figure 8.16 a) displays the implementation of the function to check whether a value is within a range in *Interactex Designer*.

Third, we provide feedback to users by setting the *Three-Color LED's* color to either red, green or blue depending on the results of the different comparisons. The *Three-Color LED* offers *Methods* to set its red, blue and green components independently. We define three *Invocations* between the *AND* operator's *conditionIsTrue()* and *conditionIsFalse()* *Events* and the *Three-Color LED's setRed()*, *setGreen()* and *setBlue()* *Methods*. Each *Invocation* passes the *value Variable* of the "0" or "255" *Variable* as a parameter according to the RGB color coding of the color we need in each case. Figure 8.16 b) displays the implementation of this function in *Interactex Designer*.

Finally, we aggregate the results of all three range comparisons (proximity, yaw and pitch) using the *AND Logical Operator* and set the *Three-Color LED's* color accordingly. If the yaw or pitch are not in the [-2 ... 2] range, then the proximity sensor is not properly aligned. In this case, we set the LED's color to red. If the yaw and pitch are properly aligned and the proximity measured by the proximity sensor is in the range we defined, then the user's hand is in front of the server he is looking for. In this case, we set the LED's color to green. If the yaw and pitch are properly aligned but the proximity range comparison returns *false*, then the user's arm is not in front of the server he/she is looking for. In this case we set the LED's color to blue. A schematic of this algorithm is shown in Figure 8.17.

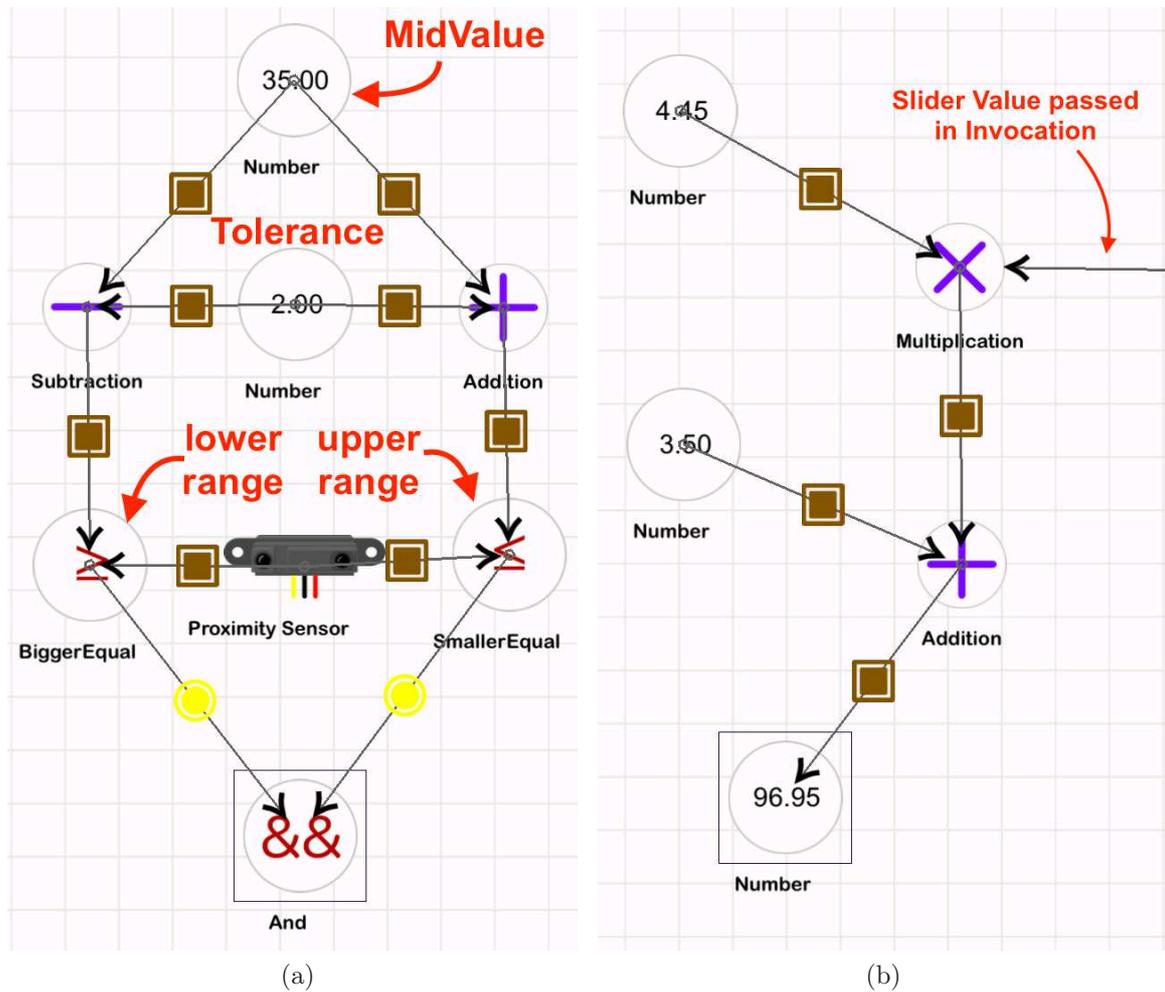


Figure 8.16: *TangoHapps* implementations of: a) Function that checks whether the proximity measured by a *Proximity Sensor* is in the range 35 ± 2 cm. b) Function that sets the color of the *RGB LED* to red.

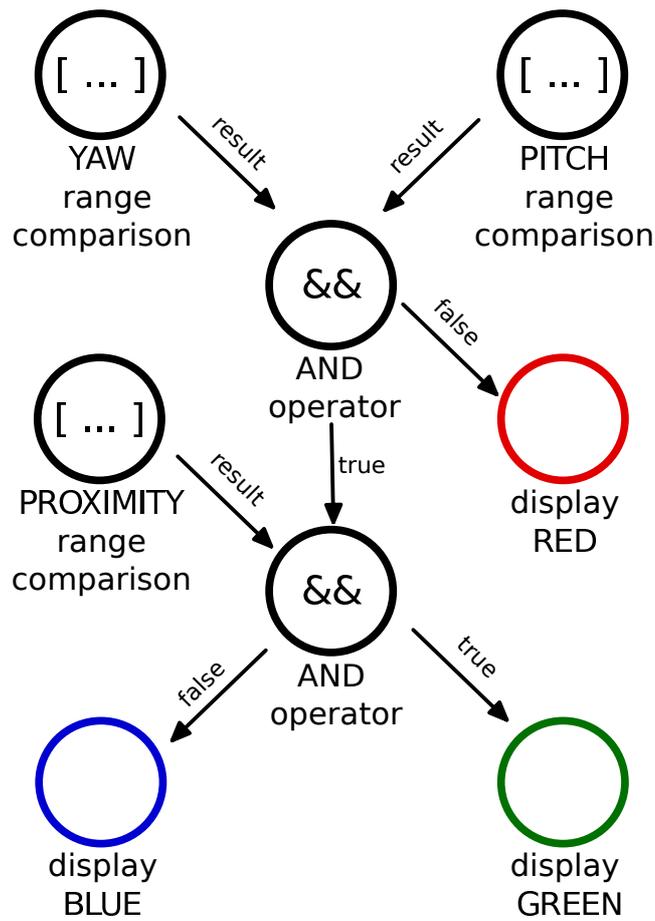


Figure 8.17: Sketch of a *TangoHapps* implementation of the Custodian's *Error Prevention* use case.

Use Case 2: Technician's Safety

We demonstrate an *Interactex* application that monitors technician's physical activity in three steps. *First*, the application should detect whether the user is wearing the jacket. This is done with the capacitive textile sensor sewed to the jacket at the shoulder delivers high analog values when the user's skin is nearby. We determine whether the technician is wearing the jacket by comparing the value delivered by the *Textile Sensor* with a certain threshold. We chose a threshold of 12.00 based on the analog values measured by the last version of the Custodian Jacket when being worn over a T-shirt and a pullover. The threshold can be tuned by developers to suit different capacitive textile sensors. The comparison with a threshold is done by the *Bigger Comparison Operator*. The *Bigger Comparison Operator* invokes the *MPU 6050* IMU to start and stop receiving linear acceleration values.

Second, it uses an IMU and the *Activity Classifier* to determine user activity and a the *Contact Book* to make emergency calls in case the user is detected to be inert for a certain period of time. The linear acceleration along the y-axis measured by the *MPU 6050* IMU is passed over to the *Window Happ Object* to buffer the linear acceleration values. In order to achieve this, the *MPU 6050's yChanged() Event* is connected to the *Window's addValue() Method* passing in a linear acceleration sample as a parameter. The *Window's size* and *overlapping Variables* are configured over the *Configuration View* to gather 50 linear acceleration samples and to overlap 50% of the samples with the last set of buffered values. The *Window's filled() Event* invokes the *Low-Pass Filter's addValues() Method* passing in the buffered samples. The *Low-Pass Filter factor Variable* is set to 0.5 over the *Configuration View*. This implies that the signal's standard deviation will be smaller or equal than half of the original signal's standard deviation. The *Low-Pass Filter's filteredValues() Event* invokes the *Activity Classifier's classifySamples()* passing in the array of filtered samples. So far, the application is able to classify user activity.

In the *third* and last step, the *Activity Classifier's* output should be used to trigger an emergency signal. In order to avoid false positives, the Custodian Jacket triggers an emergency signal if the user has not moved for 20 seconds. The *Timer Happ Object* can be used in order to make the application count for 20 seconds when the user is found to be inert. The *Activity Classifier's notMoving() Event* invokes the *Timer's start() Method*. The *Activity Classifier's walking(), running() and climbing() Events* invoke the *Timer's stop() Method* to cause the *Timer* to stop counting time after the user moved in any way. The *Timer's frequency Variable* is configured to trigger after 20 seconds. Finally, the *Timer's triggered() Event* should invoke the *Contact Book's makeEmergencyCall() Method*. Figure 8.18 displays the implementation of this use case in *TangoHapps*.

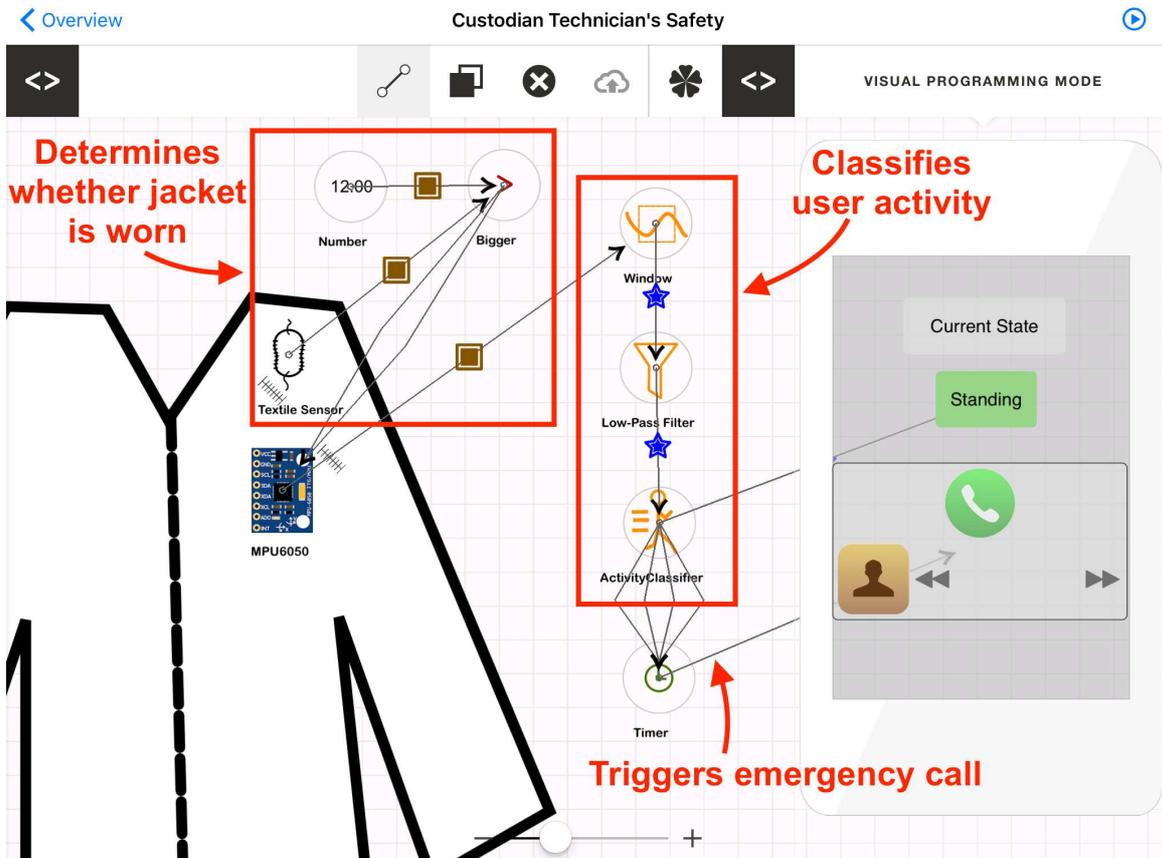


Figure 8.18: Custodian's *Technician's Safety* use case implemented with *TangoHapps*.

Use Case 3: Emergency Situation

We describe the implementation of an application in *TangoHapps* that produces sounds and vibrations when an emergency signal is received. This application simulates emergency signals with a button on the smartphone. The real application would receive emergency signals remotely over Wi-Fi.

This application relies on a function to detect whether a textile button is being pressed for more than three seconds. Two *Invocations* are needed in order to make sound and vibrations to start and stop playing. First, an *Invocation* between the smartphone's *Button's* *buttonPressed()* and the *Buzzer* and *Vibration Board's* *turnOn()* *Methods* should be defined. Second, the *Timer's* *triggered()* *Event* should invoke the *Buzzer* and *Vibration Board's* *turnOff()* *Method*. This causes the *Timer* to turn off the *Buzzer* and *Vibration Board*. The *Timer's* *frequency* *Variable* is configured to trigger three seconds after it has been started. The *Timer* is started whenever the textile button is detected to be pressed. In order to detect whether the button has been pressed, we compare whether the capacitance measured by the *Textile Sensor* is bigger than a threshold. We chose a threshold of 97.00 based on the analog values measured by the textile fabric we used on the last version of the Custodian Jacket. An additional *Invocation* between the *Bigger Comparison Operator's* *conditionIsFalse()* *Event* and the *Timer's* *stop()* *Method* stops the timer whenever the user has released the textile button. The *Timer*, when stopped, resets its counter so that users have to start pressing the textile button for three consecutive seconds. The implementation of this use case is shown in Figure 8.19. In order to additionally emit emergency calls when the textile button has been pressed for three consecutive seconds, an additional *Invocation* between the *Timer's* *triggered()* *Event* and the *Contact Book's* *makeEmergencyCall()* *Method* should be defined.

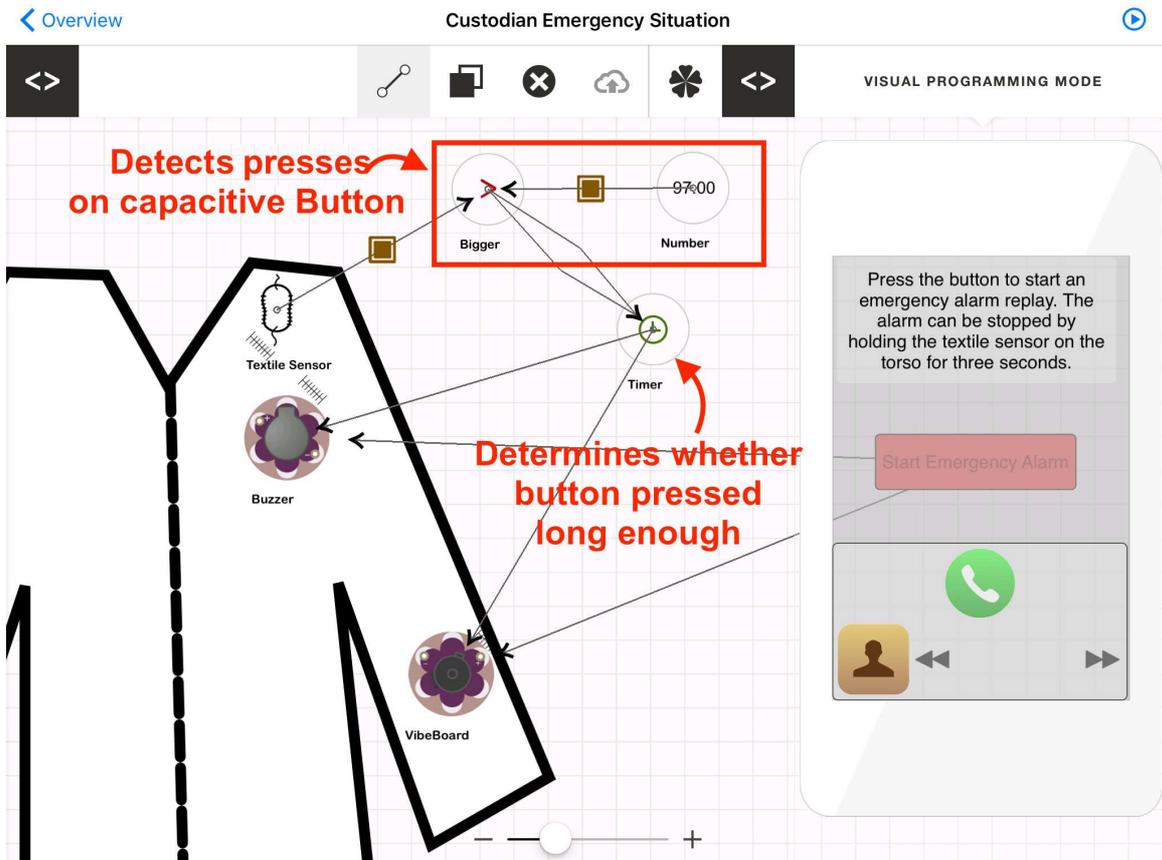


Figure 8.19: Custodian’s *Emergency Situation* use case implemented with *Tango-Happs*.

Chapter 9

Evaluation

New toolkits and development environments are often validated with the breadth of coverage of applications they support and by their ability to simplify the development [7, 8, 39, 62, 63]. We have demonstrated the breadth of coverage of *TangoHapps* by the applications we described in Chapter 8. In this chapter, we present the results of two user studies we conducted with the goal to assess *TangoHapps*' ability to simplify the development of smart textiles. The first user study was conducted with novice users and the second one with professional smart textile developers. Both user groups had none or limited background experience in electronics and programming.

9.1 User Study 1: Novice Users

To assess the ease of use of *Interactex*, we conducted a study in a middle-school classroom with 18 teenagers (7 female, 11 male, 14 and 15 years old). None of the participants had experience in programming or electronics. Participants were divided in six teams of three members per team. Each team received an Arduino UNO Kit containing vibration motors, buzzers, LEDs, light and temperature sensors to test their applications and a "cheat sheet" with circuit layout diagrams describing how electronic devices should be connected to the Arduino UNO microcontroller. The study lasted two days; four hours each day. During the first day, we taught participants the basics for creating a circuit layout and programming using the Arduino IDE. During the second day, we provided a one-hour introduction to *Interactex*. We showed participants different *Application Objects*, created *Invocations* between them and demonstrated how to create a circuit layout and deploy the application to the smartphone. We then gave participants an iPad and iPhone with *Interactex* already installed and asked them to solve the following warm-up tasks:

1. Turn on and off an LED using a switch on the smartphone
2. Iterate the music playing in the smartphone using a physical push button
3. Make a call to any number when the temperature decreases over certain threshold.

Finally, participants were given 45 minutes to develop their own application using *Interactex*.

9.1.1 Results

Participants solved warm-up tasks 1. and 2. without our support in less than ten minutes each. In order to solve these tasks, participants had to understand *Interactex*'s workflow and event-function coupling. Warm-up task 3. required the usage of the *Comparator* object to compare the value measured by the temperature sensor to the value contained within a *Variable* object. Most participants attached the temperature sensor to the *Comparator* and forgot to specify the *Variable* it should be compared to. This highlighted a potential improvement to *Interactex*'s visual programming mode to make events and methods of objects visible in the canvas as done in other flow-based programming environments. However, every team was able to finish this task after we showed them how the *Comparator* object works in a separate application.

Participants developed the following applications of their own:

1. A jacket with 12 integrated LEDs that the wearer can turn on and off from two buttons on a smartphone. LEDs turn on in a sequential fashion.
2. A jacket with a button that plays and stops playing the music on the smartphone. The volume of the music changes according to the light intensity sensed by a light sensor.
3. A T-Shirt that adapts an LED's intensity according to the light's intensity in the room.
4. A T-Shirt that measures the user's body temperature and displays it on a smartphone.
5. A T-Shirt that plays beep sounds. The frequency of the tone can be controlled over a slider on the smartphone.
6. A jacket to help find victims of an avalanche. The jacket produces sound and vibration while a button on the jacket is being pressed or while the light's intensity is smaller than a threshold.

These applications would require extensive expertise to develop without *Interactex* (e.g. data transmission, synchronization and smartphone app development). The reason why most teams chose a T-Shirt for their applications is that the default textile in *Interactex* is a T-Shirt. The T-Shirt can be replaced by another image stored in the iPad after adding the object to the canvas. Figure 9.1 a) shows three female participants developing an electronic circuit during the study, Figure 9.2 a) shows Application 1. consisting of 12 LEDs connected to the Arduino UNO and b) displays the paper prototype developed by the same team.

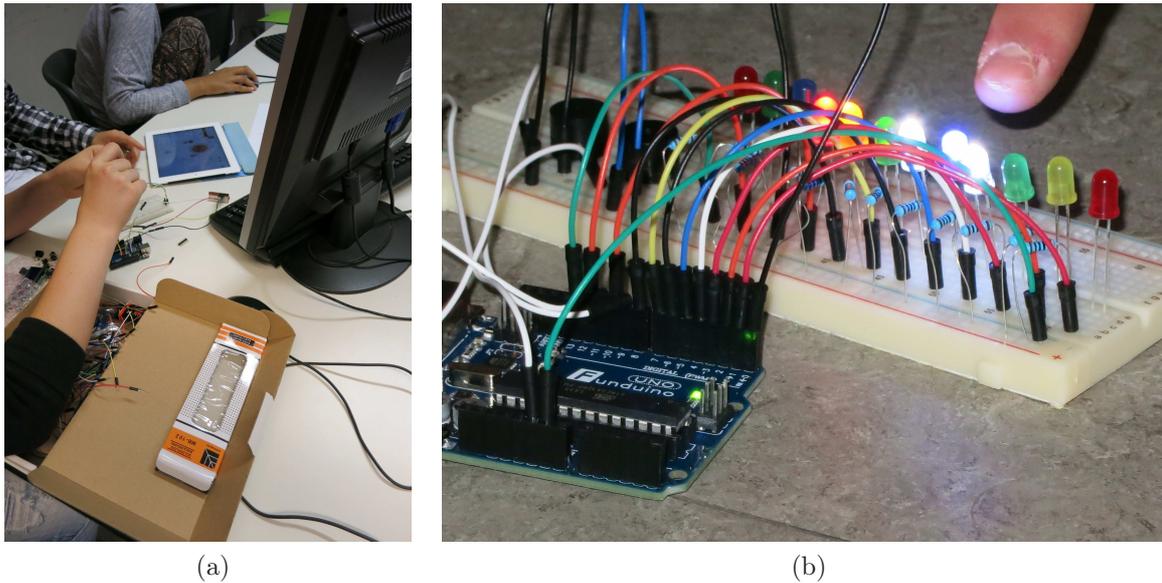


Figure 9.1: a) Three female subjects develop an electronic circuit with *Interactex Designer*. b) Application 1 consisting of 12 LEDs.

9.2 User Study 2: Professional Smart Textile Developers

In our second study, we conducted interviews with eight professional smart textile developers (2 male, 6 female, 23 - 35 years old) with varying backgrounds: textile design (5), interaction design (2) and computer science (1).

The study was conducted in five steps:

1. To enable participants to test their applications on an actual textile, we developed a textile testbed. The textile testbed is shown in Figure 9.2 b). *Interactex Firmware* was uploaded into the testbed prior to this study.
2. We started every interview by explaining our intentions to participants emphasizing our desire to assess *Interactex's* suitability for prototyping of smart textiles including positive and negative aspects. We then demonstrated the usage of the IDE with an application that turns on and off an LED in the textile testbed using two buttons on the smartphone's UI.
3. Participants had to develop any application while thinking aloud for around 20 minutes.
4. We conducted a semi-structured interview to gain insight into participant's impression about *Interactex* and to identify missing functionality. We recorded and transcribed participant's think aloud protocols and interview answers for post-analysis.

5. Participants filled a questionnaire that inquired about their professional practices regarding smart textile development and general impression about *Interactex*.

9.2.1 Study Results

When asked about their general impression about *Interactex*, every participant made at least one comment praising it. P3: "Good to know that this is available in the App Store!", P4: "even though I know how to program, I could use it and welcome that you don't need to program", P6: "I love it!". The design of the environment was one of the most liked features. P4: "the simplicity is the beauty of it", P7: "good! nice graphic interface" - right after showing *Interactex Designer* to him for the first time. Participants also praised the easiness to deploy applications to the smart textile. P6: "the workflow was perfect and easy to understand". The ability to use a smartphone to read sensor data and manipulate output devices on the testbed was perceived as one of the most convincing and impressive features. P3: "Really nice that iPhone and hardware are programmed directly and visually in the iPad environment". P5: "... it is convenient that a phone manages the interaction with the hardware".

From the feedback provided by participants, we extracted a total of 85 feature requests. Five participants requested support for new sensors, output devices and microcontrollers. Wishes included pressure sensors, servo motors and hardware components from other hardware brands (e.g. Sparkfun. Adafruit). Participants also questioned the lack of support for custom-made textile sensors, although these sensors correspond in their functionality to elements supported in *Interactex* (e.g. Switch and Textile sensor). Two participants additionally pointed out the idea to have the iPad communicate directly with the smart textile.

When asked about what they did not like about *Interactex*, participants mentioned different usability flaws. The usability flaw most mentioned (by five participants) was the lack of immediate information to users about which *Invocations* were defined in the project coupling. P4 explained it as: "though in the properties of the components things are marked in blue when selected [...] there is nothing to say if one connection is this event or this method" - referring to a coupling as "connection". Other usability flaws mentioned were the lack of a way to scale the textile independently from electronic devices attached to it and the lack of a clear indication whether a defined *Invocation* is valid.

9.2.2 Questionnaire results

The results from the questionnaire revealed that every participant had only basic programming knowledge with the exception of P2, who was a professional software developer. Participants mentioned Arduino IDE and Processing as their usual programming environments and P2 additionally mentioned Android Studio. Further-

9.2. USER STUDY 2: PROFESSIONAL SMART TEXTILE DEVELOPERS

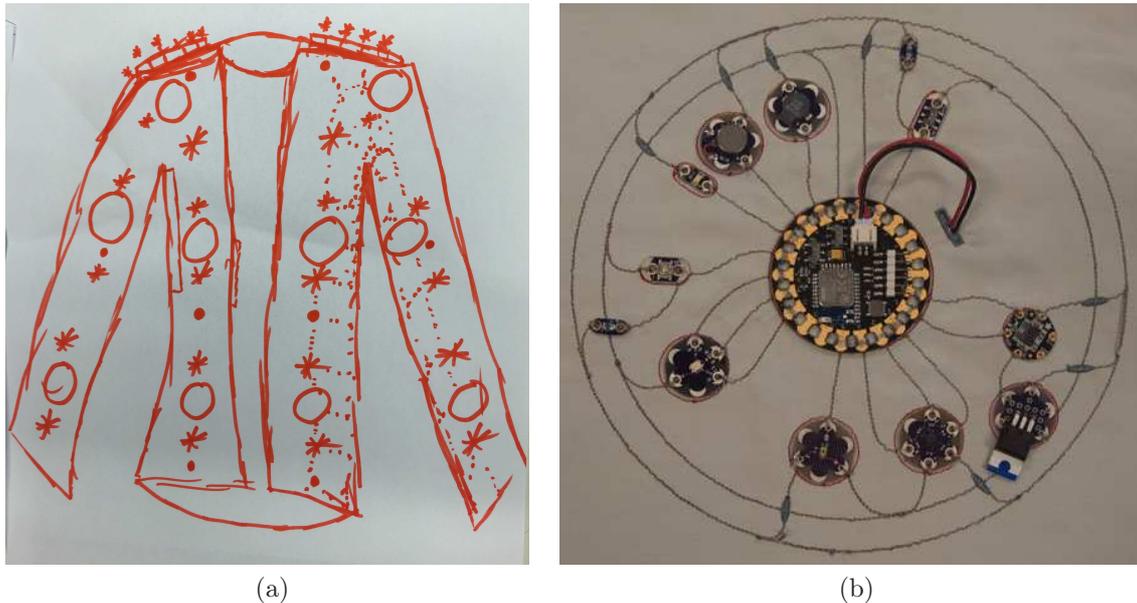


Figure 9.2: a) A paper prototype of a jacket developed during the Study 1. b) Textile testbed containing two accelerometers, a push button, switch, light sensor, buzzer, vibration motor and a three-color LED. Every electronic device is connected with conductive fabric to a custom-made Arduino Lilypad with an integrated BLE module (in the center).

more, every participant reported being used to dealing with code editors and not with visual programming environments.

The questionnaire also contained two questions in a 5-point Likert scale. The overall suitability of *Interactex* for prototyping of smart textiles was rated by participants as: *Very good* (3), *Good* (3), *Satisfactory* (2), *Sufficient* (0) and *Poor* (0). The overall positive ratings are consistent with participants' reactions during the interviews. The *Satisfactory* ratings were given by P2 and P8. P2 had extensive experience in programming and electronics and considered *Interactex* to be less suitable for herself. This is in agreement with *Interactex*'s target user group of individuals with little experience in programming. P8's rating was related to her thought that the visual programming abstractions offered in *Interactex* would be too difficult for inexperienced developers. While our findings from the study with novice users show that *Interactex*'s visual programming abstractions can be grasped by users who had never programmed before, we agree with P8 in that the realization of more complex applications might require a longer learning phase than the average 20 minutes participants took for this study. Other participants perceived the difficulty to use *Interactex* to be normal or easy. Difficulty ratings were: *Very easy* (0), *Easy* (2), *Normal* (5), *Difficult* (1) and *Very Difficult* (0).

CHAPTER 9. EVALUATION

Chapter 10

Conclusions and Future Work

Several solutions have been introduced in the research community that greatly simplify the amount of effort and knowledge required to create physical electronic devices. For example, hardware toolkits provide hardware components that can be plug and played in order to create a functional electronic device within minutes. Thanks to these solutions, hardware devices are today being built by hobbyists who have no previous background in electrical engineering. On the other hand, tools that simplify development of software for smart textiles either have an educational purpose or require experience in programming.

In this dissertation, we have made three main contributions. *First*, we have presented a comprehensive literature review of the field of smart textiles, including definitions, their evolution from the so-called *wearable computers* and technologies currently being used to create them. We have also listed and categorized tools that support the development of smart textiles, wearable computers and physical devices and summarized their common features. Furthermore, this dissertation has contributed to the body of knowledge in the field of smart textile development by eliciting generic requirements for an IDE for smart textiles. We elicited these requirements based on an exhaustive literature review that is summarized in Chapter 2 and a four year collaboration with professional textile designers. We have described the research process we followed to iteratively and incrementally explore the requirements needed to create a development environment for smart textiles. The requirements we elicited and research process described in Chapter 1 can be reused in the creation of further development tools for smart textiles.

Second, we have developed and fully described an integrated development environment for smart textiles. In this dissertation, we have described the models, design decisions, architecture and internal structure of *TangoHapps*. *TangoHapps* supports the design, implementation, testing and debugging of smart textiles, including placement of electronics, circuit layout and application development. The high-level abstractions available in *TangoHapps* lower the entrance barrier to smart textile development. *TangoHapps*' visual programming semantics span across smartphone and smart textile enabling users to take advantage of capabilities present in both devices without writing source code. In order to facilitate other researchers and professionals in the prototyping and further development of smart textiles, we made *TangoHapps*

publicly available. Its source code is available in GitHub¹ and its executable can be downloaded from Apple’s AppStore²³. Furthermore, we have presented exhaustive documentation of the IDE, which should facilitate its usage and further development. In particular, a detailed description of every *Happ Object* supported in *TangoHapps* is available in the Appendix. We also created a platform for smart textile developers to share their *Interactex* projects⁴ with other users. The platform also provides step-by-step tutorials on the usage of *Interactex*.

Third, we demonstrated that *TangoHapps* is reusable across different application domains and has a low-entrance barrier. We demonstrated its broad range of application coverage with two use cases from different application domains: rehabilitation and blue-collar working. Furthermore, we addressed the low-entrance barrier of *TangoHapps* with two user studies. The first user study was conducted with middle school students who had never programmed or worked with electronic circuits before. The second user study was conducted with experienced smart textile developers who tested *TangoHapps* and gave us their insight as end users of the IDE. Both user studies provide evidence that *TangoHapps* enables users with no experience in programming or electronics to develop applications for smart textiles within minutes. Parts of the contributions presented in this dissertation have been published in [44, 45, 43, 46, 48, 47].

10.1 Future Work

Before we are able to see people walking around the streets wearing smart garments, challenges such as washability, robustness and social acceptance must be overcome. Washability refers to the ability of a textile to resist washing without deteriorating. The strain (and surprisingly not the water, heat or detergent) a smart textile is exposed to during a washing cycle can damage its electronic devices and connections. During normal usage, textiles are also exposed to high degrees of bending and strain. For example, a study done by the ETH Zurich revealed that the upper back of a T-Shirt can be subject to an elongation as high as 20% of its normal size [81]. Therefore, smart garments should be able to resist high degrees of strain. Additionally, the integration of electronics into garments raises safety concerns and is perceived as socially awkward by some audiences (e.g. elderly) [20].

The requirements for an IDE for smart textiles will evolve together with the field of smart textiles. As new technologies become available and new applications are discovered, solutions to develop such applications more efficiently will be needed. As we have demonstrated, *TangoHapps* can already be used to develop real-world smart textile applications. In the future, *TangoHapps* should be evaluated in longer term multidisciplinary smart textile projects.

A long term vision of *TangoHapps* is to enable everyone to develop their own smart textiles. Users could design and develop the applications for a smart garment

¹<https://github.com/ls1intum/Interactex>

²<https://itunes.apple.com/us/app/interactex-designer/id973912620?mt=8>

³<https://itunes.apple.com/us/app/interactex-client/id1031238223?mt=8>

⁴<http://www.interactex.de/>

using *TangoHapps* and snap or sew the electronic devices to the textile. *TangoHapps* currently supports multiple applications on the same smart garment. Hence, users can already switch their smart garment application by simply selecting a different application in their smartphones.

Furthermore, an online platform could be used to distribute applications for smart textiles, in a similar way to how Apple's AppStore offers applications for mobile devices. Users could develop Happs using *TangoHapps* and then upload their executable and textile assembly instructions to the *TangoHapps'* HappStore for other users to download them.

CHAPTER 10. CONCLUSIONS AND FUTURE WORK

Appendix A

Happ Objects

A.1 Events Methods and Variables

In this section we list every *Happ Object's Methods, Events* and *Variables*.

Image	Name	Description	Events	Methods	Variables
	Button	Triggers events when pressed and when released.	<i>buttonPressed()</i> , <i>buttonReleased()</i>	-	<i>pressed : Boolean</i>
	Label	Displays text and numbers.	-	<i>setText(String)</i> , <i>appendText(String)</i>	<i>text : String</i>
	Switch	Triggers events when switched <i>on</i> or <i>off</i> .	<i>switchedOn()</i> , <i>switchedOff()</i> , <i>onChanged(Boolean)</i>	-	<i>on : Boolean</i>
	Slider	Used to select a value from a range of values. Triggers events when its handle is moved by the user.	<i>valueChanged(Number)</i>	-	<i>value : Number</i>
	Touchpad	Triggers events when the user performs the following multi-touch gestures on it: <i>tap</i> , <i>double tap</i> , <i>long tap</i> , <i>pinch</i> and <i>pan</i> .	<i>pannedX(Number)</i> , <i>pannedY(Number)</i> , <i>tapped()</i> , <i>doubleTapped()</i> , <i>pinched(Number)</i>	-	-
	Music Player	Accesses mobile device's music library, offers functionality to iterate through the music list, play songs and displays music information.	<i>started()</i> , <i>stopped()</i>	<i>play()</i> , <i>pause()</i> , <i>next()</i> , <i>previous()</i> , <i>setVolume(Number)</i>	<i>volume : Number</i>
	Contact Book	Accesses user's contacts and offers functionality to iterate through contacts and make calls.	-	<i>call()</i> , <i>previous()</i> , <i>next()</i>	-
	Image View	Displays an image.	-	-	-
	Monitor	Displays values over time (e.g. sensor readings).	-	<i>addValue1(Number)</i> , <i>addValue2(Number)</i>	-

Table A.1: *UI Widgets* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	Button	A button that can be pressed. Triggers events when pressed and when released.	<i>buttonPressed()</i> , <i>buttonReleased()</i>	-	<i>pressed : Boolean</i>
	Switch	An on/off switch. Triggers events when switched <i>on</i> or <i>off</i> .	<i>switchedOn()</i> , <i>switchedOff()</i> , <i>switchChanged(Boolean)</i>	-	-
	Light Sensor	Measures light intensity.	<i>valueChanged(Number)</i>	<i>start()</i> , <i>stop()</i>	<i>lightIntensity : Number</i>
	Temperature Sensor	Measures temperature.	<i>valueChanged(Number)</i>	<i>start()</i> , <i>stop()</i>	<i>temperature : Number</i>
	Accelerometer	Measures acceleration forces in a 3D space.	<i>xChanged(Number)</i> , <i>yChanged(Number)</i> , <i>zChanged(Number)</i>	<i>start()</i> , <i>stop()</i>	<i>x : Number, y : Number, z : Number</i>
	LSM Compass	Measures acceleration forces and direction of magnetic fields in a 3D space.	<i>valueChanged(Object)</i> , <i>headingChanged(Number)</i>	<i>start()</i> , <i>stop()</i>	<i>acceleration : Object</i> , <i>heading : Number</i>
	MPU 6050	Measures acceleration acceleration forces and its orientation in a 3D space.	<i>accelerationChanged(Object)</i> , <i>orientationChanged(Object)</i> , <i>yawChanged(Number)</i> , <i>pitchChanged(Number)</i> , <i>rollChanged(Number)</i> , <i>xChanged(Number)</i> , <i>yChanged(Number)</i> , <i>zChanged(Number)</i>	<i>start()</i> , <i>stop()</i>	<i>acceleration: Object</i> , <i>orientation: Object</i>
	Textile Sensor	Represents an analog textile sensor.	<i>valueChanged(Number)</i>	<i>start()</i> , <i>stop()</i>	<i>value : Number</i>
	Proximity Sensor	Measures the distance to any object within a range of up to 10 meters.	<i>proximityChanged(Number)</i>	<i>start()</i> , <i>stop()</i>	<i>proximity : Number</i>

Table A.2: *Sensors* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	LED	A Light Emitting Diode (LED). Can be turned on or off and its intensity can be set.	<i>turnedOn()</i> , <i>turnedOff()</i>	<i>turnOn()</i> , <i>turnOff()</i> , <i>setIntensity()</i>	<i>on</i> : <i>Boolean</i> , <i>intensity</i> : <i>Number</i>
	Buzzer	An element that produces a sound frequency. It can be turned on, turned off and its sound frequency can be set.	-	<i>turnOn()</i> , <i>turnOff()</i> , <i>setFrequency(Number)</i>	<i>frequency</i> : <i>Number</i>
	Three-Color LED	An LED that emits light in multiple colors.	-	<i>turnOn()</i> , <i>turnOff()</i> , <i>setRed(Number)</i> , <i>setGreen(Number)</i> , <i>setBlue(Number)</i>	-
	Vibration Board	Produces vibrations. It can be turned on, off and its vibration frequency can be set.	-	<i>turnOn()</i> , <i>turnOff()</i> , <i>setFrequency(Number)</i>	<i>frequency</i> : <i>Number</i>
	Textile Speaker	Represents a radio module.	<i>onChanged(Boolean)</i>	<i>turnOn()</i> , <i>turnOff()</i> , <i>setFrequency(Number)</i> , <i>setVolume(Number)</i>	

Table A.3: *Output Devices* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	Lilypad	Official Lilypad Arduino Main Board.	-	-	-
	Lilypad Simple	Official Lilypad Arduino Simple. Has less pins than the Lilypad Arduino Main Board.	-	-	-
	BLE-Lilypad	Lilypad Arduino Main Board with an integrated Bluetooth Low Energy module.	-	-	-

Table A.4: *Microcontrollers* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	Boolean	Stores a Boolean value.	<i>valueChanged(Boolean)</i>	<i>setValue(Boolean)</i>	<i>value : Boolean</i>
	Number	Stores a number.	<i>valueChanged(Number)</i>	<i>setValue(Number)</i>	<i>value : Number</i>
	String	Stores a String.	<i>valueChanged(String)</i>	<i>setValue(String)</i>	<i>value : String</i>

Table A.5: *Variables* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	Bigger	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if the first input variable is bigger than the second input <i>Variable</i> . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a boolean <i>Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Number)</i> , <i>setValue2(Number)</i>	<i>isTrue : Boolean</i>
	BiggerEqual	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if the first input variable is bigger or equal than the second input <i>Variable</i> . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a boolean <i>Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Number)</i> , <i>setValue2(Number)</i>	<i>isTrue : Boolean</i>
	Smaller	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if the first input variable is smaller than the second input <i>Variable</i> . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a boolean <i>Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Number)</i> , <i>setValue2(Number)</i>	<i>isTrue : Boolean</i>
	SmallerEqual	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if the first input variable is smaller or equal than the second input <i>Variable</i> . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a boolean <i>Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Number)</i> , <i>setValue2(Number)</i>	<i>isTrue : Boolean</i>

Table A.6: Comparison Operators supported by TangoHapps.

Image	Name	Description	Events	Methods	Variables
	Equal	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if the first input variable is equal to the second input <i>Variable</i> . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a <i>boolean Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Number)</i> , <i>setValue2(Number)</i>	<i>isTrue : Boolean</i>
	NotEqual	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if the first input variable is not equal to the second input <i>Variable</i> . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a <i>boolean Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Number)</i> , <i>setValue2(Number)</i>	<i>isTrue : Boolean</i>
	And	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if both boolean input variables are true . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a <i>boolean Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Boolean)</i> , <i>setValue2(Boolean)</i>	<i>isTrue : Boolean</i>
	Or	Emits the <i>conditionIsTrue()</i> and <i>conditionChanged()</i> Events with a <i>Boolean Variable</i> set to <i>true</i> if either boolean input variable is true . Otherwise emits <i>conditionIsFalse()</i> and <i>conditionChanged()</i> with a <i>boolean Variable</i> set to <i>false</i> .	<i>conditionIsTrue()</i> , <i>conditionIsFalse()</i> , <i>condition-Changed(Boolean)</i>	<i>setValue1(Boolean)</i> , <i>setValue2(Boolean)</i>	<i>isTrue : Boolean</i>

Table A.7: *Logical Operators* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	Addition	Adds two numbers.	<i>computed(Number)</i>	<i>setOperand1(Number), setOperand2(Number), compute()</i>	<i>result : Number</i>
	Subtraction	Subtracts <i>operand2</i> from <i>operand1</i> .	<i>computed(Number)</i>	<i>setOperand1(Number), setOperand2(Number), compute()</i>	<i>result : Number</i>
	Multiplication	Multiplies two numbers	<i>computed(Number)</i>	<i>setOperand1(Number), setOperand2(Number), compute()</i>	<i>result : Number</i>
	Division	Divides <i>operand1</i> by <i>operand2</i> .	<i>computed(Number)</i>	<i>setOperand1(Number), setOperand2(Number), compute()</i>	<i>result : Number</i>
	Modulo	Calculates the residual of the division between <i>operand1</i> and <i>operand2</i> .	<i>computed(Number)</i>	<i>setOperand1(Number), setOperand2(Number), compute()</i>	<i>result : Number</i>

Table A.8: *Arithmetic Operators* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	Window	Takes signal values as input, and returns chunks of the same signals, which might overlap.	<i>filled(Object)</i>	<i>addValue(Number)</i>	<i>data : Object</i>
	Low-Pass Filter	Low-passes a signal.	<i>filteredValues(Object)</i>	<i>addValue(Number), addValues(Object), removeAllValues()</i>	-
	High-Pass Filter	High-passes a signal.	<i>filteredValues(Object)</i>	<i>addValue(Number), addValues(Object), removeAllValues()</i>	-
	Mean Extractor	Calculates the mean of a set of values.	<i>featureExtracted(Number)</i>	<i>addValue(Number), addValues(Object), removeAllValues(), compute()</i>	-
	Deviation Extractor	Calculates the deviation of a set of values.	<i>featureExtracted(Number)</i>	<i>addValue(Number), addValues(Object), removeAllValues(), compute()</i>	-
	Peak Detector	Calculates the peak in a set of values and provides the peak's value and index in an event. It offers methods to set the range in a set of samples where the peak should be searched.	<i>featureExtracted(Number), indexExtracted(Number)</i>	<i>addValue(Number), removeAllValues(), compute(), setRangeStart(Number), setRangeEnd(Number)</i>	-
	Activity Classifier	Classifies user motion based on accelerometer input. Activities supported are: 'walking', 'running', 'climbing' and 'not moving'.	<i>walking(), running(), climbing(), notMoving()</i>	<i>classifySamples(Object)</i>	-
	Posture Classifier	Classifies user postures based on IMU input. Supported postures are: 'standing', 'lying down on stomach' and 'lying down on back'.	<i>standing(), lyingDown(), lyingUp()</i>	<i>classifySamples(Object)</i>	-

Table A.9: Signal processing and classification *Functions* supported by *TangoHapps*.

Image	Name	Description	Events	Methods	Variables
	Timer	Generates an event after x time. It's trigger time and whether it should trigger once or many times can be configured over the <i>Configuration View</i> in the <i>Palette</i> .	<i>triggered()</i>	<i>start(), stop()</i>	-
	Sound	Represents a sound. It offers a single method to play it.	-	<i>play()</i>	-
	Recorder	Provides a way to store a set of values (e.g. from a sensor) and to feed a set of values into other objects.	<i>startRecording(), stopRecording()</i>	<i>startRecording(), stopRecording(), addValue(Number)</i>	-
	Mapper	Scales values. This is useful to make values fit to the range required by other objects. For example, the <i>Slider</i> UI widget produces by default values in the range [0 255] and the <i>Buzzer</i> hardware element requires a frequency in the range [0 20000]. The <i>Mapper</i> would take the <i>Slider</i> 's input range [Min1 Max1] and scale it linearly to the <i>Buzzer</i> 's output range [Min2 Max2]. Generates an event whenever the value changes.	<i>valueChanged(Number)</i>	<i>setMin1(Number), setMax1(Number), setMin2(Number), setMax2(Number), setValue(Number)</i>	<i>value : Number</i>

Table A.10: Utility programming elements supported by *TangoHapps*.

A.2 Palette Objects

Figures A.1 and A.2 display every object available in *Interactex Designer*.

A.3 Variables

In addition to the *Variables* that we listed in Section A.1, *Happ Objects* expose *Variables* that are used by developers to change the *Happ Object's* default configuration. These *Variables* are displayed over the *Configuration View* in the *Palette*. It should be noted that *Variables* of a superclass are shared among all its subclasses and that *Microcontrollers*, *Arithmetic Operators* and *Logical Operators* have no *Variables*. Figure A.3 in the Appendix displays the *Variables* of a *Label* and *Timer* objects.

APPENDIX A. HAPP OBJECTS

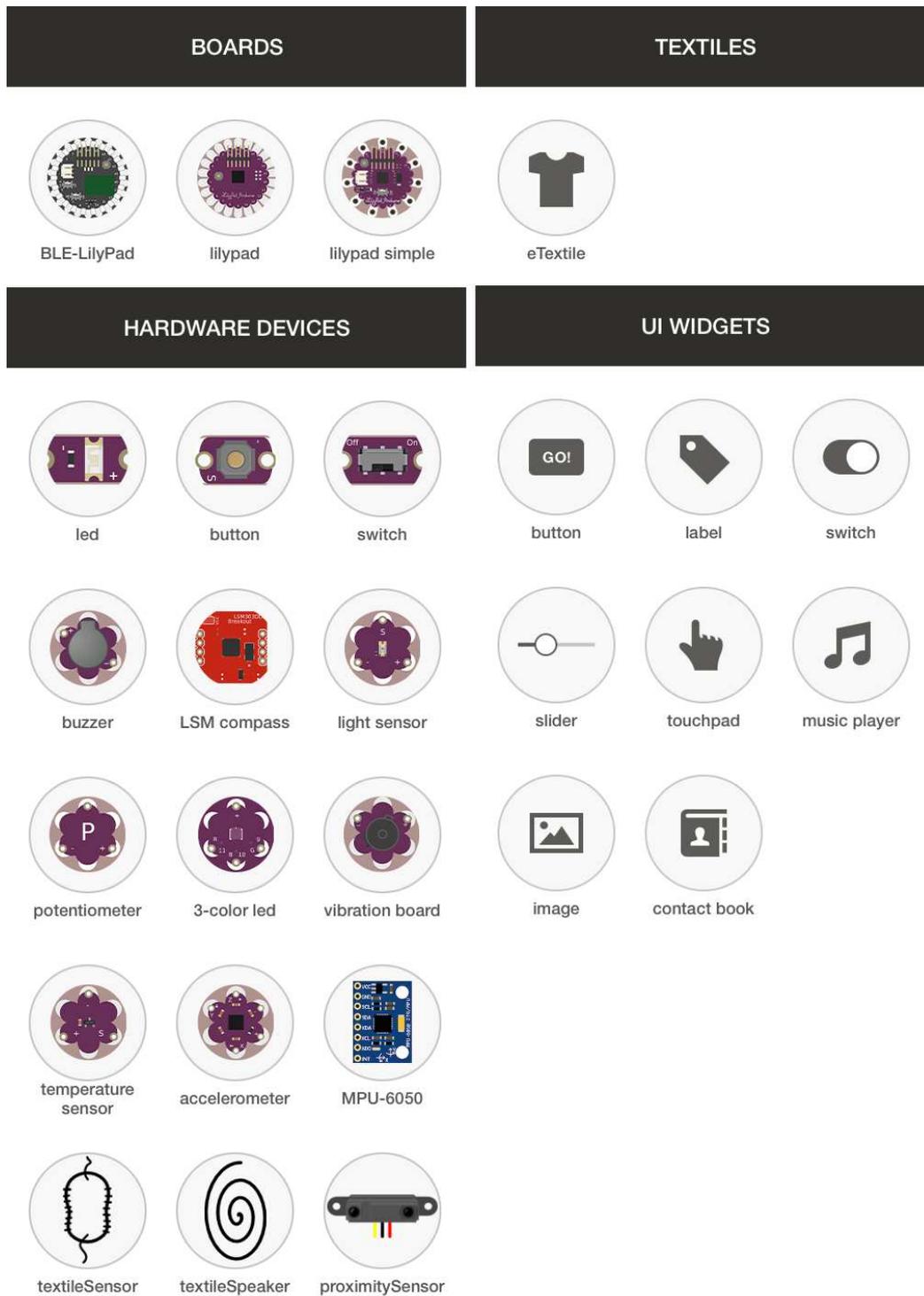


Figure A.1: *Palette Objects* for instantiating *Smart Textiles*, *Electronic Devices* (e.g. *Hardware Devices* and *Microcontrollers*) and *UI Widgets* available in *Interactex Designer*.

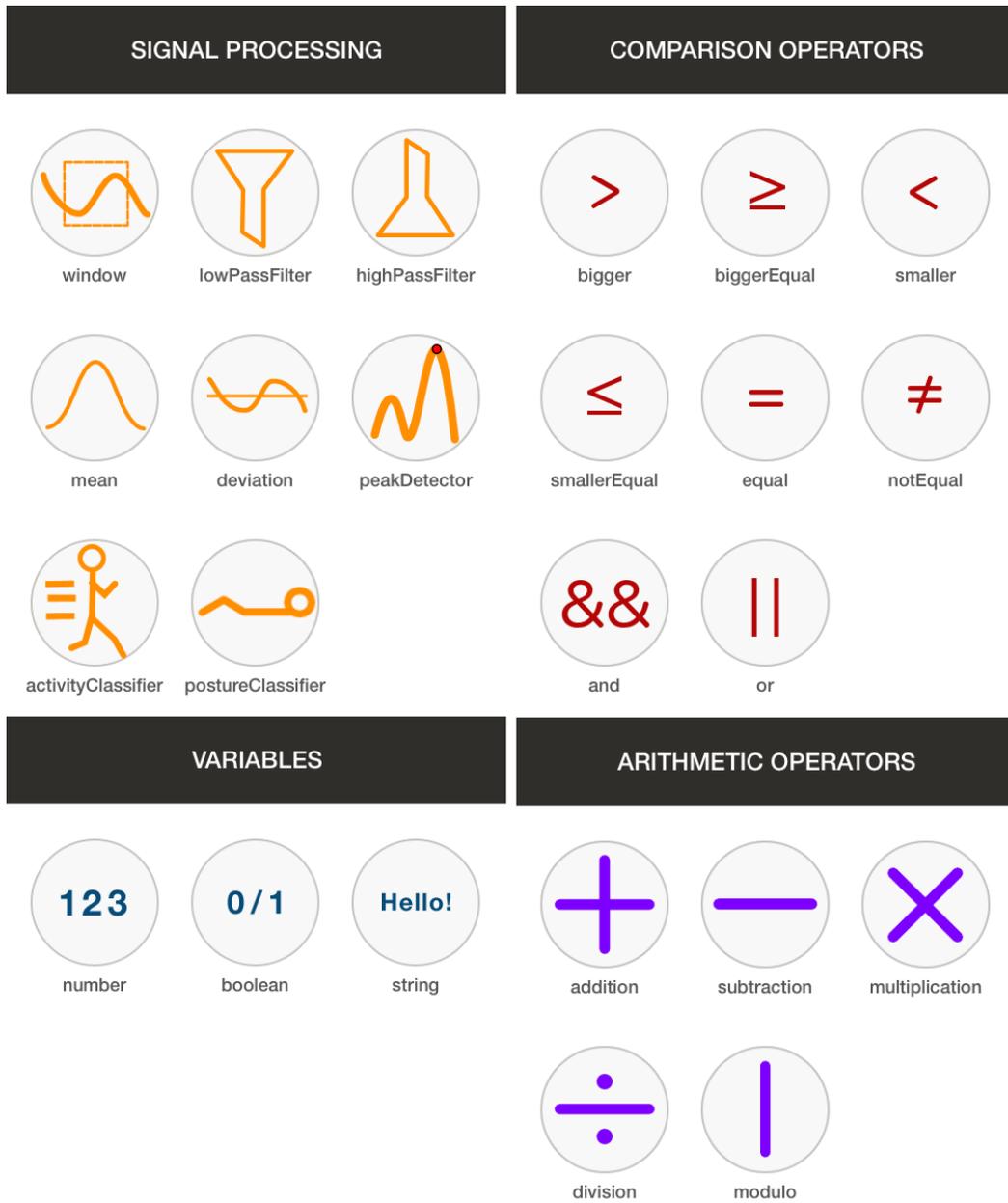


Figure A.2: *Palette Objects* for instantiating *Signal Processing Functions*, *Variables* and *Operators* available in *Interactex Designer*.

Class	Variables
UI Widget	<ul style="list-style-type: none"> • <i>Width</i>. Used to set the width of the <i>UI Widget</i>. • <i>Height</i>. Sets the height of the <i>UI Widget</i>. • <i>Color</i>. Sets the background color of the widget. Not every <i>UI Widget</i> has a background color. Changes to this <i>Variable</i> will be effective on the widgets that have a background color.
Button	<i>Title</i> . The text on the <i>Button</i> .
Label	<i>Text</i> . The text on the <i>Label</i> .
Switch	<i>On at start</i> . Whether the <i>Switch</i> should be turned on when the <i>Happ</i> starts execution.
Slider	<ul style="list-style-type: none"> • <i>Value</i>. The value that should be assigned to the <i>Slider</i> initially. • <i>Min</i>. The minimum allowed value. • <i>Max</i>. The maximum allowed value.
Touchpad	<ul style="list-style-type: none"> • <i>X-Multiplier</i>. Constant used to scale the displacement values along the x-axis delivered by the <i>Touchpad</i>. • <i>Y-Multiplier</i>. Constant used to scale the displacement values along the y-axis delivered by the <i>Touchpad</i>.

Table A.11: *Variables of UI Widgets* (Part 1/2).

Class	Variables
Music Player	<ul style="list-style-type: none"> • <i>Play button</i>. Whether the <i>Music Player</i> should show the play button. • <i>Next button</i>. Whether the <i>Music Player</i> should show a button to play next song in the playlist. • <i>Previous button</i>. Whether the <i>Music Player</i> should show a button to play previous song in the playlist. • <i>Volume view</i>. Whether the <i>Music Player</i> should show the volume slider. • <i>Visible</i>. Whether the <i>Music Player</i> should be visible in the mobile device's interface.
Contact Book	<ul style="list-style-type: none"> • <i>Call button</i>. Whether the <i>Contact Book</i> should show the call button. • <i>Next button</i>. Whether the <i>Contact Book</i> should show a button to iterate to the next contact. • <i>Previous button</i>. Whether the <i>Contact Book</i> should show a button to iterate to the previous contact.
Image View	<i>Image</i> . Opens a pop-up for selection of the image among the user's mobile phone image albums.
Monitor	<ul style="list-style-type: none"> • <i>X-Axis</i>. Whether the <i>Monitor</i> should display the horizontal axis. • <i>Y-Axis</i>. Whether the <i>Monitor</i> should display the vertical axis.

Table A.12: *Variables of UI Widgets (Part 2/2)*.

Class	Variables
Hardware Device	<ul style="list-style-type: none"> • <i>Name</i>. Enables developers to provide a name to the <i>Hardware Device</i> to identify it more easily during development. The name will be displayed in a label below the <i>Editable Object</i>'s image in the canvas. • <i>Autoroute</i>. If there is a single <i>Microcontroller</i> added to the canvas, this button causes the <i>Hardware Device</i> to be wired to any compatible available pins of the <i>Microcontroller</i>.
Button	Exposes no <i>Variables</i> .
Switch	Exposes no <i>Variables</i> .
Light Sensor	Exposes no <i>Variables</i> .
Temperature Sensor	Exposes no <i>Variables</i> .
Accelerometer	Exposes no <i>Variables</i> .
LSM Compass	Exposes no <i>Variables</i> .
MPU 6050	Exposes no <i>Variables</i> .
Potentiometer	<ul style="list-style-type: none"> • <i>Notify strategy</i>. Determines how the Potentiometer notifies other Objects about new values being available. Can be set to either <i>Always</i> - which causes the <i>Potentiometer</i> to always emit events when new values are available -, <i>inRange</i> - which causes the <i>Potentiometer</i> to emit events only when the values lie between a specific range - and <i>Once</i> - which causes the <i>Potentiometer</i> to emit an event when the value enters a specified range. • <i>Min notify value</i>. The minimum value in the range used by the notification strategies <i>inRange</i> and <i>Once</i>. • <i>Max notify value</i>. The maximum value in the range used by the notification strategies <i>inRange</i> and <i>Once</i>.
Proximity Sensor	

Table A.13: *Variables of Hardware Devices (Part 1/2)*.

Class	Variables
LED	<ul style="list-style-type: none"> • <i>Intensity</i>. The initial intensity of the <i>LED</i>. • <i>On at start</i>. Whether the <i>LED</i> should be turned on when the <i>Happ</i> begins execution.
Buzzer	<ul style="list-style-type: none"> • <i>Frequency</i>. The initial frequency of the <i>Buzzer</i>. • <i>On at start</i>. Whether the <i>Buzzer</i> should be turned on when the <i>Happ</i> begins execution.
Three-Color LED	<ul style="list-style-type: none"> • <i>Red</i>. The <i>Three-Color LEDs</i> initial red component intensity. • <i>Green</i>. The <i>Three-Color LEDs</i> initial green component intensity. • <i>Blue</i>. The <i>Three-Color LEDs</i> initial blue component intensity.
Vibration Board	<i>Frequency</i> . The <i>Vibration Board</i> 's initial frequency.

Table A.14: *Variables of Hardware Devices (Part 2/2)*.

Class	Variables
Number	<i>Value.</i> Text box to input the <i>Number</i> 's initial value.
Boolean	<i>Value.</i> Switch to input the <i>Boolean</i> 's initial value.
String	<i>Value.</i> Text box to input the <i>String</i> 's initial value.
Window	<ul style="list-style-type: none"> • <i>Size.</i> The amount of samples the <i>Window</i> should buffer. • <i>Overlapping.</i> The amount of samples in the current buffer that were also present in the previous buffer.
Low-Pass Filter	<i>Factor.</i> Sets the intensity of the <i>Filter</i> . A factor of 0 leaves the signal unmodified. The bigger the <i>factor</i> , the smoother the signal becomes.
High-Pass Filter	The <i>High-Pass Filter</i> has the same <i>Variables</i> as the <i>Low-Pass Filter</i> .
Mean Extractor	Exposes no <i>Variables</i> .
Deviation Extractor	Exposes no <i>Variables</i> .
Peak Detector	<ul style="list-style-type: none"> • <i>Min.</i> Sets the minimum value of the range in which peaks are searched for. • <i>Max.</i> Sets the maximum value of the range in which peaks are searched for. • <i>Upper peak.</i> Defines whether the <i>Peak Detector</i> should search for upper or lower peaks.
Activity Classifier	Exposes no <i>Variables</i> .
Posture Classifier	Exposes no <i>Variables</i> .

Table A.15: *Variables of Programming Objects (Part 1/2).*

Class	Variables
Timer	<ul style="list-style-type: none"> • <i>Frequency</i>. The trigger frequency of the <i>Timer</i> in seconds. • <i>Time Elapsed</i>. The amount of seconds since the timer has been started (0 if the timer has not been started). • <i>Repeats</i>. Whether the Timer should trigger once or indefinitely until stopped.
Sound	<i>Sound</i> . Opens a popup for selection of a sound file.
Recorder	Exposes no <i>Variables</i> .
Mapper	<ul style="list-style-type: none"> • <i>Input Min</i>. Defines the minimum value of the input range. • <i>Input Max</i>. Defines the maximum value of the input range. • <i>Output Min</i>. Defines the minimum value of the input range. • <i>Output Max</i>. Defines the maximum value of the input range. <p>Based on these four <i>Variables</i>, the <i>Mapper</i> defines a linear function which it uses to scale values in the input range to values in the output range.</p>

Table A.16: *Variables of Programming Objects (Part 2/2)*.

A.4 Simulation Objects

Simulation Objects provide an interface for users to modify the state of *Happ Objects* and display runtime information useful for debugging purposes. Tables A.17, A.18 and A.19 list what each *Simulation Object* subclass displays and how it reacts to user input. It should be noted that *UI Widgets Simulation Objects* behave in an identical way as the *UI Widget*.

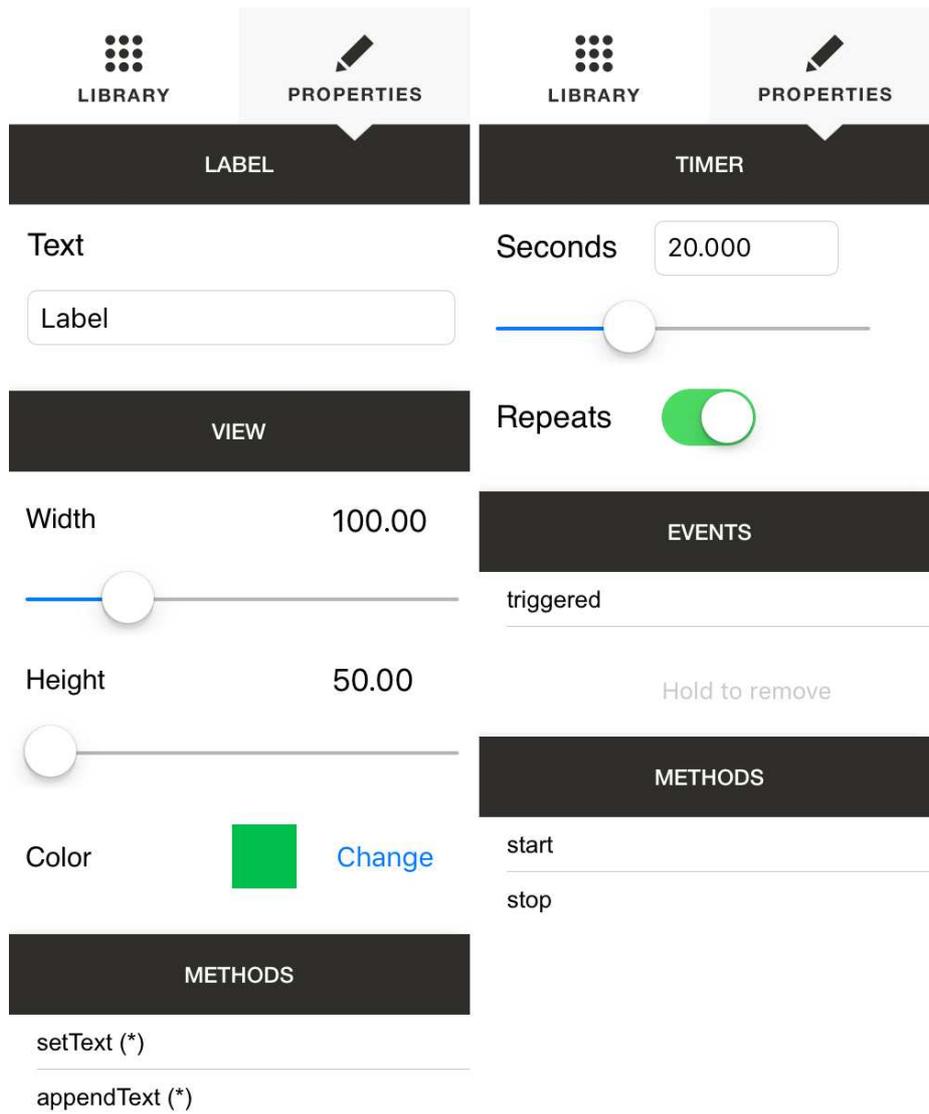


Figure A.3: Configuration View of a Label and Timer.

Simulation Object	Input	Output
Button	Finger touches on the <i>Button Simulation Object</i> cause the <i>Button</i> to be pressed/released.	Pressing and releasing the button produce a sound effect.
Switch	Taps on the <i>Switch Simulation Object</i> cause the <i>Switch</i> to be turned on/off.	<ul style="list-style-type: none"> • Pressing and releasing the button produce a sound effect. • A different image is shown displaying the switch turned on or off.
Light Sensor	Finger touches on the <i>Light Sensor Simulation Object</i> cause the light intensity measured by the <i>Light Sensor</i> to increase. The temperature starts decreasing automatically when the user is not touching the <i>Light Sensor Simulation Object</i> .	<ul style="list-style-type: none"> • A sprite of a light beam turns brighter or darker depending on the simulated light intensity. • A label is shown below the element containing the current light intensity value.
Temperature Sensor	Clockwise rotation gestures around the <i>Temperature Sensor Simulation Object</i> cause the temperature measured by the object to increase. Counter-clockwise rotation gestures cause the simulated temperature to decrease.	<ul style="list-style-type: none"> • An image of the thermometer with a moving arrow sprite indicate current simulated temperature. • A label is shown below the element containing the current temperature value.

Table A.17: Subclasses of *Sensor Simulation Object*.

Simulation Object	Input	Output
Accelerometer	The <i>Accelerometer Simulation Object</i> reacts to the acceleration measured by the accelerometer integrated in the tablet device.	A ball is shown around a circle frame that moves in the direction of the acceleration.
LSM Compass	The <i>LSM Compass Simulation Object</i> reacts to the acceleration and compass heading measured by the accelerometer and magnetometer integrated in the tablet device.	Displays the same output as the <i>Accelerometer</i> and an additional ring representing an analog compass labelled with four coordinates (North, South, West, East) that rotates to match the coordinates as the user rotates the tablet.
MPU 6050	The <i>MPU 6050 Simulation Object</i> reacts to the acceleration and rotation measured by the accelerometer and gyroscope integrated in the tablet device.	Displays the same output as the <i>Accelerometer</i> .
Potentiometer	The <i>Potentiometer Simulation Object</i> offers a knob which users can drag to simulate different values.	<ul style="list-style-type: none"> • The <i>Potentiometer</i>'s simulated value can be read implicitly from the position of the knob. • Additionally, as with the <i>Light Sensor</i> and <i>Temperature Sensor</i>, a label is shown below the element containing the current temperature value.

Table A.18: Subclasses of *Sensor Simulation Object*.

Simulation Object	Input	Output
LED	A single tap gesture on the <i>LED Simulation Object</i> turns the <i>LED</i> on/off.	Displays a sprite of a light beam which gets brighter or darker depending on the intensity of the LED. The sprite is hidden if the LED is turned off.
Buzzer	A single tap gesture on the <i>Buzzer Simulation Object</i> turns it on/off.	<ul style="list-style-type: none"> • The <i>Buzzer Simulation Object</i> plays a shaking animation when it is turned on. • A sound at the same frequency as the <i>Buzzer</i> is played.
Three-Color LED	A single tap gesture on the <i>Three-Color LED Simulation Object</i> turns it on/off.	Displays a sprite of a light beam that changes color depending on the current simulated color configuration of the <i>Three-Color LED Simulation Object</i> . The sprite is hidden if the LED is turned off.
Vibration Board	A single tap gesture on the <i>Vibration Board Simulation Object</i> turns it on/off.	The <i>Vibration Board</i> plays a shaking animation when it is turned on.

Table A.19: Subclasses of *Output Device Simulation Object*.

Class	Input	Output
Number	Accepts no user input.	Displays a label with the currently stored value.
Boolean	Accepts no user input.	Displays a label with the currently stored value.
String	Accepts no user input.	Displays a label with the currently stored value.
Window	Accepts no user input.	Displays no output.
Low-Pass Filter	Accepts no user input.	Displays no output.
High-Pass Filter	Accepts no user input.	Displays no output.
Mean Extractor	Accepts no user input.	Displays a label with the current mean.
Deviation Extractor	Accepts no user input.	Displays a label with the current deviation.
Peak Detector	Accepts no user input.	Displays no output.
Activity Classifier	Accepts no user input.	Displays an animation of a person walking, running, standing or climbing.
Posture Classifier	Accepts no user input.	Displays an image of a person standing, lying down on stomach or lying down on the back.

Table A.20: Subclasses of *Programming Simulation Objects*.

Class	Input	Output
Timer	A single tap on the <i>Timer Simulation Object</i> causes the <i>Timer</i> countdown to reset.	Displays a label with the current countdown time.
Sound	Accepts no user input.	The sound is played.
Recorder	Accepts no user input.	The recorder's image changes to indicate whether it is recording or not.
Mapper	Accepts no user input.	Displays no output.

Table A.21: Subclasses of *Programming Simulation Objects*.

A.5 TextIT Objects

In this section we describe every *TextIT* object. *Control Objects* are shown in Tables A.22 and A.23. *Visualization Objects* are shown in Tables A.24, A.25, A.26 and A.27.

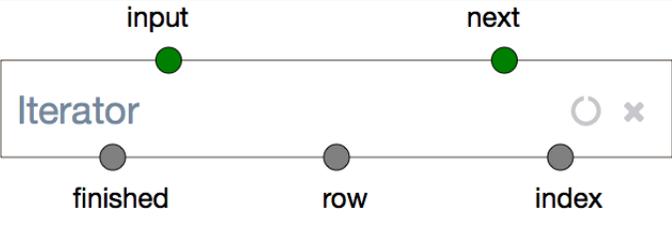
Image	Name	Description
	<i>Iterator</i>	Receives input data on “input” port. Events triggered on the “next” port cause the <i>Iterator</i> to process the next element in the input array.
	<i>Delayer</i>	In this case, the <i>Delayer</i> will trigger a signal every 50 milliseconds.
	<i>Library Manager</i>	Clicking on the <i>Edit</i> button leads to a file selection window to choose the data file from the filesystem.

Table A.23: *TextIT*’s *Iterator*, *Delayer* and *Library Manager* Control Objects.

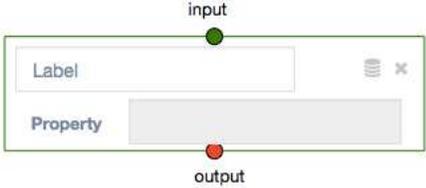
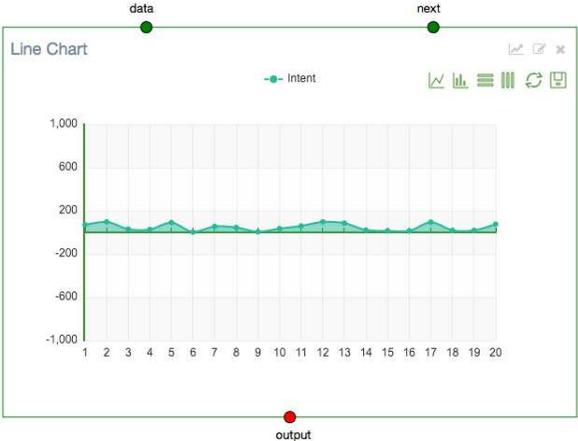
Image	Name	Description
	<i>Label</i>	The <i>Label</i> displays the value provided through its input port.
	<i>Line Chart</i>	The <i>Line Chart</i> displays a series of values as lines or bars. Line series can be defined from the element's <i>Edit Window</i> . Values provided through the input port have to match the amount of line series defined. The <i>Line Chart</i> has additional buttons to show / hide the series, save a screenshot of the chart and to switch between line or bar visualization styles.
	<i>Pie Chart</i>	The <i>Pie Chart</i> displays values as a pie chart. Features similar configurable functionality as the <i>Line Chart</i> to show / hide the series, save the chart's current state in a screenshot, and to switch between pie and funnel chart visualization styles.

Table A.24: *TextIT's Label, Line Chart and Pie Chart Visualization Objects.*

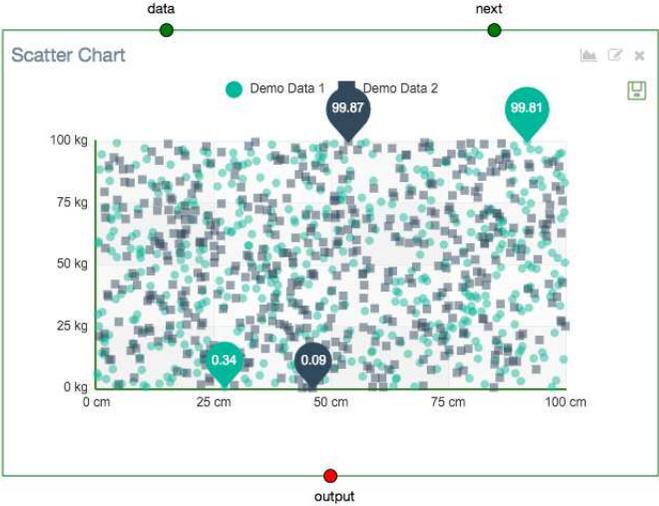
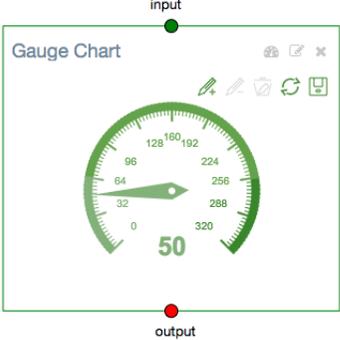
Image	Name	Description
	<p><i>Scatter Chart</i></p>	<p>Displays points in a 2D Cartesian coordinate system. Values of every point can be seen when hovering the mouse over it. Series and axis labels can be configured over the <i>Edit Window</i>.</p>
	<p><i>Gauge Chart</i></p>	<p>The <i>Gauge Chart</i> is similar to a label in that it displays a single value. However, the <i>Gauge Chart</i> displays values by animating a needle moving clockwise.</p>

Table A.25: *TextIT's Scatter Chart and Gauge Chart Visualization Objects.*

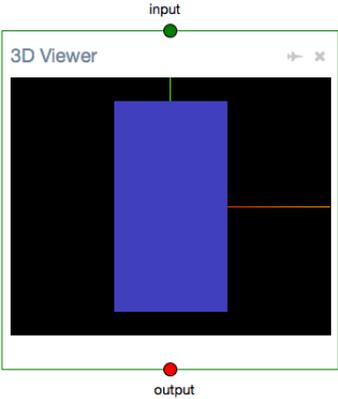
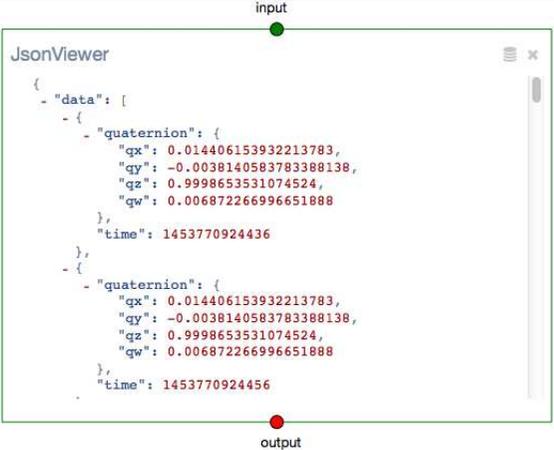
Image	Name	Description
	<i>3D Viewer</i>	<p>The <i>3D Viewer</i> renders a 3D scene containing an object rotated according to the quaternion provided through its input port. If fed with series of quaternions, it will animate the rotation of the 3D object.</p>
	<i>JSON Viewer</i>	<p>The JSON Viewer displays a JSON file in a formatted manner. The <i>JSON Viewer</i> does not enable editing the JSON files.</p>

Table A.26: *TextIT's 3D Viewer and JSON Viewer Visualization Objects.*

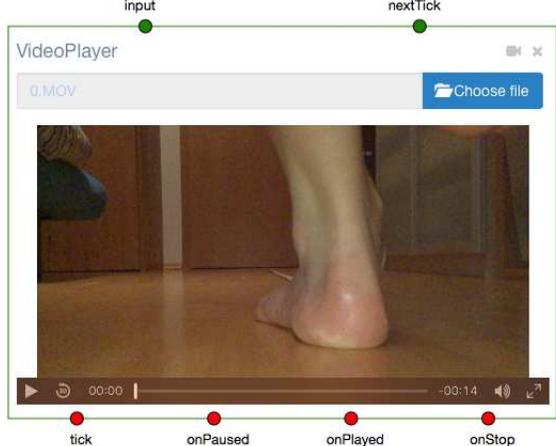
Image	Name	Description
	<p><i>Table Viewer</i></p>	<p>The <i>Table Viewer</i> is similar to the <i>JSON Viewer</i>. The <i>Table Viewer</i> displays a dictionary provided through its input port as a table.</p>
	<p><i>Video Player</i></p>	<p>As its name indicates, the <i>Video Player</i> plays video files. Video files are added over the <i>Edit Window</i>. Volume and screen mode (i.e. normal, full screen) are configurable.</p>

Table A.27: *TextIT's Table Viewer and Video Player Visualization Objects.*

A.5.1 Code Editor

A *TextIT* object with particular importance is the *Code Editor*. The *Code Editor*'s *Edit Window* is divided in left and right frames. Developers write JavaScript source code on the left and visualize execution results on the right side. Syntax errors and compile warnings (e.g. unused *Variables*) are displayed next to the source code line. When the source code in the *Code Editor* is executed, the *Code Editor* creates its output ports dynamically. A screenshot of the *Code Editor* is shown in Figure A.4.

Code Editor

```

1 //JavaScript syntax
2 var foo = 2;
3
4 //use JavaScript functions
5 var pow = Math.pow(foo,3);
6
7 //injected dependency: FeatureExtractor.js
8 var euler = eulerToAngle(Math.PI);
9
10 //asynchronous execution
11 $.get("ajax/test.html", function(data) {
12     alert("Load successful.");
13 });
14
15 //return statements are shown on the right panel
16 return {
17     foo:foo,
18     pow: pow,
19     euler: euler
20 };
21 |
22 //compile warning
23 ⚠ var something = 3;
24
25 //compile error
26 ✖ errors are shown as a red cross
27

```

```

{
  "foo": 2,
  "pow": 8,
  "euler": 180
}

```

Missing ";" before statement
Missing semicolon.

Hide Visualisation Close Save and Execute

Figure A.4: *TextIT's Code Editor*. The window is divided in two views. Code is shown at the left and execution results are shown on the right view. *Compile* warnings are shown as exclamation marks and errors are shown as red crosses on the left margin.

A.5.2 Line Chart Displaying Jogging Signal

Figure A.5 displays a *TextIT* project that displays linear acceleration data recorded while a user was jogging. When the *Data Loader*'s *output* port is connected to the *Line Chart*'s *data* port, the *Line Chart* renders the signal in a line chart plot.

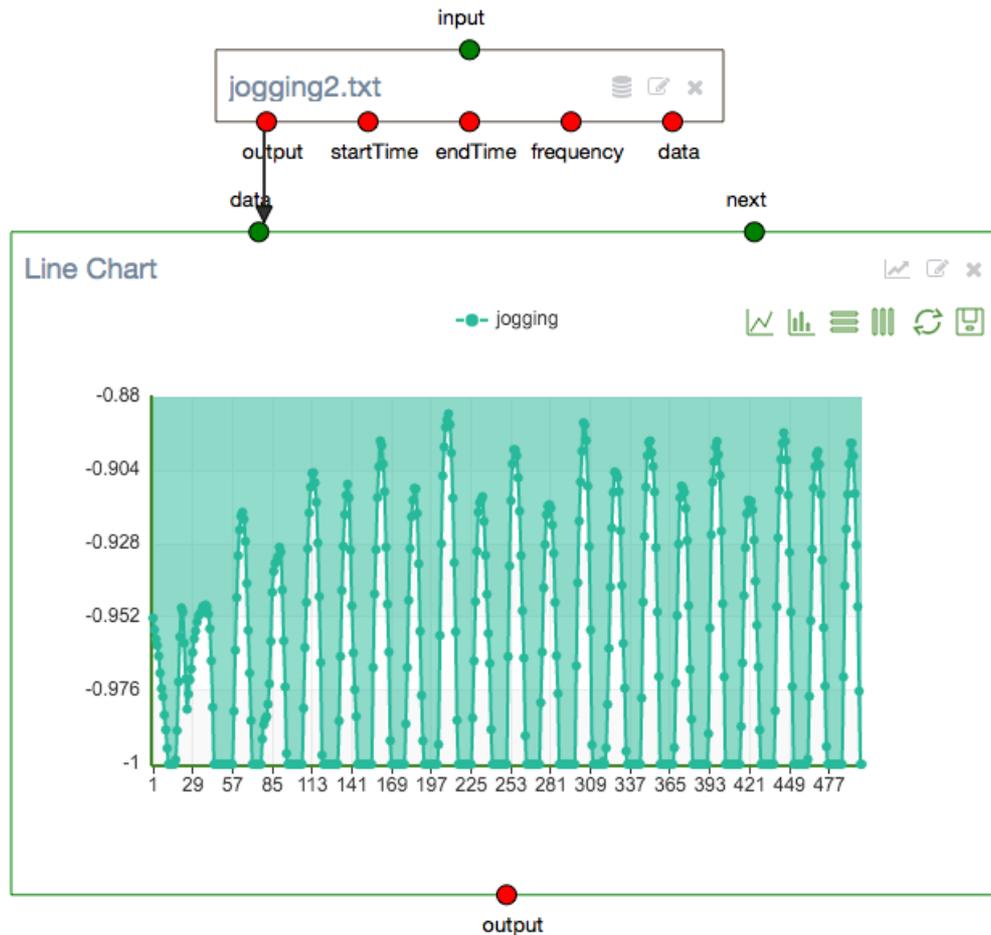


Figure A.5: Jogging signal displayed by a *Line Chart View* in *TextIT*.

A.5.3 TextIT Plugin for Counting Peaks in a Signal

This section presents the pseudo-code used to count the number of peaks in a signal. This pseudo-code is used by the KneeHapp bandage to count the number of side hops a patient performs in a specified period of time. Section 8.1.3 describes the use case and the usage of this pseudo-code in more detail.

APPENDIX A. HAPP OBJECTS

```
function peakCount = countPeaks(v, delta)
    minValue = Inf;
    maxValue = -Inf;
    minIdx = NaN;
    maxIdx = NaN;
    lookForMax = 1;

    for i = 1 : length(v)
        currentSample = v(i);
        if currentSample > maxValue
            maxValue = currentSample;
            maxIdx = i;
        end

        if currentSample < minValue
            minValue = currentSample;
            minIdx = i;
        end

        if lookForMax
            if currentSample < maxValue - delta
                peakCount ++
                minValue = currentSample;
                minIdx = i;
                lookForMax = 0;
            end
        else
            if currentSample > minValue + delta
                maxValue = currentSample;
                maxIdx = i;
                lookForMax = 1;
            end
        end
    end
end
```

List of Figures

1.1	a) In the <i>summative</i> research approach the artifacts produced by a technology are assessed. b) In the <i>formative</i> research approach, the artifacts produced by a technology are used to assess and improve the technology.	3
2.1	a) Smart garments compared to other devices according to their integration with the human body, based on previous work from Steve Mann [75]. b) Wearable devices represented as a taxonomy.	11
2.2	Classification of wearable computers and smart textiles according to the degree of integration of the electronics into the textile.	11
2.3	a) Qing Dynasty’s abacus ring. Image taken from http://www.chinaculture.org . b) Eudaemonics’ shoe used to cheat at the casino. Image taken from: http://eyetap.org/wearcam/eudaemonic/ with permission of Steve Mann. c) Ivan Sutherland’s HMD [116]. Communications of the ACM 2002, Vol. 11:2. Copyright ©2002 by ACM, Inc. Reprinted with permission of ACM, Inc.	12
2.4	a) Steve Mann’s HMD [73]. Reprinted with permission of Steve Mann. b) CMU’s VuMan 2 wearable computer [110]. Reprinted with permission of Daniel Siewiorek. c) Forms for wearability [37]. Reprinted with permission of Francine Gemperle.	13
2.5	a) Georgia Tech’s Wearable Motherboard (GTWM) [95]. Copyright ©2002 by ACM, Inc. Reprinted with permission of ACM, Inc. b) <i>Musical Jacket</i> [99]. Reprinted with permission of Maggie Orth.	15
2.6	Textile fabrication process.	16
2.7	Different types of conductive yarns. (a) A copper foil wrapped around a non-conductive yarn. Reprinted with permission of Maggie Orth of <i>International Fashion Machines</i> . (b) A wire wrapped around non-conductive fiber strands [114]. Licensed under <i>Creative Commons BY 4.0</i> . (c) Multifilament yarn coated with a metallic layer [114]. Licensed under <i>Creative Commons BY 4.0</i>	17

LIST OF FIGURES

2.8	Electronic devices attached to fabric. (a) Flexible electronic module glued to the fabric and connected with embroidered conductive lines developed by the <i>Fraunhofer's Institute for Reliability and Microintegration</i> [71]. (b) Microcontroller connected with conductive thread to conductive snap buttons crimped. Developed in collaboration between the <i>Technische Universität München</i> and the <i>Universität der Künste</i> [45].	18
2.9	Model of a smart textile.	19
2.10	Taxonomy of textile sensors.	21
2.11	Taxonomy of output devices.	23
2.12	Textile output devices: a) Textile thermochromic display and b) textile speaker. Reprinted with permission of Katharina Bredies.	23
2.13	Taxonomy of connections used in smart textiles.	24
3.1	Main components of an IDE for smart textiles.	36
3.2	Model of a <i>Smart Textile</i>	37
3.3	Model of the <i>Editor</i>	38
3.4	Model of the <i>Simulator</i>	39
3.5	Model of the <i>Deployer</i>	40
4.1	Main classes of the <i>Running Engine</i>	42
4.2	Model of the <i>Happ Object Library</i>	43
5.1	Software architecture of the <i>THIDE</i> based on the layered and repository architectural styles. The Happ subsystem is the repository. Software components in the <i>TangoHapps IDE</i> and <i>TangoHapps Framework</i> access the Happ. Accesses to the Happ are represented with dashed lines.	53
5.2	Deployment of software components in <i>THIDE</i>	55
6.1	Overview of the <i>Editor</i>	57
6.2	Main classes involved in the <i>Palette</i>	58
6.3	Main classes involved in the <i>Toolbar</i>	60
6.4	Main classes involved in the <i>Canvas</i>	60
6.5	Main classes involved in the <i>Editor's</i> functionality to create circuit layouts.	62
6.6	Layout (a) and object diagram (b) of an electric circuit that consists of a <i>Button</i> connected to an <i>Arduino Lilypad</i>	63
6.7	Main classes in the <i>Simulator</i>	64
6.8	Usage of the Decorator pattern in the <i>Simulator</i>	65
6.9	Main classes of the <i>Deployer</i>	66
6.10	Main classes in the <i>Plugin Editor</i>	67
6.11	Main classes of the <i>Plugin Interpreter</i>	70
6.12	Main classes of the <i>Firmata Library</i> . The <i>Virtual Communication Module</i> belongs to the <i>Simulator</i>	71
7.1	<i>Interactex Designer's Project Selection Screen</i>	74

7.2	<i>Interactex Designer</i> UI overview.	75
7.3	Invocations between <i>Events</i> of a <i>Button</i> and <i>Methods</i> of an <i>LED</i>	77
7.4	Circuit layout of the <i>CTS-Gauntlet</i> developed in <i>Interactex Designer</i>	78
7.5	Simulation of different objects in <i>Interactex Designer</i>	79
7.6	<i>Interactex Client's User Application Screen</i> (a) and <i>Default Application Screen</i> (b).	81
7.7	Overview of <i>TextIT's</i> user interface.	83
7.8	Visual components of a <i>TextIT Object</i>	84
7.9	<i>WaveCap</i> implemented in <i>Interactex Designer</i>	84
7.10	Development of an algorithm to detect jogging speed in <i>TextIT</i>	86
7.11	<i>TextIT</i> code to estimate jogging speed for the <i>WaveCap</i> application.	87
7.12	Usage of a <i>TextIT</i> plugin in <i>Interactex Designer</i> to control smartphone's volume based on estimated jogging speed.	87
7.13	Circuit layout of <i>WaveCap</i> developed in <i>Interactex Designer</i> . <i>Hardware Devices</i> from left to right: a <i>Textile Sensor</i> , a <i>BLE-Lilypad</i> and a <i>Textile Speaker</i> (renamed to "Radio Speaker").	88
8.1	a) KneeHapp Bandage. b) KneeHapp sock.	93
8.2	a) KneeHapp coordinate system. b) Illustration of the medial collapse during a squat. and how it affects the pitch.	94
8.3	One-leg hop linear acceleration on y and z-axis raw (a) and filtered (b).	96
8.4	One-leg hop average pressure.	96
8.5	Linear acceleration of seven side hops. raw (a) and filtered (b).	97
8.6	KneeHapp's circuit design created with <i>TangoHapps</i>	98
8.7	KneeHapp's <i>Range of Motion</i> use case implemented with <i>TangoHapps</i>	99
8.8	KneeHapp's functionality to produce auditive feedback when the user's knee deviates more than 4 degrees during a <i>One-Leg Squat</i>	101
8.9	KneeHapp's <i>One-Leg Squat</i> use case implemented with <i>TangoHapps</i>	102
8.10	KneeHapp's <i>One-Leg Hop</i> use case implemented with <i>TangoHapps</i>	104
8.11	KneeHapp's <i>Side Hops</i> use case implemented with <i>TangoHapps</i>	106
8.12	a) Custodian Jacket - fourth version. b) Backside of a rack in a super-computer center. Some height unit labels are hidden by cables.	110
8.13	Linear acceleration along y-axis while user is standing (a) and walking (b).	111
8.14	Classification of physical activities based on IMU data. Axes correspond to the deviation of the linear acceleration along the x, y and z-axes.	112
8.15	Circuit Design of the Custodian Jacket. a) Jacket's torso including <i>Buzzer</i> , <i>Textile Sensor</i> , <i>Button</i> , <i>MPU 6050</i> and <i>BLE-Lilypad</i> . b) Jacket's Sleeve including <i>MPU 6050</i> , <i>Proximity Sensor</i> , <i>Three-Color LED</i> and <i>BLE-Lilypad</i>	113
8.16	<i>TangoHapps</i> implementations of: a) Function that checks whether the proximity measured by a <i>Proximity Sensor</i> is in the range $35 \pm 2cm$. b) Function that sets the color of the <i>RGB LED</i> to red.	115
8.17	Sketch of a <i>TangoHapps</i> implementation of the Custodian's <i>Error Prevention</i> use case.	116

LIST OF FIGURES

8.18	Custodian's <i>Technician's Safety</i> use case implemented with <i>TangoHapps</i> .	118
8.19	Custodian's <i>Emergency Situation</i> use case implemented with <i>TangoHapps</i>	120
9.1	a) Three female subjects develop an electronic circuit with <i>Interactex Designer</i> . b) Application 1 consisting of 12 LEDs.	123
9.2	a) A paper prototype of a jacket developed during the Study 1. b) Textile testbed containing two accelerometers, a push button, switch, light sensor, buzzer, vibration motor and a three-color LED. Every electronic device is connected with conductive fabric to a custom-made Arduino Lilypad with an integrated BLE module (in the center).	125
A.1	<i>Palette Objects</i> for instantiating <i>Smart Textiles, Electronic Devices</i> (e.g. <i>Hardware Devices</i> and <i>Microcontrollers</i>) and <i>UI Widgets</i> available in <i>Interactex Designer</i>	142
A.2	<i>Palette Objects</i> for instantiating <i>Signal Processing Functions, Variables and Operators</i> available in <i>Interactex Designer</i>	143
A.3	<i>Configuration View</i> of a <i>Label</i> and <i>Timer</i>	151
A.4	<i>TextIT's Code Editor</i> . The window is divided in two views. Code is shown at the left and execution results are shown on the right view. <i>Compile</i> warnings are shown as exclamation marks and errors are shown as red crosses on the left margin.	164
A.5	Jogging signal displayed by a <i>Line Chart View</i> in <i>TextIT</i>	165

List of Tables

1.1	Smart textiles used to elicit the initial requirements of <i>TangoHapps</i> . . .	5
7.1	<i>Toolbar Buttons</i> in <i>Interactex Designer</i>	76
7.2	Images of <i>Hooks</i> in <i>Interactex Designer</i> . Filled shapes indicate that a parameter provided by an <i>Event</i> comply those expected by a <i>Method</i> . Hollow shapes indicate that the <i>Event</i> does not provide a parameter of the type expected by the <i>Method</i>	77
A.1	<i>UI Widgets</i> supported by <i>TangoHapps</i>	132
A.2	<i>Sensors</i> supported by <i>TangoHapps</i>	133
A.3	<i>Output Devices</i> supported by <i>TangoHapps</i>	134
A.4	<i>Microcontrollers</i> supported by <i>TangoHapps</i>	135
A.5	<i>Variables</i> supported by <i>TangoHapps</i>	135
A.6	<i>Comparison Operators</i> supported by <i>TangoHapps</i>	136
A.7	<i>Logical Operators</i> supported by <i>TangoHapps</i>	137
A.8	<i>Arithmetic Operators</i> supported by <i>TangoHapps</i>	138
A.9	Signal processing and classification <i>Functions</i> supported by <i>TangoHapps</i> .	139
A.10	Utility programming elements supported by <i>TangoHapps</i>	140
A.11	<i>Variables of UI Widgets</i> (Part 1/2).	144
A.12	<i>Variables of UI Widgets</i> (Part 2/2).	145
A.13	<i>Variables of Hardware Devices</i> (Part 1/2).	146
A.14	<i>Variables of Hardware Devices</i> (Part 2/2).	147
A.15	<i>Variables of Programming Objects</i> (Part 1/2).	148
A.16	<i>Variables of Programming Objects</i> (Part 2/2).	149
A.17	Subclasses of <i>Sensor Simulation Object</i>	152
A.18	Subclasses of <i>Sensor Simulation Object</i>	153
A.19	Subclasses of <i>Output Device Simulation Object</i>	154
A.20	Subclasses of <i>Programming Simulation Objects</i>	155
A.21	Subclasses of <i>Programming Simulation Objects</i>	155
A.22	<i>TextIT's Control Objects</i>	157
A.23	<i>TextIT's Iterator, Delayer and Library Manager Control Objects</i>	158
A.24	<i>TextIT's Label, Line Chart and Pie Chart Visualization Objects</i>	159
A.25	<i>TextIT's Scatter Chart and Gauge Chart Visualization Objects</i>	160
A.26	<i>TextIT's 3D Viewer and JSON Viewer Visualization Objects</i>	161
A.27	<i>TextIT's Table Viewer and Video Player Visualization Objects</i>	162

LIST OF TABLES

Bibliography

- [1] S. C. Anand, J. F. Kennedy, M. Miraftab, and S. Rajendran. *Medical textiles and biomaterials for healthcare*. Elsevier, 2005.
- [2] S. Ananthanarayan, M. Sheh, A. Chien, H. Profita, and K. Siek. Pt Viz: Towards a Wearable Device for Visualizing Knee Rehabilitation Exercises. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1247–1250, New York, NY, USA, 2013. ACM.
- [3] F. Axisa, P. M. Schmitt, C. Gehin, G. Delhomme, E. McAdams, and A. Dittmar. Flexible technologies and smart clothing for citizen medicine, home health-care, and disease prevention. *IEEE transactions on information technology in biomedicine*, 9(3):325–336, 2005.
- [4] M. Ayoade and L. Baillie. A Novel Knee Rehabilitation System for the Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2521–2530, New York, NY, USA, 2014. ACM.
- [5] R. T. Azuma and Others. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.
- [6] S. K. Bahadir, V. Koncar, and F. Kalaoglu. Wearable obstacle detection system fully integrated to textile structures for visually impaired people. *Sensors and Actuators A: Physical*, 179:297–311, 2012.
- [7] R. Ballagas, F. Memon, R. Reiners, and J. Borchers. iStuff mobile: rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1107–1116. ACM, 2007.
- [8] R. Ballagas, M. Ringel, M. Stone, and J. Borchers. iStuff: a physical user interface toolkit for ubiquitous computing environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 537–544. ACM, 2003.
- [9] D. Bannach, P. Lukowicz, and O. Amft. Rapid prototyping of activity recognition applications. *Pervasive Computing, IEEE*, 7(2):22–31, 2008.
- [10] W. Barfield and T. Caudell. *Fundamentals of wearable computers and augmented reality*. CRC Press, 2001.

BIBLIOGRAPHY

- [11] S. Bielska, M. Sibinski, and A. Lukasik. Polymer temperature sensor for textronic applications. *Materials Science and Engineering: B*, 165(1):50–52, 2009.
- [12] F. Block, M. Haller, H. Gellersen, C. Gutwin, and M. Billinghurst. VoodooSketch: extending interactive surfaces with adaptable interface palettes. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 55–58. ACM, 2008.
- [13] S. Brady, D. Diamond, and K.-T. Lau. Inherently conducting polymer modified polyurethane smart foam for pressure sensing. *Sensors and Actuators A: Physical*, 119(2):398–404, 2005.
- [14] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java (3rd Edition)*. Pearson Education, aug 2009.
- [15] L. Buechley. A construction kit for electronic textiles. In *10th IEEE International Symposium on Wearable Computers, 2006*, pages 83–90. IEEE, 2006.
- [16] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-oriented software architecture: a system of patterns. 1996. *Part II*, 2001.
- [17] F. Carpi and D. De Rossi. Electroactive polymer-based devices for e-textiles in biomedicine. *IEEE Transactions on Information Technology in Biomedicine*, 9(3):295–318, sep 2005.
- [18] L. M. Castano and A. B. Flatau. Smart fabric sensors and e-textile technologies: a review. *Smart Materials and Structures*, 23(5):53001, 2014.
- [19] M.-H. Cheng, L.-C. Chen, Y.-C. Hung, and C. M. Yang. A real-time maximum-likelihood heart-rate estimator for wearable textile sensors. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, pages 254–257. IEEE, 2008.
- [20] K. Cherenack and L. van Pieterse. Smart textiles: Challenges and opportunities. *Journal of Applied Physics*, 112(9):91301, 2012.
- [21] S. Coyle, F. Benito-Lopez, T. Radu, K. Lau, and D. Diamond. Fibers and fabrics for chemical and biological sensing. *Research Journal of Textile and Apparel*, 14(4):64–72, 2012.
- [22] D. Curone, E. L. Secco, A. Tognetti, G. Loriga, G. Dudnik, M. Risatti, R. Whyte, A. Bonfiglio, and G. Magenes. Smart Garments for Emergency Operators: The ProeTEX Project. *IEEE Transactions on Information Technology in Biomedicine*, 14(3):694–701, may 2010.
- [23] S. Curry. An introduction to the Java Ring. *Java World, April*, 1998.
- [24] R. Daude and M. Weck. Mobile approach support system for future machine tools. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 24–30, oct 1997.

- [25] A. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. a CAPpella: programming by demonstration of context-aware applications. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 6(1):40, 2004.
- [26] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. a CAPpella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–40. ACM, 2004.
- [27] M. Di Rienzo, F. Rizzo, G. Parati, G. Brambilla, M. Ferratini, and P. Castiglioni. MagIC System: a New Textile-Based Wearable Device for Biological Signal Monitoring. Applicability in Daily Life and Clinical Setting. *Conference proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 7:7167–7169, 2005.
- [28] B. Ding, M. Wang, J. Yu, and G. Sun. Gas sensors based on electrospun nanofibers. *Sensors*, 9(3):1609–1624, 2009.
- [29] P. Dragicevic and J.-D. Fekete. Support for input adaptability in the ICON toolkit. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 212–219. ACM, 2004.
- [30] P. Dragicevic and J.-D. Fekete. The input configurator toolkit: towards high input adaptability in interactive applications. In *Proceedings of the working conference on Advanced visual interfaces*, pages 244–247. ACM, 2004.
- [31] B. Eskofier, M. Oleson, C. DiBenedetto, and J. Hornegger. Embedded surface classification in digital sports. *Pattern Recognition Letters*, 30(16):1448–1456, 2009.
- [32] J. Favre, R. Aissaoui, B. Jolles, J. de Guise, and K. Aminian. Functional calibration procedure for 3D knee joint angle description using inertial sensors. *Journal of Biomechanics*, 42(14):2330–2335, oct 2009.
- [33] J. Favre, B. M. Jolles, R. Aissaoui, and K. Aminian. Ambulatory measurement of 3D knee joint angle. *Journal of biomechanics*, 41(5):1029–1035, 2008.
- [34] S. Feiner, B. MacIntyre, T. Hollerer, and A. Webster. A touring machine: prototyping 3D mobile augmented reality systems for exploring the urban environment. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 74–81, oct 1997.
- [35] F. H. Fu, C. H. Bennett, C. Lattermann, and C. B. Ma. Current trends in anterior cruciate ligament reconstruction part 1: biology and biomechanics of reconstruction. *The American Journal of Sports Medicine*, 27(6):821–830, 1999.
- [36] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

BIBLIOGRAPHY

- [37] F. Gemperle, C. Kasabach, J. Stivoric, M. Bauer, and R. Martin. Design for wearability. In *Second International Symposium on Wearable Computers, 1998.*, pages 116–122. IEEE, 1998.
- [38] G. Gioberto, C.-H. Min, C. Compton, and L. E. Dunne. Lower-limb Goniometry Using Stitched Sensors: Effects of Manufacturing and Wear Variables. In *Proceedings of the 2014 ACM International Symposium on Wearable Computers, ISWC '14*, pages 131–132, New York, NY, USA, 2014. ACM.
- [39] S. Greenberg and C. Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 209–218. ACM, 2001.
- [40] L. Y. Griffin, J. Agel, M. J. Albohm, E. A. Arendt, R. W. Dick, W. E. Garrett, J. G. Garrick, T. E. Hewett, L. Huston, M. L. Ireland, and Others. Noncontact anterior cruciate ligament injuries: risk factors and prevention strategies. *Journal of the American Academy of Orthopaedic Surgeons*, 8(3):141–150, 2000.
- [41] P. Grossman. The LifeShirt: a multi-function ambulatory system monitoring health, disease, and medical intervention in the real world. *Stud Health Technol Inform*, 108:133–141, 2004.
- [42] J. Habetha. The MyHeart project-fighting cardiovascular diseases by prevention and early diagnosis. In *Proceedings of the IEEE Engineering in Medicine and Biology Society*, pages 6746–6749. Citeseer, 2006.
- [43] J. Haladjian. TangoHapps: an integrated development environment for smart garments. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 471–476. ACM, 2015.
- [44] J. Haladjian, K. Bredies, and B. Bruegge. Interactex: An integrated development environment for smart textiles. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ACM, 2016.
- [45] J. Haladjian, Z. Hodaie, H. Xu, M. Yigin, B. Bruegge, M. Fink, and J. Hoehner. KneeHapp: a bandage for rehabilitation of knee injuries. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 181–184. ACM, 2015.
- [46] J. Haladjian, D. Ismailović, B. Köhler, and B. Bruegge. A quick prototyping framework for adaptive serious games with 2D physics on mobile touch devices. In *IADIS Mobile Learning 2012 (ML 2012)*, Berlin, 2012.
- [47] J. Haladjian, D. Richter, P. Muntean, D. Ismailović, and B. Brügge. A framework for the creation of mobile educational games for dyslexic children. In *ML 2013 - Mobile Learning*, Lisbon, Portugal, mar 2013.

- [48] J. Haladjian, F. Ziegler, B. Simeonova, B. Köhler, P. Muntean, D. Ismailović, and B. Brüggel. A framework for game tuning. In *GET 2012 - IADIS Game and Entertainment Technologies*, Lisbon, Portugal, jul 2012.
- [49] H. Harms, O. Amft, D. Roggen, and G. Tröster. Rapid prototyping of smart garments for activity-aware applications. *Journal of Ambient Intelligence and Smart Environments*, 1(2):87–101, 2009.
- [50] B. Hartmann, L. Abdulla, M. Mittal, and S. R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–154. ACM, 2007.
- [51] B. Hartmann, S. R. Klemmer, and M. Bernstein. d. tools: Integrated prototyping for physical interaction design. *IEEE Pervasive Computing*, 2005.
- [52] S. Hodges, N. Villar, N. Chen, T. Chugh, J. Qi, D. Nowacka, and Y. Kawahara. Circuit stickers: Peel-and-stick construction of interactive electronic prototypes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1743–1746. ACM, 2014.
- [53] T. Holleczeck, A. Rüegg, H. Harms, and G. Tröster. Textile Pressure Sensors for Sports Applications. In *Proceedings of the 9th IEEE Conference on Sensors (IEEE Sensors 2010)*, pages 732–737, nov 2010.
- [54] C.-T. Huang, C.-L. Shen, C.-F. Tang, and S.-H. Chang. A wearable yarn-based piezo-resistive sensor. *Sensors and Actuators A: Physical*, 141(2):396–403, 2008.
- [55] Y. Huang and M. Eisenberg. Plushbot: an application for the design of programmable, interactive stuffed toys. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*, pages 257–260. ACM, 2011.
- [56] S. E. Hudson and J. Mankoff. Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 289–298. ACM, 2006.
- [57] Z. Hui, T. X. Ming, Y. T. Xi, and L. X. Sheng. Pressure sensing fabric. In *MRS Proceedings*, volume 920, pages 0920—S05. Cambridge Univ Press, 2006.
- [58] J. A. Jacko. *Human computer interaction handbook: Fundamentals, evolving technologies, and emerging applications*. CRC press, 2012.
- [59] A. R. Kahn, J. D. Hixson, J. E. Puffer, and E. E. Bakken. Three-Years’ Clinical Experience with Radioisotope Powered Cardiac Pacemakers. *IEEE Transactions on Biomedical Engineering*, BME-20(5):326–331, sep 1973.

BIBLIOGRAPHY

- [60] E.-S. Katterfeldt, N. Dittert, and H. Schelhowe. EduWear: smart textiles as ways of relating computing technology to everyday life. In *Proceedings of the 8th International Conference on Interaction Design and Children*, pages 9–17. ACM, 2009.
- [61] B. Kaufmann and L. Buechley. Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 291–298. ACM, 2010.
- [62] S. R. Klemmer, J. Li, J. Lin, and J. A. Landay. Papier-M{â}ch{é}: Toolkit support for tangible interaction. In *Proceedings of the 16th annual ACM symposium on user interface software and technology (UIST 2003), Vancouver, British Columbia, Canada, 2003*.
- [63] S. R. Klemmer, J. Li, J. Lin, and J. A. Landay. Papier-Mache: toolkit support for tangible input. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 399–406. ACM, 2004.
- [64] S. Klepper, S. Krusche, S. Peters, B. Bruegge, and L. Alperowitz. Introducing Continuous Delivery of Mobile Apps in a Corporate Environment: A Case Study. In *Proceedings of the 2nd International Workshop on Rapid Continuous Software Engineering*, pages 5–11. IEEE/ACM, 2015.
- [65] J. Krumm. *Ubiquitous computing fundamentals*. CRC Press, 2009.
- [66] S. Krusche, L. Alperowitz, B. Bruegge, and M. Wagner. Rugby: An Agile Process Model Based on Continuous Delivery. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pages 42–50. ACM, 2014.
- [67] B. Larson, H. Elmqvist, L. Ryden, and H. Schüller. Lessons From the First Patient with an Implanted Pacemaker:. *Pacing and Clinical Electrophysiology*, 26(1p1):114–124, 2003.
- [68] J. C. Lee, D. Avrahami, S. E. Hudson, J. Forlizzi, P. H. Dietz, and D. Leigh. The calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 167–175. ACM, 2004.
- [69] Y. Li, J. I. Hong, and J. A. Landay. Topiary: a tool for prototyping location-enhanced applications. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 217–226. ACM, 2004.
- [70] Y. Li, M. Y. Leung, X. M. Tao, X. Y. Cheng, J. Tsang, and M. C. W. Yuen. Polypyrrole-coated conductive fabrics as a candidate for strain sensors. *Journal of materials science*, 40(15):4093–4095, 2005.

- [71] T. Linz, C. Kallmayer, R. Aschenbrenner, and H. Reichl. Embroidering electrical interconnects with conductive yarn for the integration of flexible electronic modules into fabric. In *null*, pages 86–91. IEEE, 2005.
- [72] S. Lysecky and F. Vahid. Enabling nonexpert construction of basic sensor-based systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(1):1, 2009.
- [73] S. Mann. An historical account of the 'WearComp' and 'WearCam' inventions developed for applications in 'personal imaging'. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 66–73, oct 1997.
- [74] S. Mann. Wearable computing: a first step toward personal imaging. *Computer*, 30(2):25–32, feb 1997.
- [75] S. Mann. Can Humans Being Clerks make Clerks be Human? Exploring the Fundamental Difference between UbiComp and WearComp. *it-Information Technology (vormals it+ ti) Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 43(2/2001):97, 2001.
- [76] D. Marculescu, R. Marculescu, N. H. Zamora, P. S. Marbell, P. K. Khosla, S. Park, S. Jayaraman, S. Jung, C. Lauterbach, W. Weber, and Others. Electronic textiles: A platform for pervasive computing. *Proceedings of the IEEE*, 91(12):1995–2018, 2003.
- [77] N. Marquardt and S. Greenberg. Distributed physical interfaces with shared phidgets. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 13–20. ACM, 2007.
- [78] T. Martin, M. Jones, J. Edmison, and R. Shenoy. Towards a design framework for wearable electronic textiles. *Seventh IEEE International Symposium on Wearable Computers, 2003. Proceedings*, 2003.
- [79] C. Mattmann, O. Amft, H. Harms, G. Tröster, and F. Clemens. Recognizing upper body postures using textile strain sensors. In *11th IEEE International Symposium on Wearable Computers, 2007*, pages 29–36. IEEE, 2007.
- [80] C. Mattmann, T. Kirstein, and G. Tröster. A method to measure elongations of clothing. In *International Scientific Conference Ambience05, Tampere, Finland*, page 18, 2005.
- [81] C. Mattmann and G. Troster. Design concept of clothing recognizing back postures. In *3rd IEEE/EMBS International Summer School on Medical Devices and Biosensors, 2006*, pages 24–27. IEEE, 2006.
- [82] M. Mauriello, M. Gubbels, and J. E. Froehlich. Social Fabric Fitness: The Design and Evaluation of Wearable E-textile Displays to Support Group Running. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems, CHI '14*, pages 2833–2842, New York, NY, USA, 2014. ACM.

BIBLIOGRAPHY

- [83] D. Meoli and T. May-Plumlee. Interactive electronic textile development: A review of technologies. *Journal of Textile and Apparel, Technology and Management*, 2(2):1–12, 2002.
- [84] J. Meyer, P. Lukowicz, and G. Tröster. Textile pressure sensor for muscle activity and motion detection. In *10th IEEE International Symposium on Wearable Computers, 2006*, pages 69–72. IEEE, 2006.
- [85] L. Morton, L. Baillie, and R. Ramirez-Iniguez. Pose calibrations for inertial sensors in rehabilitation applications. In *IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2013*, pages 204–211, oct 2013.
- [86] L. J. Najjar, J. C. Thompson, and J. J. Ockerman. A wearable computer for quality assurance inspectors in a food processing plant. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 163–164, oct 1997.
- [87] Z. Nakad, M. T. Jones, and T. Martin. Communications in Electronic Textile Systems. In *Communications in Computing*, pages 37–46, 2003.
- [88] G. Ngai, S. C. F. Chan, W. W. Y. Lau, and J. C. Y. Cheung. A framework for collaborative etextiles design - An introduction to co-etex. In *Proceedings of the 2009 13th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2009*, pages 191–196, 2009.
- [89] G. Ngai, S. C. F. Chan, V. T. Y. Ng, J. C. Y. Cheung, S. S. S. Choy, W. W. Y. Lau, and J. T. P. Tse. I*CATCH: A Scalable Plug-n-play Wearable Computing Framework for Novices and Children. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '10*, pages 443–452, New York, NY, USA, 2010. ACM.
- [90] M. Ockendon and R. Gilbert. Validation of a Novel Smartphone Accelerometer-Based Knee Goniometer. *Journal of Knee Surgery*, 25(04):341–346, may 2012.
- [91] J. J. Ockerman, L. J. Najjar, and J. C. Thompson. Wearable computers for performance support: initial feasibility study. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 10–17, oct 1997.
- [92] M. Pacelli, G. Loriga, N. Taccini, and R. Paradiso. Sensing Fabrics for Monitoring Physiological and Biomechanical Variables: E-textile solutions. In *3rd IEEE/EMBS International Summer School on Medical Devices and Biosensors, 2006*, pages 1–4, 2006.
- [93] J. A. Paradiso and E. Hu. Expressive footwear for computer-augmented dance performance. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 165–166, oct 1997.

- [94] R. Paradiso, G. Loriga, N. Taccini, A. Gemignani, and B. Ghelarducci. WEALTHY-a wearable healthcare system: new frontier on e-textile. *Journal of Telecommunications and Information Technology*, pages 105–113, 2005.
- [95] S. P. S. Park, K. Mackenzie, and S. Jayaraman. The wearable motherboard: a framework for personalized mobile information processing (PMIP). *Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324)*, pages 170–174, 2002.
- [96] H. Perner-Wilson, L. Buechley, and M. Satomi. Handcrafting Textile Interfaces from a Kit-of-no-parts. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction, TEI '11*, pages 61–68, New York, NY, USA, 2011. ACM.
- [97] R. W. Picard and J. Healey. Affective wearables. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 90–97, oct 1997.
- [98] E. R. Post and M. Orth. Smart fabric, or "wearable clothing". In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 167–168, oct 1997.
- [99] E. R. Post, M. Orth, P. R. Russo, and N. Gershenfeld. E-broidery: Design and fabrication of textile-based computing. *IBM Systems journal*, 39(3.4):840–860, 2000.
- [100] R. Ramakers, K. Todi, and K. Luyten. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2457–2466. ACM, 2015.
- [101] A. Rathnayake and T. Dias. Yarns with embedded electronics. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 385–388. ACM, 2015.
- [102] V. Savage, X. Zhang, and B. Hartmann. Midas: fabricating custom capacitive touch sensors to prototype interactive objects. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 579–588. ACM, 2012.
- [103] S. Schneegass, M. Hassib, T. Birmili, and N. Henze. Towards a garment OS: supporting application development for smart garments. In *Proceedings of the 2014 ACM International Symposium on Wearable Computers: Adjunct Program*, pages 261–266. ACM, 2014.
- [104] W. R. Sherman and A. B. Craig. *Understanding virtual reality: Interface, application, and design*. Elsevier, 2002.
- [105] Y. Shimokochi and S. J. Shultz. Mechanisms of noncontact anterior cruciate ligament injury. *Journal of athletic training*, 43(4):396, 2008.

BIBLIOGRAPHY

- [106] M. Sibinski, M. Jakubowska, and M. Sloma. Flexible temperature sensors on fibers. *Sensors*, 10(9):7934–7946, 2010.
- [107] J. Siegel and M. Bauer. A field usability evaluation of a wearable system. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 18–22, oct 1997.
- [108] S. G. Slater. New technology sensor fabrics to monitor health data. *Home Health Care Management & Practice*, 19(6):480–481, 2007.
- [109] A. Smailagic. ISAAC: a voice activated speech response system for wearable computers. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 183–184, oct 1997.
- [110] A. Smailagic and D. P. Siewiorek. Matching interface design with user tasks. Modalities of interaction with CMU wearable computers. *Personal Communications, IEEE*, 3(1):14–25, feb 1996.
- [111] M. B. Spitzer, N. M. Rensing, R. McClelland, and P. Aquilino. Eyeglass-based systems for wearable computing. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 48–51, oct 1997.
- [112] P. Stanley-Marbell, D. Marculescu, R. Marculescu, and P. K. Khosla. Modeling, analysis, and self-management of electronic textiles. *IEEE Transactions on Computers*, 52(8):996–1010, 2003.
- [113] T. Starner, J. Weaver, and A. Pentland. A wearable computer based American sign language recognizer. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 130–137, oct 1997.
- [114] M. Stoppa and A. Chiolerio. Wearable electronics and smart textiles: a critical review. *Sensors*, 14(7):11957–11992, 2014.
- [115] M. Sundholm, J. Cheng, B. Zhou, A. Sethi, and P. Lukowicz. Smart-mat: Recognizing and Counting Gym Exercises with Low-cost Resistive Pressure Sensing Matrix. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '14*, pages 373–382, New York, NY, USA, 2014. ACM.
- [116] I. E. Sutherland. A Head-mounted Three Dimensional Display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I, AFIPS '68 (Fall, part I)*, pages 757–764, New York, NY, USA, 1968. ACM.
- [117] A. Taivalsaari. Classes vs. Prototypes - Some Philosophical and Historical Observations. In *Journal of Object-Oriented Programming*, pages 44–50. SpringerVerlag, 1996.
- [118] H. Z. Tan and A. Pentland. Tactual displays for wearable computing. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 84–89, oct 1997.

- [119] X. M. Tao. *Smart Fibres, Fabrics and Clothing: Fundamentals and Applications*. Woodhead Publishing Series in Textiles. Elsevier Science, 2001.
- [120] B. Thomas, S. Tyerman, and K. Grimmer. Evaluation of three input mechanisms for wearable computers. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 2–9, oct 1997.
- [121] C. Thompson, J. J. Ockerman, L. J. Najjar, and E. Rogers. Factory automation support technology (FAST): a new paradigm of continuous learning and support using a wearable. In *First International Symposium on Wearable Computers, 1997. Digest of Papers*, pages 31–38, oct 1997.
- [122] E. O. Thorp. The invention of the first wearable computer. In *Second International Symposium on Wearable Computers, 1998. Digest of Papers*, pages 4–8, oct 1998.
- [123] D. Ungar and R. B. Smith. *Self: The power of simplicity*, volume 22. ACM, 1987.
- [124] L. Van Langenhove and C. Hertleer. Smart clothing: a new life. *International journal of clothing science and technology*, 16(1/2):63–72, 2004.
- [125] N. Villar, J. Scott, S. Hodges, K. Hammil, and C. Miller. .NET gadgeteer: a platform for custom devices. In *Pervasive Computing*, pages 216–233. Springer, 2012.
- [126] A. Wakita and Y. Anezaki. Intuino: an authoring tool for supporting the prototyping of organic interfaces. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, pages 179–188. ACM, 2010.
- [127] M. Weiser. The Computer for the 21st Century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, jul 1999.
- [128] Y.-L. Yang, M.-C. Chuang, S.-L. Lou, and J. Wang. Thick-film textile-based amperometric sensors and biosensors. *The Analyst*, 135(6):1230–1234, 2010.
- [129] S.-C. Yeh, S.-M. Chang, S.-Y. Chen, W.-Y. Hwang, T.-C. Huang, and T.-L. Tsai. A lower limb fracture postoperative-guided interactive rehabilitation training system and its effectiveness analysis. In *14th International Conference on e-Health, Networking, Applications and Services (Healthcom), 2012 IEEE*, pages 149–154, oct 2012.
- [130] H. Zhang, X. Tao, T. Yu, and S. Wang. Conductive knitted fabric as large-strain gauge under high temperature. *Sensors and Actuators A: Physical*, 126(1):129–140, 2006.
- [131] H. Zhang, L. Tian, L. Zhang, and G. Li. Using textile electrode EMG for prosthetic movement identification in transradial amputees. In *IEEE International Conference on Body Sensor Networks (BSN), 2013*, pages 1–5, may 2013.

BIBLIOGRAPHY

- [132] C. Zysset, K. Cherenack, T. Kinkeldei, and G. Tröster. Weaving integrated circuits into textiles. In *2010 International Symposium on Wearable Computers (ISWC)*, pages 1–8. IEEE, 2010.