Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

ADFIS - An Application Development Framework for Interactive Surfaces

Master thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Tim Vereecken

Promoter: Prof. Dr. Beat Signer
Advisor: Reinout Roels

Academic year 2014-2015

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN

VAKGROEP COMPUTERWETENSCHAPPEN

ADFIS - An Application Development Framework for Interactive Surfaces

Masterproef ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad
Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Tim Vereecken

Promotor:	Prof. Dr. Beat Signer
Begeleider:	Reinout Roels

Academiejaar 2014-2015

Abstract

Ever since the introduction of interactive surfaces in research, the software components of interactive surface systems have been implemented with minimal reuse of existing solutions. Due to the lack of frameworks that provide the necessary abstractions, developing interactive surface applications has involved implementing hardware interfaces and software abstractions. The time spent on implementing interfaces and abstractions could be better spent focusing on the application itself. Most systems are heterogeneous in nature, meaning that there are many varieties of hardware platforms and techniques to achieve an interactive surface. Despite the heterogeneity of interactive surfaces, many systems still have software components and abstractions that could be reused in other systems. Due to the popularity of multi-touch as an interaction method, multi-touch frameworks have been developed which provide developers with abstractions for multi-touch, but they do not encompass the full scope of the application space with regards to interaction methods.

We have created an overview of the current state of interactive surface research, from which requirements were created. Using those requirements, we created ADFIS. ADFIS is a cross-platform, hardware independent framework for the development of applictions for interactive surfaces, meant to alleviate the absence of a broad framework that provides useful abstractions. It offers a feature set that can be used to implement many applications using common interaction methods for interactive surfaces. The abstractions that ADFIS provides are event driven and generic, making them flexible and extensible. During development of ADFIS we implemented a paper document detection algorithm and created a hierarchical taxonomy for interactive surfaces. We also investigated the integration possibilities of different interaction methods on interactive surfaces.

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.
I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

First and foremost I would like to thank my parents for allowing me the opportunity to attend university, which is something too easily taken for granted. Thank you for the support over the course of my acadamic pursuits and for not allowing me to descend down the cascades of secondary education.

Furthermore, I would also like to thank my friends from my graduation year at SOFT and WISE lab for their company during my studies and my thesis period. A special mention goes out to my second family at the Vrije Universiteit Brussel, the VUB Players.

Finally, I would like to extend special thanks to my promoter Prof. Dr. Beat Signer and my advisor Reinout Roels for their their eagerness to explore a different research topic as well as their extensive guidance and advise over the past year.

Tim Vereecken
August 2015

Contents

1 Introduction

This introductory chapter aims to provide general information about this thesis. Section 1.1 defines the problem statement in which we argue the need for a framework for interactive surfaces. The solution that we propose is presented in Section 1.2, followed by a summary of every contribution in Section 1.3.

1.1 Problem Statement

Many research projects that involve the development of an interactive surface consist of a software solution for specific hardware. The software solutions generally have different requirements but provide a similar set of basic functionalities. Although there is some reuse of software components, it is usually limited to very specific functionality or those components are not generally found in other software solutions.

The presence of common features and/or similar components implies that there is a significant amount of time and resources being wasted on redundant work. Many applications could be built by reusing common components, since there is an overlap of the basic functionality that interactive surface components provide. This is especially true for applications built on the same hardware platform. Those applications can maximally reuse development resources. However, even in the case of different hardware platforms there is an overlap. No two interactive hardware platforms are the same, but two hardware implementations of an interactive surface that use different technologies for registering user input still share the function of registering input. As such, there is much to gain from providing interactive surface developers with a solid basis with generic functionality on which their applications can be built.

1.2 Objective

The aim of this thesis is to design and implement a solution that addresses the problem of redundant work in interactive surface application development. This solution consists of an application development framework that provides developers with a set of reusable and extensible components that will facilitate the development process of their applications. We decided to broaden the scope of the proposed solution by extending the framework's features to also support paper documents. This allows applications to register sheets of paper which can be interacted with. This interaction method allows for paper documents to be used as a bridge between the physical and digital space.

The targeted application space for a software solution for interactive surfaces is not easily defined. It could be defined by targeting a number of subsets of a classification system for interactive surfaces. In *Tabletops - Horizontal Interactive Displays* [44], three classification systems for tabletops are presented: by interaction, by identification & tracking and by a matrix representation of combining technical properties. Tabletops are a subset of interactive surfaces, but the classification system is still applicable.

The interaction method classification system, illustrated by Figure 1.1, subdivides into hand and device based interaction and respectively subdivides into finger and gesture interaction and mouse, stylus, graspable UI and phicon[1] interaction.

[1]A phicon is a physical object that refers to a digital object. It differs from graspable UI or tangibles in the sense that phicons tend to have a one-to-one mapping and its purpose is object reference rather than application of a function.

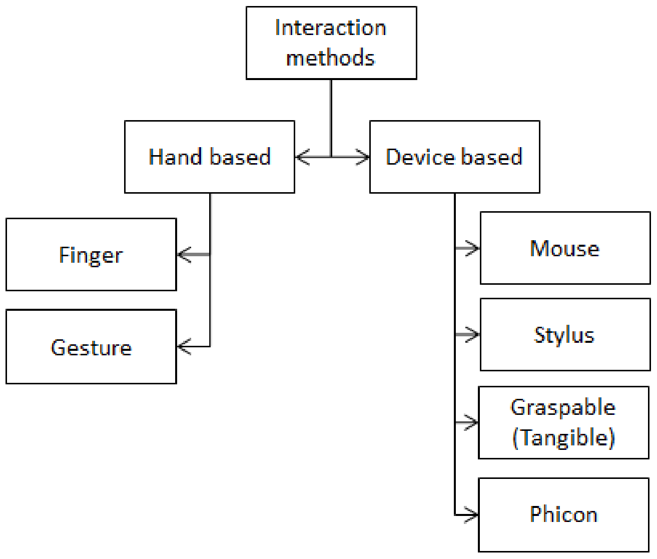


Figure 1.1: Classification by interaction method

Tracking and identification classification, illustrated by Figure 1.2, subdivides into optical, electrical and acoustic technologies. Optical technologies include visible and non-visible light, pattern recognition and fiducials as well as and blob and shape detection. Electrical technologies subdivide into resistive, capacitive, inductive as well as electromagnetic and frequency technologies.

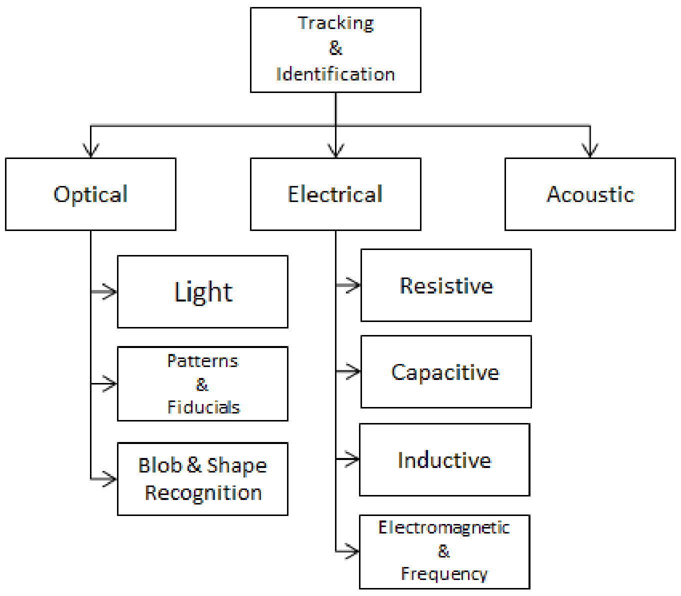


Figure 1.2: Classification by tracking technology

The aforementioned classification system is however very specific and does not focus on the larger scope of interactive surfaces. The proposed framework aims to be a solution that is independent of the technologies used in the hardware system. In Figure 1.3 we propose a hierarchical classification for interactive surfaces. ADFIS, the framework that we propose supports only flat interactive surfaces, which is a subset of the proposed classification system.

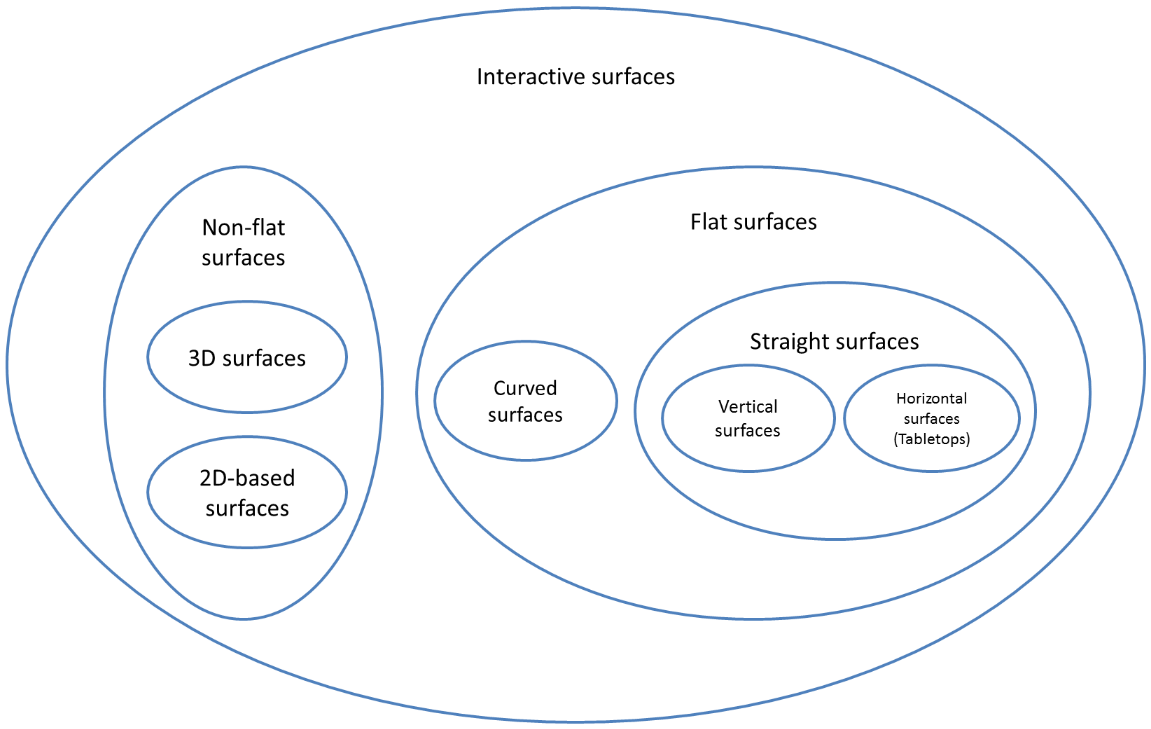


Figure 1.3: Hierarchical classification of interactive surfaces

Interactive surfaces that have a 2D base (such as systems with a deformable surface) and 3D interaction are not supported. Examples of such systems are the FuSa touch [46], an interactive display with a fur-like surface, and the Augmented Reality Sandbox [53], a system where sand is used on a surface to shape visually projected landscapes. Interactive surfaces that utilise a 3D hardware system, such as the Sphere [7], an interactive surface that is projected onto a diffuse ball, are also not supported. In theory, it might be possible that The Sphere's system could be implemented using ADFIS, but this would require mapping the plane of the 2D UI to 3D coordinates, depending on the hardware used. An example of a 3D system that can be partially supported is Dalsgaard & Halskov's tangible 3D tabletop [12]. The tangible 3D tabletop can be partially implemented using the framework since the framework provides support for tangibles. It does however incorporate a 3D projection aspect, which could be implemented as an extension to ADFIS, so while ADFIS would not be able to support all the features of the application, it would still facilitate development.

An example of a curved surface is the BendDesk [72] system. BendDesk is a research project that aims to investigate the interaction characteristics of combining horizontal and vertical interaction areas, connected by a curved area. There is no physical distinction between the horizontal, curved and vertical area, but from the users' perspective and conceptually they are different. Internally, two sets of cameras and projectors are used to display the interface and to track interactions. Although the authors found that the curved area of the surface is an impediment to interacting with the system, similar systems might be created that provide a solution for the deficiencies of BendDesk. The framework could be used to create multiple windows to cover each of the surface areas, although interaction between windows would have to be implemented by the developer and use of the curved area would require translation of the coordinate system.

1.3 Contribution

As presented in Section 1.2, we address the issue of redundant work being done during development of interactive surfaces. Due to the maturity of the research field of interactive surfaces it is not possible to implement a software solution without further examination of related work. Our contribution can be summarised as follows:

- We reviewed many publications with regards to interactive surfaces, of which the most significant publications are used to create a historical overview from the early prototypes to commercial applications.
- Due to the many hardware systems that have been designed and used over the years, we created a hierarchical classification of interactive surfaces which is used to determine the scope of the framework which we created.
- Through observation that flat interactive surfaces are used most commonly we engineered requirements for such systems, followed by implementation of a framework that satisfies those requirements. We

implemented ADFIS, a framework for application development for interactive surfaces.

- During the development of ADFIS, we investigated the integration possibilities of several interaction methods.
- Because of the nature of the development hardware which was used to create ADFIS, we implemented a paper document detection algorithm, achieved with blob detection for IR-based optical imaging technology.

The major contribution of the aforementioned list is that we have developed the first application development platform for interactive surfaces: ADFIS. To our knowledge and at the moment of writing, no other comparable frameworks exist. ADFIS is a cross-platform, hardware independent framework that supports the most popular multi-touch protocols, basic gestures and gesture extensibility, tangibles, blob recognition, projection and visualisation support. The ADFIS framework facilitates interactive surface development by providing development support on multiple levels. First, ADFIS includes an abstraction layer that allows application development to be independent of the hardware used. This is done by providing an abstraction layer that supports most multi-touch devices, enabled by support of the most used multi-touch protocols and the appropriate abstractions. Also included in this abstraction layer is support for gesture interaction, tangibles and paper document detection.
The abstraction layer is designed to be generic and extensible which allows for optimal reuse and adaptation of the functionalities that it provides. Most features in the abstraction layer can be used as preconfigured or customised depending on the application's requirements, such as the interceptor system for touch events (this is explained in more detail in Section 4.5). These features are components that are found in many interactive surfaces. Although specialised frameworks exist for many of the aforementioned components, combining two or more frameworks can prove to be problematic. ADFIS organises input events from multiple input sources and ensures that input events are made available to the UI layer in a unified manner. Another problem of specialised frameworks is that since they are heterogeneous, they might not be easily integrable with other technologies.

1.4 Structure

Chapter 1 is the current chapter which introduces the context and definition of the problem that ADFIS provides a solution for. After this introduction, related work with regards to interactive surfaces and interactive surface application development frameworks is explored in Chapter 2. From related work and the research area of interactive tabletops requirements were defined that are discussed in Chapter 3. Based on those requirements ADFIS has been designed and implemented. Its architecture is detailed in Chapter 4, and its functionalities is provided in Chapter 5. To demonstrate the capabilities of ADFIS, a demo application was implemented which is discussed in Chapter 6. A conclusion which summarises this project is given in Chapter 7, followed by a discussion of some future work in Chapter 8.

2 Related Work

This chapter aims to provide the context of interactive surfaces, focusing on applications of interactive surfaces and related technologies that either influenced or relate to interactive surfaces.

2.1 Interactive Surfaces

In this section we discuss the historical background of interactive surfaces, as well as recent research developments and commercial applications.

2.1.1 Pioneering Efforts

Interactive surfaces have been around since their early prototype versions. These prototypes were pioneering efforts to create an interactive environment using either horizontal or vertical spaces, mainly dedicated to a task or a specific collection of related tasks.

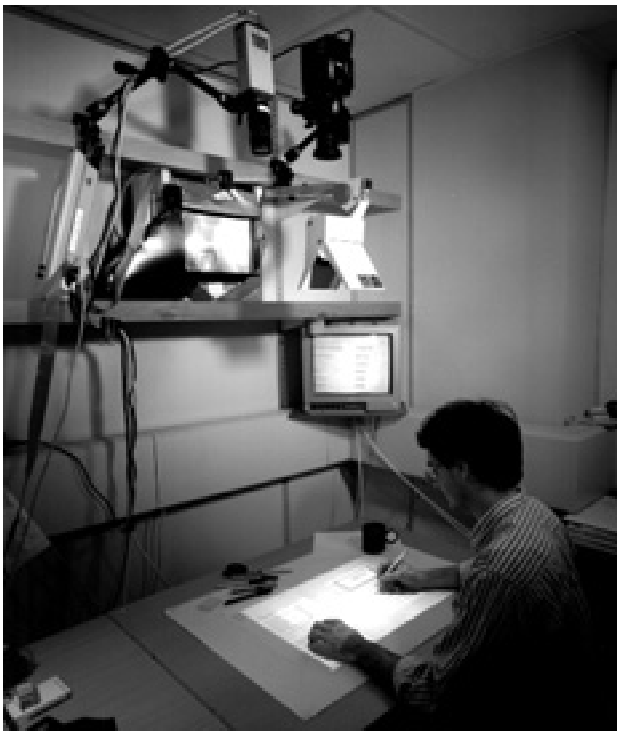


Figure 2.1: Wellner's DigitalDesk
(source: [73])

Pierre Wellner's DigitalDesk [73], shown in Figure 2.1, shows the first interactive surface, which is later expanded upon [74]. Wellner describes the DigitalDesk as a *"real physical desk [..] but it is enhanced to provide some characteristics of an electronic workstation"*. Wellner explains that DigitalDesk is an effort to bridge the physical-digital divide, for which he attempts to solve this problem by making the physical desk more like the digital desk. It features top-down projection onto the desktop and on paper documents on the desktop as a display method, document and character recognition as a registration method, and both pen and finger input as interaction methods.

Fitzmaurice et al. presented the ActiveDesk [20] in 1995 as a demonstration of the interaction paradigm they coined the *Graspable User Interface*. In the graspable user interface, users use physical objects or graspables that they can interact with, resulting in interactions with the system. The graspables used in their system are conceptually shaped like bricks, and their interaction methods are similar to how interaction is done in modern multi-touch applications (e.g. scaling, rotating or dragging). The authors state that the Graspable UI design philosophy has several advantages:

- *"It encourages two handed interactions;*
- *shifts to more specialized, context sensitive input devices;*
- *allows for more parallel input specification by the user, thereby improving the expressiveness or the communication capacity with the computer;*
- *leverages off of our well developed, everyday skills of prehensile behaviors for physical object manipulations;*
- *externalises traditionally internal computer representations;*
- *facilitates interactions by making interface elements more 'direct' and more 'manipulable' by using physical artefacts;*
- *takes advantage of our keen spatial reasoning skills;*
- *offers a space multiplex design with a one to one mapping between control and controller; and finally,*
- *affords multi-person, collaborative use. The capabilities of the system were demonstrated with a drawing application that used the bricks as handles for different shapes. The interface itself was displayed using back-projection while tracking of the graspables was done using an overhead system."*

The capabilities of the system were demonstrated with a drawing application that used the bricks as handles for different shapes. The interface itself was displayed using back projection while tracking of the graspables was realised using an overhead system.

The BUILD-IT[52] system is an interactive surface that allows for assembly line and production plant design. They recognise similar advantages of the interaction method of graspable UI that Fitzmaurice et al. proposed with the ActiveDesk but stress the natural character of the interaction and propose the *Natural User Interface* as a term for this interaction paradigm. In the BUILD-IT system, interaction is performed exclusively using bricks. The interactions are done on a table surface that serves as both a projection and interaction area. A 3D visualisation is projected on a vertical screen. The brick interactions are registered by projecting and detecting IR light using an overhead projector and camera.

Because of confusion due to the fact that *GUI* is also an acronym for *Graphical User Interface*, the term *Graspable User Interface* was later renamed to *Tangible User Interface* or TUI by Ullmer et al. [70]. They addressed this problem, and stated that tangibles in ActiveDesk did not sufficiently encourage the use of the interaction capabilities of the system. They translated the metaphors from the graphical user interface to the tangible user interface: windows became lenses, icons became phicons, handles became phandles. . . This translation allowed for a more intuitive interaction. To demonstrate their design for tangible interaction, they developed an application named *Tangible Geospace* which allowed for exploration of a geographical area. The application was developed on the metaDESK platform, in which the interface is back projected. The tracking of tangibles is done using back-projected infrared light which is reflected back to a camera also mounted underneath the surface.

Another interesting case to consider is SenseTable [48]. It features pucks as an interaction method, which are actually a step backwards from the model presented by Ullmer et al. since information about the function of the puck is projected on top of it. This provides a less intuitive interaction method. However, the major contribution of this research was not its interaction method, but rather its use of electromagnetic sensing which allows parts of the system's technology (namely the tracking component) to be integrated into the interactive surface itself, as opposed to front or back projection and detection. Projection is still located above the surface itself.

Like the SenseTable, the DiamondTouch system [15] is also one of the first interactive surfaces that integrates tracking technology into the surface rather than above or beneath it. The system works electronically, which allows for multiple users to be tracked through capacitive coupling. When a user touches the surface, a circuit is closed that is received in the chair of the user. Objects on the surface do not interfere with the system's functions, but this has the downside that only touch interactions are supported.

Similar to DiamondTouch, SmartSkin [54] integrates tracking into the surface itself and uses front projection to display its user interface. The tracking component tracks touch interactions through a mesh of transmitter and receiver electrodes, which also allows for gesture recognition. The largest contribution of this paper was that it was one of the first interactive surfaces to combine both touch and tangible interactions.

A major breakthrough in multi-touch technology was the introduction of Frustrated Total Internal Reflection (FTIR) to multi-touch detection [24], illustrated by Figure 2.2. This technique introduced multi-touch detection at a low cost. A sheet of acrylic serves as the interaction surface, while IR light is projected through the side of the sheet, which is totally reflected between the top and bottom sides of the acrylic. When the sheet is touched, the IR light is no longer caught between sides of the sheet, it is reflected outwards. This reflected light is captured by an IR camera, and used to detect touch points.

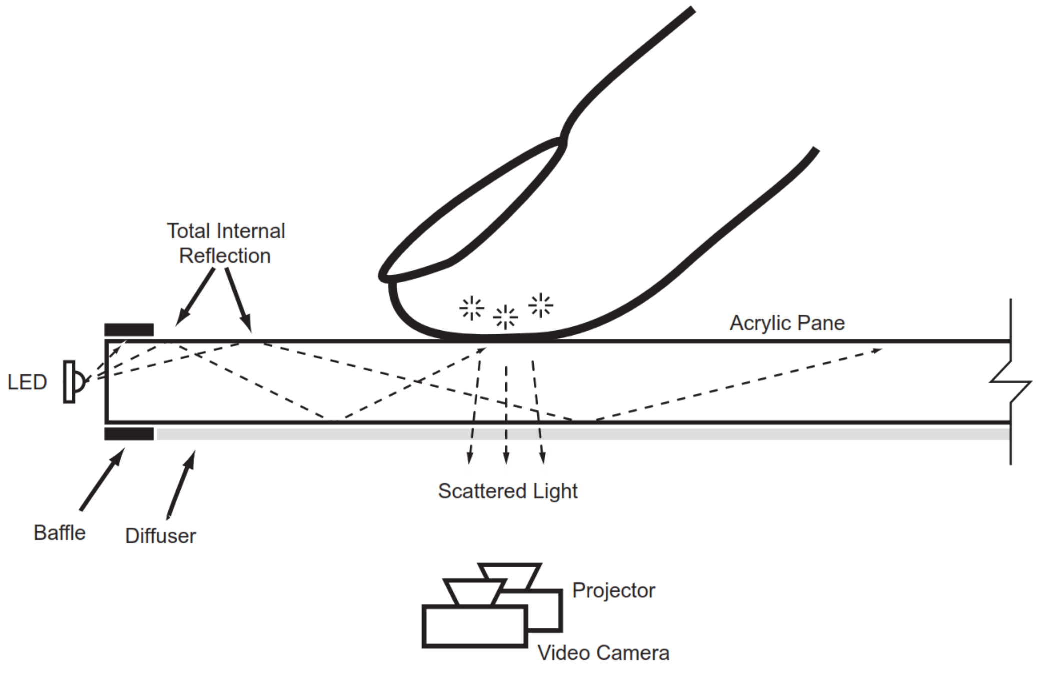


Figure 2.2: Schematic representation of FTIR technology (source: [24])

2.1.2 Modern Research

The research field of interactive surfaces has matured to a point where research is often multidisciplinary. Publications contain elements from many fields. These include engineering which focuses on hardware systems for interactive surfaces as well as human-computer interaction (HCI) which studies individual interaction techniques. In the next few sections, we discuss some interesting topics: hardware, multi-surface environments and interaction methods.

Hardware

Nowadays various hardware platforms and applications of interactive surfaces exist. The most popular type of interactive surface is the interactive tabletop. However, the term *interactive surface* is a broader term that encompasses multiple hardware solutions. Interactive surfaces also include vertical surfaces such as wall-mounted multi-touch screens. These range from small, table-size platforms such as the PlayAnywhere [75] to large, wall-sized applications such as Han's Multi-Touch Interaction Wall [25]. The PlayAnywhere is a portable device that projects its user interface next to where the device is placed and provides an interactive surface that has a surface area comparable to a large tablet. The Multi-Touch Interaction Wall is a scaled up version of Han's Multi Touch technology [24] to wall-size, as previously mentioned when we discussed FTIR.

One of the problems of the popular multi-touch technologies such as FTIR is that touch interaction does not provide tactile feedback. Mobile devices sometimes alleviate this partially with a vibration pulse, but this technique is not applicable to larger scale interactive surfaces because of it many limitations. Many attempts to provide a solution have been made, although many approaches have resulted in systems that are in an experimental stage or have a limited integrability, which makes them difficult to combine with other components of interactive surface hardware. Very distinct approaches have been explored as a solution for tactile feedback. Some systems use pin array systems, which elevate pins or rods using either actuators [29][38] or user interaction [8]. It is also possible to use a layer of fluid which can be made rigid locally by using electromagnets [30]. Another approach entirely is to use servomotors to make the entire surface provide tactile feedback by using FTIR technology and a 6-axis motion panel [45], althoug this restricts tactile feedback to two simultaneous touch points.

Multi-Surface Environments

The Environs framework [13] is a framework for development of distributed interactive applications in multi-display environments. The framework facilitates development by allowing a common codebase of an application. Only the device-specific code needs to be reimplemented. The communication model is based on Wi-Fi connection between devices, and can even handle devices being on different networks. Environs is currently limited by its focus on video streaming. While achieving good results in its scope (low latency while streaming), its feature set is limited.

The challenges and limitations of multi-surfaces environments become apparent when creating complex environments. The Cube [56] is a facility which contains several multi-surface environments. These environments use either multi-touch LCD displays as well as projected interactive surfaces. It purpose is to study multi-surface environments and interactions. Among the insights gained from the applications that are deployed in the Cube are discussed here:

- Touch events are mostly handled locally, which provides more responsiveness but also increases application complexity;
- because of the amount of touch events, one application had to change its client-server architecture to a peer-to-peer system which improved latency and performance;
- multiplexing touch events allows for easy multi-monitor interactions;
- existing multi-surface frameworks focus on rendering rather than end-user interaction.

These insights provide us with important aspects of multi-surface environments that need to be considered when creating abstraction layers and software support for such systems.
In fact, one such framework uses both server-client and peer-to-peer architectures. Using this mixed architecture, the PolyChrome framework achieves cross-device collaborative web visualisations for heterogeneous devices [5], but it is focused on web technologies only.

The Cube focuses on large-scale displays only. Other research has investigated multi-surface environments on a smaller scale using mobile devices such as tablets and tabletops. SkyHunter [64] uses a Microsoft Kinect[1] to detect orientation and position to achieve inter-device interaction by flicking UI elements in the direction of another device. HuddleLamp [49] uses only tablets and an RGB depth-aware camera to detect position and orientation. Its use of RGB entails that no IR light is used, which allows other hardware components to use IR. A commercial application which uses an inter-device communication technique is discussed in Section 2.1.4.

[1]https://www.microsoft.com/en-us/kinectforwindows/

Interaction

Recently, a research project named Kickables [60] has created one of the very few interactive surfaces situated on the floor, rather than using a wall or elevated horizontal interface. Kickables employs a whole new set of tangibles that are specifically designed for interaction through the use of feet. The tangibles are intuitive shapes that are adapted so that interaction results in detectable state, and the UI is projected around the tangible. The detection of state is done by bottom projection and detection, although the authors also indicate that top detection works, although there is an occlusion problem when using overhead hardware. When using back projection and detection, a special floor is required.

The physical orientation of an interactive surface can have an impact on its user interface. For instance, user interaction with a very large wall-mounted multi-touch screen will most likely be situated near the vertical center of the screen, since the vertical edges of the screen would be difficult for the user to reach. Therefore, the content and UI elements are best situated in the middle of the screen. Conversely, an interactive tabletop should have its content off-centre and UI elements closer to the side. Research has confirmed that user interactions on a tabletop are closer to the edge [62].

Due to the maturity of touch technology, the use of specialised UI components (widgets) is a very active research field. Menu widgets in specific are subject to investigation because of the issues with translating the UI design paradigms from desktop to tabletop [71]. Specialised radial menus have been created as a potential solution to menus on interactive surfaces, such as unimanual multi-finger interaction menus [6] and pull-out menus which are created with bi-manual gestures [76] HandyWidgets.

2.1.3 Educational Applications

By their nature, it is easy to conceive that interactive environments could be used to stimulate learning and facilitate teaching in educational settings. Teaching activities such as lectures can be facilitated with smart boards (typically, these are whiteboards augmented with front projection) or vertical interactive surfaces, which are meant to replace the traditional blackboards in the classroom. Smart boards and vertical displays allow for more than just drawing and writing. They can also be used for projection of media and annotations on different forms of content, as well as easy changing of teaching environment such as a different subject. Another possibility of using an interactive surface is that content can be shared with the students, provided they also have access to an interactive surface or linked device.

Research concerning the use of vertical surfaces is scarce. Instead, more research is being done that investigates the use of interactive surfaces from the student's perspective. The general trend in literature points towards a consensus that educational applications on interactive surfaces (and specifically, tabletops) is more advantageous in a context of collaborative tasks [39][27], such as co-located teamwork. What those collaborative tasks entail is one of the challenges of interactive environment research [63]. Interactive surfaces can stimulate creativity, indications of increased co-located collaboration. Tabletops can prove to be beneficial for inter- and intra-group collaboration, although it it difficult to make general statements about the applicability of the different technologies on the range of educational tasks [19]. Considering the amount of text produced by students in learning environments, multi-touch technologies can be restricting when text entry is required. If speech recognition technologies ever become accurate enough to replace keyboards, they could be used to avoid this problem.

As is generally the case when new, exciting technology is introduced, some trends can be observed. We tend to over-generalise our conclusions from experimental research, but these generalisations tend to be wrong. Some educational applications on interactive surfaces fail, but this cannot be generalised to the assumption that education applications cannot be beneficial. We also tend to over-expect technology to make tasks easier. Instead, we should make sure that the teacher using the application and system can appropriate them properly [16]. Interactive educational environments should have generic applications where content can be added easily by teachers using standard building blocks. User identification is also very important here, because applications need to be aware of the user to show progress to teacher and student for (self-)evaluation purposes [39]. Because of the importance of the teacher in the pedagogical process, it is necessary that the teacher can properly use the application and hardware to support the learning process [27]. Without proper understanding of an interactive system by the teacher, the system is at risk of negatively impacting the learning process and therefore discarded. Considering the investments required for such a system, this might cause aversion from educational institutions towards interactive environments.

2.1.4 Commercial Applications

Interactive surfaces in the private sector are currently in an early stage. Many commercial applications are being demonstrated during conventions or presentations, or seem to be very limited in scope. There is a noticeable trend in that most applications seem to be developed for the service industry; such as the catering industry, architecture and real estate. This is unsurprising, as the activities in those sectors tend to occur while sitting down at a table. Semi-private institutions such as museums and hospitals also seem to be environments where interactive surfaces can be found.

Software Solutions

The Interactive Concept Table [10] developed for Pizza Hut by Chaotic Moon Studios shows a commercially-ready application that combines many aspects of a visit to a restaurant. General information is displayed to the side of the surface, while the selection of dish is presented in the middle. This selection consists of flexible customisation of a dish and accommodates user interactions from all sides, although only one user can interact with the surface at a time. After an order has been completed, the system also provides entertainment such as games. Another observation can be made: one of the users places their phone on the surface, which reacts to it and provides visual feedback by outlining the phone's shape. This is most likely done by incorporating Near Field Communication (NFC) detection in the surface. After ordering, the user uses an interaction menu next to the phone to pay for the order.
Another restaurant application was built by TMSolutions [68], which is more generic than the Interactive Concept Table. It also allows for ordering of dishes using menu browsing and selection through multi-touch interactions on the table, but also allows for calling a waiter to the table and calling a taxi to the user's location.

The Cleveland Museum of Art[2] has a number of interactive surfaces that facilitate art exploration. Most of the applications are mainly multi-touch, but some interactions are done through body gestures. The use of tablets allows users to obtain more information about artworks using proximity sensing. Tablets can also be plugged into a stand at an interactive wall, where the user can save interactions with the wall to their tablet.

[2]http://www.clevelandart.org

SmartPixel.tv [66] developed a tabletop application for real estate agents, which allows them to present visualisations of real estate. In their application, a visualisation of multivariate data can be adapted on the fly: by using sliders to change price boundaries and room properties, the visualisation changes its display of available apartments in a building. It also shows the building in its neighbourhood and points of interest nearby.

Technology Demonstrations

An interactive environment was presented in 2014 by Samsung [58]. This environment consists of an interactive table that supports multi-touch interaction and smartphone interaction, using a three-section wall (main view and two periphery views to the side) which can be interacted with using hand and arm gestures. The three-section wall is illustrated by Figure 2.3. Those gestures are detected through the use of bar shaped IR detector and a camera that registers body position. The detection of the smartphone is done by attaching small bar codes to the side of the smartphone, which are recognised by cameras inside an elevated border of the surface. Interactions with the wall do not seem to cause any response on the table, but elements of the table can be dragged to the side oriented towards the wall, causing some communication such as displaying an image. Other interactions include file sharing between smartphones by dragging them across the surface to another device.

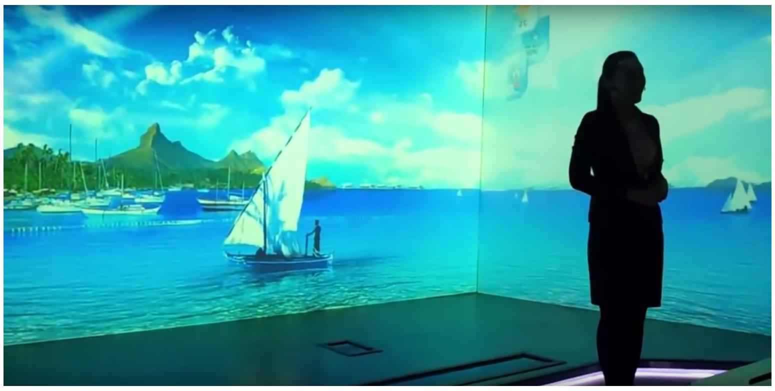
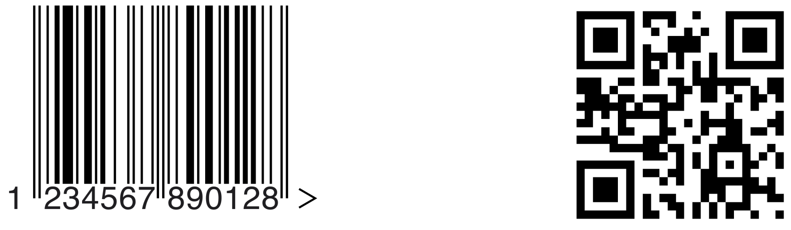


Figure 2.3: The wall component of Samsung's interactive environment (source: [58])

The *On the Fly Paper* system by Takram design engineering [67] is a concept interactive surface that uses overhead projection of the interface. Interaction with the interface is done using a tangible that consists of a sheet of paper or plastic with holes in a specific pattern. An infrared projector covers the entire surface with a projection of IR light. The surface reflects IR light, which is picked up by an overhead mounted camera. The pattern of light that the tangible lets through is interpreted as a specific command. The location and orientation of the tangible is also detected, which allows for the overhead projector to display images onto the tangible. Moreover, the tangibles pattern can be changed by covering one or more holes. This triggers specific commands, such as play/pause/stop functionalities in a video player, or selection of elements in a periodic table.

2.2 Paper Documents

Depending on the requirements of the application, the use of paper documents on interactive surfaces requires the use of advanced technologies. If an application only needs to identify that there is a document present, then this can be achieved using several techniques. The easiest technique is through the use of fiducials. Generally, fiducials or fiducial markers are objects that provide either a point of reference or a measure. Fiducials are commonly used in archaeological photography to provide a sense of scale. In the absence of standardised instruments, rulers or coins are good references for scale. In computer science, fiducials tend to be machine readable. Other fiducials in computer science include the ubiquitous barcodes used in supermarkets and QR codes , which are bar codes in a matrix format (Figure 2.4). This allows for the encoding of more information and can generally be recognised by any smartphone with a camera.



(a) A barcode (b) A QR-code

Figure 2.4: Examples of fiducials (source: [3])

[3] https://upload.wikimedia.org/wikipedia/commons/thumb/3/3a/Example_barcode.svg/2000px-Example_barcode.svg.png
https://upload.wikimedia.org/wikipedia/commons/5/55/Qrcode_wikipedia_fr.jpg

In the context of interactive surfaces, fiducials are — usually predetermined — graphical symbols. The pattern of those symbols is known beforehand to a recognition component that uses a certain algorithm to recognise those patterns in an image. The nature of fiducials allows the application to quickly identify and track the position of the fiducial and whatever object it is attached to. Most fiducials also allow for detection of orientation. If the fiducial is always attached in the same location on a document (which could be achieved by pre-printing them on a paper document), a document can always be found. Fiducials are also robust, distortion or lighting changes affect them minimally.

Another method of tracking through the use of paper augmentation is the use of tags, such as RFID (Radio Frequency Identification) tags. These tags are small tags that emit a specific radio frequency, which can be used to identify the object that they are attached to. Smaller tags tend to be powered by induction; the proximity of the magnetic field in the reader induces an electrical current in the tag, which then emits its signal. Using multiple receivers the location can be tracked, for instance through the use of a mesh or triangulation. Electrical tracking would require a specific hardware setup. For instance, receivers can be placed beneath the surface. In that case, the signal would have to be strong enough to penetrate the surface, as well as not interfere with the surface itself.

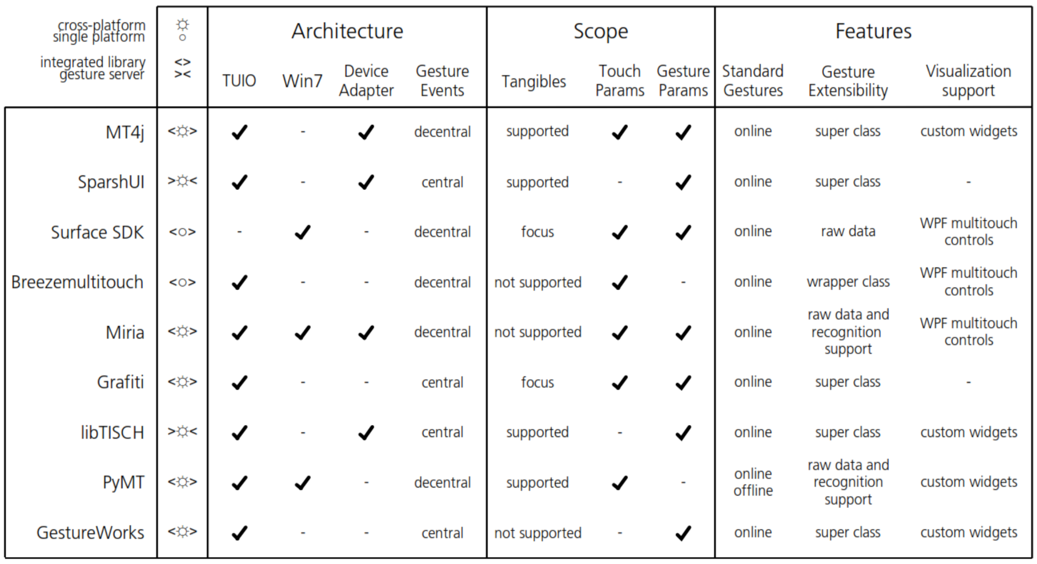
When not using fiducials, tracking documents becomes harder. A compromise solution was explored by Kim et al. [35], in which they achieved non-obtrusive document recognition and tracking through the use of images of documents. Those images are stored, along with their corresponding SIFT features (SIFT is an algorithm that generates distinct features for image recognition). In the system, a video feed is processed frame-by-frame. Each frame is analysed for SIFT features. When SIFT features have been found, fuzzy matching retrieves the document that corresponds with the feature set and results in a match. There are some drawbacks to this approach. The first is that the system matches documents offline, which is an issue when real-time interaction is required. Secondly, the recognition rate is not 100%, because the SIFT algorithm does not always generate enough features for good recognition and when using fuzzy matching, bad matches can occur. Finally, any paper document that is used must be known to the system, the matching needs a set of features that are matched against. It it clear that this approach also has some very distinct disadvantages.

2.3 Multi-touch Frameworks

In this section we discuss some of the available multi-touch frameworks that exist, commonly used for tabletop applications.

2.3.1 Taxonomical Overview

In 2010, Kammer et al. [33] created a taxonomy and overview of available multi-touch frameworks. Many aspects of a framework were considered, which are categorised into feature set, scope and architecture. The feature set category considers the basic gesture functionality a framework supports, how gestures can be added and the degree of visualisation support. The scope category considers whether a framework supports tangibles, if the parameters of a touch event can be accessed and if the parameters of a gesture event can be accessed. The architecture category considers TUIO supports, Windows 7 support, device adapter support and centralisation of gesture events. A visual overview of the categories and frameworks can be seen below in Figure 2.5.



cross-platform / single platform; integrated library / gesture server	☼ ○ <> ><	Architecture				Scope			Features		
		TUIO	Win7	Device Adapter	Gesture Events	Tangibles	Touch Params	Gesture Params	Standard Gestures	Gesture Extensibility	Visualization support
MT4j	<☼>	✓	-	✓	decentral	supported	✓	✓	online	super class	custom widgets
SparshUI	>☼<	✓	-	✓	central	supported	-	✓	online	super class	-
Surface SDK	<○>	-	✓	-	decentral	focus	✓	✓	online	raw data	WPF multitouch controls
Breezemultitouch	<○>	✓	-	-	decentral	not supported	✓	-	online	wrapper class	WPF multitouch controls
Miria	<☼>	✓	✓	✓	decentral	not supported	✓	✓	online	raw data and recognition support	WPF multitouch controls
Grafiti	<☼>	✓	-	-	central	focus	✓	✓	online	super class	-
libTISCH	>☼<	✓	-	✓	central	supported	-	✓	online	super class	custom widgets
PyMT	<☼>	✓	✓	-	decentral	supported	✓	-	online offline	raw data and recognition support	custom widgets
GestureWorks	<☼>	✓	-	-	central	not supported	-	✓	online	super class	custom widgets

Figure 2.5: Overview of multi-touch frameworks
(source: [33])

The most notable difference between frameworks besides their implementation language is the centralisation of gesture events. Most gesture events are not centralised and use the event system of the implementation language.

Gesture events are dispatched from gesture recognisers and are received by listeners that are registered to listen to those events. Grafiti [14], GestureWorks [28], libTISCH [17] and Sparsh-UI [51] process gestures inside the framework itself, as opposed to forwarding touch events to application components that perform gesture recognition.

libTISCH and Sparsh-UI also have a stricter divide between gesture recognition and application. In libTISCH, a gesture recognition layer is used which sends gesture messages over a network protocol. Sparsh-UI uses a gesture server where input devices send touch events to, and gesture events are sent to clients from the gesture server. Because of this network communication, the client can be written in any programming language provided they adhere to the protocol defined by the server.It is suggested that the centralised architecture of gestures be integrated in the operating system itself. Because of its already widespread support, it might be interesting to investigate the possibility of integrating this into the TUIO protocol, possibly by moving gesture recognition into the hardware itself.

Another notable difference is the lack of TUIO support by the Surface SDK, which can be explained by the fact that the Surface SDK is specifically created for the development of applications on the Microsoft PixelSense[4] (previously: Microsoft Surface). This limits the applicability of the framework considerably, especially since device adapter support is also absent. In that sense, Miria is the only framework that offers maximal hardware compatibility supporting both TUIO and the Windows 7 touch protocol as well as providing support for device adapters.

Most frameworks provide touch and gesture parameters. In the TUIO protocol, which most frameworks support, position, speed, acceleration and angular distance are given. This allows frameworks to forward this information to the application. Sparsh-UI lacks those parameters because of its strong focus on gestures, which any touch events are converted to and has many gesture parameters instead. The absence of gesture parameters in Breezemultitouch and PyMT is unclear.

Gestures can be either online or offline. In the case of offline gestures, the touch events are only processed on the last release of a touch point. The distinction between online and offline gestures is only made by PyMT, which has explicit support for offline gestures. Extensibility of gestures is usually

[4]https://www.microsoft.com/en-us/pixelsense/default.aspx

done by subclassing a `Gesture` class, although some frameworks support this through raw data and recognition support or a wrapper class.

The authors state that most frameworks were — at the time of writing — still under development and might prove unstable. Some frameworks also had a very limited flexibility.

2.3.2 Tactive

Tactive [22] is a framework for cross-platform development of tabletop applications which was developed after Kammer et al. published their taxonomical overview. Similarly to the problem statement of this thesis, the authors of Tactive also observed that many tabletop applications implement their own software solutions, despite the fact that they share common features. Tactive consists of two layers: a layer that handles operating system and hardware and an application layer. The lower layer provides support for multi-touch if it is not present in the operating system and supports the qTUIO [1] library, a library that integrates the TUIO library with Qt[5], a cross-platform application development framework. The application layer contains handlers that organise communication between layers and other application components, as well as several UI components that facilitate UI development.

In the context of interactive surfaces, Tactive provides a strong basis to develop applications. As its main development feature for new applications, it allows for abstraction of both software and hardware details and developers are able to implement their software using only web technologies (mainly JavaScript). Because of this strong integration of web technologies, it also supports encapsulation of web applications into widgets using web views, so already developed applications can be easily integrated in a multi touch application. Tactive, however, is focused on multi-touch applications only, and does not make the distinction between the terms *tabletop* and *interactive surface*. It supports only multi-touch and does not support other user interactions such as tangibles and paper documents.

2.4 User Identification

One of the issues in interactive surfaces has been the problem of user identification, which is a problem that occurs when multiple users interact with the same device. Some interaction methods of interactive surfaces allow for

[5]http://www.qt.io/

identification when they are inherently identifiable, such as tangibles. They can easily be linked to a specific user, although this restricts the use of a tangible to one user. The same reasoning holds for paper documents. Although this also has limited applicability, gestures can be used to identify users, such as when performing complex, bi-manual gestures.

Due to its importance in interactive surfaces, user identification by analysing user input has been investigated since the DiamondTouch, as previously discussed in Section 2.1. The DiamondTouch uses capacitive coupling to identify touch points; a touch press closes a circuit which the user is a part of. Also previously mentioned is that using capacitive coupling restricts interaction methods to touch. It does have the advantage that it is not intrusive, the user does not have to change anything to their interaction method.

Since DiamondTouch other methods have been developed that are intrusive. This includes the use of visual tags. This involves tagging users' hands or gloves with fiducial markers [40][41]. This technique even allows to identify which finger is causing a touch point. Other body modifications include other wearables such as IdWristbands [42] and IR-Ring [57]. Both methods use IR light to identify users. IdWristbands uses Arduino-controlled wristbands with integrated IR leds that blink at a unique frequency, while IR-Ring functions similarly but uses pseudo-random blink patterns. Another non-intrusive technique is the use of proximity sensors, as used by Medusa [2]. Medusa tracks user presence; body position; can distinguish between left and right arm and maps users to touch points. A disadvantage of Medusa's approach is that the mapping can be ambiguous which can result in faulty identification. The use of proximity sensors has also been investigated in combination with reactive territoriality [37]. HandsDown [59] is a system that identifies users by the contour of their hand imprint and requires users to put their hand flat on the surface to recognise the imprint. This technique is limited to user identification and cannot track individual touch interactions.

Some systems identify users based on the characteristics of users' feet and shoes. The Smart Floor system [47] uses pressure-sensitive floor tiles to register footsteps and using user profiles of footsteps, it maps the data to touch points. Although achieving good footstep recognition results, the mapping step is inaccurate. The required use of custom floor tiles is also a disadvantage. A similar system is Multitoe [4], which identifies users through recognising sole imprints. This is achieved by using a back-projected floor and FTIR. Multitoe is focused on interactive floors and does not provide a link to multi-touch interaction using hands. Finally, Bootstrapper [55] explores user identification through recognition of shoe profiles, which are registered with depth cameras.

All of the aforementioned systems have significant disadvantages. Some systems are intrusive, requiring either the wearing of small devices or require special interaction. Other disadvantages include position dependence, meaning that users are required to have a fixed position at the interactive surface. Finally, some methods have difficulties linking a user identity to touch interactions. Recognising that these problems are major hindrances, Carpus [50] is a non-intrusive user identification method that is position independent and accurately links users to touch interactions. It uses a high-resolution overhead camera to capture the dorsal region of the hand, from which features are extracted. Those features are used to match with future hand images. The algorithm also recognises fingers, which allows for non-ambiguous linking of identity to touch point.

2.5 Conclusion

The research field of interactive surfaces has known many systems since its origins in the nineties, most of which are very distinct. Although most interactive surfaces support touch interaction in some form, other interaction methods have always been an important aspect. Those other interaction methods are at their most valuable when they contribute to providing easier interaction, such as when they are used complementary to multi-touch. This can be observed especially when considering user identification on interactive surfaces, which has many downsides when implemented using touch interactions. The alternatives, such as tangibles and the Carpus system [50], are non-intrusive. They do not require any extra effort for the user as would be the case when requiring a change in interaction method, such as is necessary when using — for instance — HandsDown [59].

From reviewing literature on educational applications we learned that interactive surfaces offer significant advantages to team-oriented educational tasks, on the condition that the teacher uses the system and its applications correctly. Although literature on the applications which are available to educational organisations is limited, we observe that tabletops are predominantly used. The use of tabletops implies a strong reliance on multi-touch as the main interaction method. Other interaction methods offer opportunities but they require more research [19]. Commercial applications follow a similar trend of having multi-touch as the main interaction method although the complementary use of tangibles [67] and physical objects in the environment are considered as well [68].

One of the most important findings from reviewing literature is that virtually no interactive surface makes any mention of having used previously developed frameworks. Some software components, such as the TUIO, are reused but offer only support at a lower level, leaving developers to implement most of the abstractions necessary themselves. Multi-touch frameworks are available, although they know varying limitations [33] and only provide support for multi-touch interactions. Individually, libraries and/or frameworks are available for other interaction methods, but the complementary use of other interaction methods is not supported by any known framework. In the next chapter, we propose requirements for such a framework.

3 Requirements

In the previous chapter, we reviewed literature on interactive surfaces from early prototypes to modern research. Based on our findings from related work, we defined requirements for a framework that would facilitate the development of applications for interactive surfaces. Those requirements are presented in this chapter. In the following sections we describe these requirements in detail. Chapter 4 presents the architectural design of how ADFIS implements those requirements, and Chapter 5 details the extent to which ADFIS satisfies these requirements.

3.1 Multi-touch Support

Multi-touch capabilities are are a major part of the functionalities found in interactive surfaces. Other interaction methods are found mainly in research projects whilst multi-touch has become ubiquitous since the introduction of multi-touch screens in smartphones. As such, multi-touch interaction is being regarded as an interaction method that feels natural to users, although more research is needed to support this hypothesis.

In the context of interactive surfaces, the available multi-touch capabilities are almost exactly the same as the multi-touch capabilities of the average smartphone. These capabilities consist of the detection and tracking of touch points. Detection and tracking of touch points are structured in many frameworks or SDKs by distinguishing three touch point states: press, move and release, albeit by different names (e.g. birth, update and death) but they represent the same state. This state should also include an identifier and the location of the touch point. Some systems also include a *stationary* state in which a touch point has exited the *pressed* state but is not in a *move* or *released* state.

Using the information stream of touch point states, application elements can react to these state updates which is generally done using an event system, since the asynchronous character of event system means that processing state updates does not block the entire application. In fact, this system is also widely used to process mouse and keyboard input. Multi touch support is also required to integrate multi-touch gesture recognition capabilities.

Lately, many hardware platforms make use of the TUIO protocol. The TUIO protocol [32] is a communication protocol for multi-touch surfaces. It allows for abstraction of touch and tangible events and as such it can be used to facilitate implementation of tabletop applications (but is not limited to horizontal surfaces) by providing interfaces for sensors and displays.
As it was designed to allow for interconnection of table interfaces, its default communication channel is UDP (User Data Protocol) transport. Anyone familiar with TCP/IP and UDP/IP could remark that UDP is not obvious choice for communication hardware events since UDP does not guarantee successful transmission or correct ordering of packets. The specification of TUIO explains that the authors opted for UDP because of its low latency. The downsides of UDP (most notably, potential packet loss) is mitigated with the inclusion of redundant information to packets, which allows the system to handle packet loss gracefully. Since TUIO has become a standard protocol, any framework that offers multi-touch functionalities should be compliant with the protocol.

3.2 Gesture Support

Multi-touch gestures can be seen as an extension of multi-touch, or as a subset, depending on the definition of multi-touch interaction. In multi-touch interaction, the availability of information of more than one touch point simultaneously allows for gesture processing[1]. Gestures, such as simple multi-

[1]However, it is important to note that gestures such as double tap or single stroke interactions are also gestures, although they are not multi-touch interactions.

touch interactions, have become natural interactions. The most common and intuitive gestures include dragging (single or multi-touch), rotating and scaling and are usually found in gesture frameworks. Dragging is sometimes omitted since it is possible to bind the position of a touch point to a translation of an object, which results in the same outcome.

Beyond the set of standard gestures, many complex gestures can be implemented. The possibility space of gestures includes gestures using two hands and distinguishing between the number of fingers used in the gesture. Complex gestures open up a whole range of interaction possibilities and application short cuts that can be used, making multi-touch gesture interaction a very verbose interaction method. Going even further, gestures combining multi-touch and tangibles could be implemented.

3.3 Tangibles Support

Tangibles are physical objects that can be placed on an interactive surface. They can be useful to have in an application context, since they differ from touch interaction in some interesting ways. First, they are more persistent than touch points. Every time a touch point is released, its existence is lost forever to the application. Another touch point with the same identifier might exist, but they cannot be considered the same. Tangibles tend to have a one-to-one mapping with their identifiers. Another part of their higher persistence is that the time between entry and removal is generally longer than that of a touch point.

Another characteristic of tangibles is that they can have orientation and size, which makes their static properties more expressive than that of a touch point. On the flip side, the dynamic interaction of touch points (gestures) are more expressive than those of tangibles, but depending on the requirements of the application, the different properties of interaction makes tangibles a useful application component.

3.4 Paper Document Support

The use of paper documents on interactive surfaces has been around since Wellner's first conception of the DigitalDesk [73], previously mentioned in Section 2.1. Wellner's original critique on the prediction of the paperless office still stands: as shown in Figure 3.1, the paper industry did not show any signs of decline before 2009, which is when the financial crisis hit every market. After 2009, the production of paper products has since increased and is showing signs of stagnation [21]. These statistics are indicative of an ever steady use of paper.

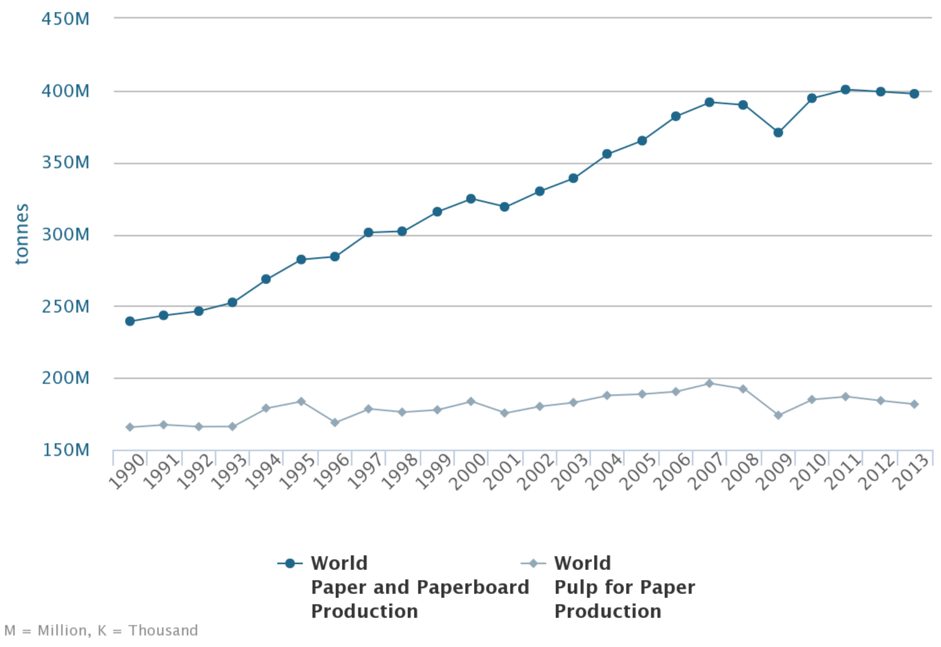


Figure 3.1: Evolution of paper production from 1990 to 2013 (source: faostat)

Sellen & Harper published *The Myth of the Paperless Office* [26] in 2001, in which they argue that the paperless office is most likely a false prediction. Their research points towards an increase in total paper usage, rather than a decline. This is a significant finding, considering that - according to the authors' research - 30 to 40 percent of the paper use (in the United Kingdom and the United States) occurs in the office. Due to its physical nature, paper is also easier to understand and to interact with. Interacting with digital documents requires interaction skills that are harder to obtain. Sellen & Harper state that paper will preserve its predominant position in knowledge work.
Sellen & Harper's work is slightly dated, but other studies seem to confirm the findings of their work. An example of such a publication which has shown that paper is still in active use can be found in [69].

It remains unsure whether the paperless office will ever arrive, but presently we cannot ignore the presence of paper in our daily lives, as well as individual preferences to paper usage.

3.5 Territoriality

Applications sometimes require interactions such as selection of objects. This functionality is difficult to implement when dealing with multiple users on the same interface. Multiple solutions for this problem exist.

One of those solutions is to group interactions together based on locality: whenever touch points are below a certain distance threshold, they are considered to be coming from the same user and could be interpreted as a gesture. The advantages of this method are that users can freely move around the interactive surface. The disadvantages include the fact that interactions from users which are close to each other might interfere. This ambiguity could be difficult to handle properly, especially when gestures are targeted at the same object and are contradicting.

Another approach to this problem is to split the surface area into smaller areas that contain user interactions per user [60][9]. This solution has the advantage that user interactions do not need to be preprocessed and there is no ambiguity as to the destination of the interaction events. There is also the possibility of implementing support for user area communication, which can change the semantics of an interaction based on the source and destination user area. The disadvantages of this approach are that user cannot freely move around the interactive surface without moving their user area. The most suitable solution will depend on the application area.

3.6 Visualisation Support

As interactive surfaces are heavily reliant on visual interaction, it is strongly advised to include a graphics library in a development framework. This graphics library should be high level and focus on rapid development rather than performance optimisation. Although the focus of the support should be high level, performance can be an issue. For instance, the underlying hardware platform can be a bottleneck, but this most likely occurs when dealing with a lot of users or when using resource-demanding graphics. For the latter it is generally possible to use a lower level graphics library that can be integrated in the application. For instance, many graphics libraries have

OpenGL[2] bindings to accommodate demanding graphics. As discussed in Section 2.3.1, visualisation support in multi-touch frameworks varies heavily from no support at all to widget support. The advantage of providing no visualisation support is that the developer can choose how to implement the interface. Such is the case when using Sparsh-UI (or any other framework that uses a network protocol to send touch data) which allows the developer to choose any framework in any impelmentation language.

3.7 Platform Independence

Platform independence can refer to both hardware and operating system independence or cross-platform compatibility. As mentioned in [33], hardware independence can be achieved by creating a hardware abstraction layer or by designing a framework to work with hardware independent protocols such as TUIO or the Windows 7 touch protocol. It is also possible to provide support for device adapters, which are small pieces of code that handle communication with a hardware interface.

Operating system independence refers to the ability of the system to be ran independent of the hardware's operating system. This can be achieved in two ways. The first option is to implement a framework in an implementation language which is operating system independent by itself such as Java or C# since they both run on a virtual machine. The second option is to build an abstraction layer that supports various operating systems. Such an abstraction layer would provide a unified interface to the hardware that handles differences related to the operating system. This allows applications to use the provided abstractions without having knowledge of the underlying hardware and how the hardware is accessed.

ADFIS is implemented in Java. The choice of programming language is mainly due to its popularity, which allowed us to explore many options regarding frameworks for individual components since many libraries have a Java API. Java applications are also cross-platform since they rely on execution in the Java Virtual Machine (JVM) rather than compilation.

[2]http://www.opengl.com

4 ADFIS Architecture

This chapter discusses the hardware that was used to develop ADFIS, explains its architectural design and documents the implementation details of the framework.

4.1 Development Platform

Ideally, a hardware solution for interactive surfaces has every component integrated into the surface. The perfect platform does not have any hardware components above or beneath the surface. This presents many engineering challenges and such a system does not exist at the time of writing. It can be argued that the next best alternative is back projection in the case of horizontal orientation of the interactive surface, since the space underneath the surface is less used and provides a less intrusive setup than when using overhead mounted components. It should be noted that hardware components beneath the surface will restrict the use of the system to a standing position, as there will be limited or no leg room. Back projection does have the added advantage of not creating occlusion issues as is the case when using overhead projection or detection.

A schematic of the hardware platform is given in Figure 4.1. It shows the hardware components used by ADFIS: multi-touch is achieved with an overlay over an LED TV which displays the application's GUI. A camera and projector are mounted overhead on the ceiling. The camera and touch overlay are connected to a computer with USB, and the projector and LED TV are connected with a VGA and HDMI cable respectively.

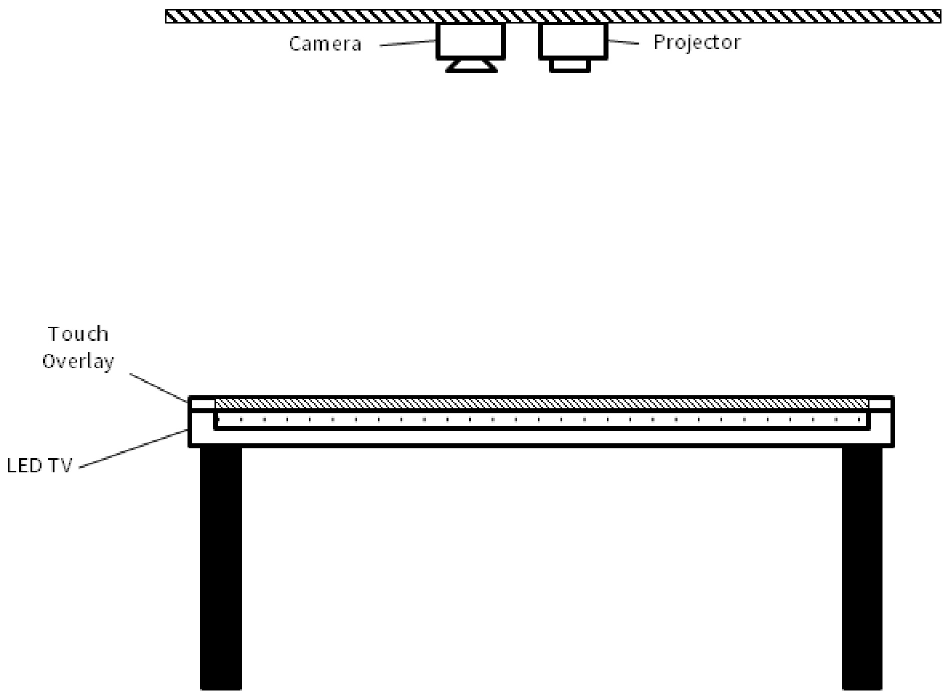


Figure 4.1: Schematic of the development hardware

The development of our framework and its demo application was done using a Samsung UN46F5000 TV[1] with an overlay specific for that model from Tabler.tv which is shown in Figure 4.2.

[1]http://www.samsung.com/us/video/tvs/UN46F5000AFXZA

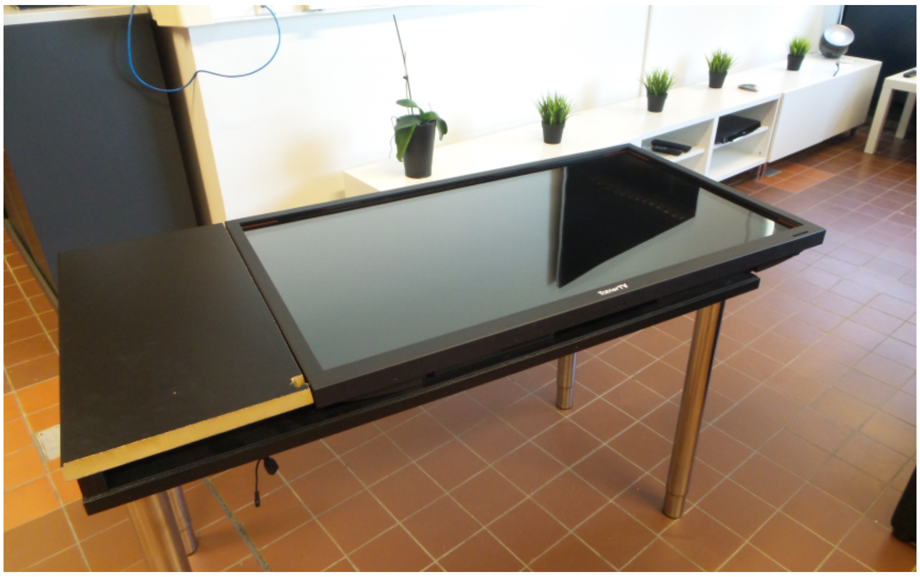


Figure 4.2: Samsung UN46F5000 with overlay from Tabler.tv, mounted on an adjustable table

The overlay consists of a frame that is mounted on top of the UN46F5000 screen. This allows for touch point detection over USB using Windows 7 touch or the TUIO protocol. The technology used in this frame is a patented touch detection technique named Cell Imaging Technology, of which the details are not public domain. What is known is that the technology uses IR-based optical imaging to detect touch points. This technology consists of modules that project IR light and detect changes in the registered light. If the changes are above a certain threshold, touch is detected. A maximum of 32 touch points can be detected simultaneously.

The technology used by Tabler.tv — IR-based optical imaging — has a number of advantages and disadvantages. On the upside, the technology is very cost-effective. Because of the use of modules, applications that require a less demanding hardware can opt for less modules which reduces the total cost. Detecting IR light is also quite fast, in a test that draws circles at the position of touch points we found no noticeable delay. The most negative point is that because of the use of IR light, sunlight will interfere with the detection of touch points. Exposing the overlay to direct sunlight will flicker touch points on and off and will dramatically interfere with the touch points that need to be detected. It also restricts the further use of IR light, such as for the purpose of blob detection. Another issue is that using optical technologies will detect any object as a touch point, limiting the use of tangibles and paper documents. A solution for those issues is presented in Chapter 4.

For tangible recognition, a video feed is sent to reacTIVision using a Microsoft LifeCam Studio [2], which is mounted overhead to the ceiling. The LifeCam Studio camera allows for zooming, panning and tilting the image to center on the tabletop. This is not accurate enough to get an acceptable position for the tangibles, but the reacTIVision framework provides the option to use a calibration grid on the image feed to correct any positioning issues. Using the LifeCam Studio camera also allows for adjusting image brightness, white balance, contrast, saturation and exposure. We found that some fine-tuning is needed to ensure that reacTIVision consistently tracks fiducials[3]. If the configuration is not optimal, detection of fiducials in certain positions can flicker, triggering tangible entry and removal events very quickly. Reflection can also be an issue: when using a reflective surface, non-consistent lighting in the room can affect detection.

Projection on documents is done using an Optoma Pico PK320 projector[4], which is also mounted overhead, next to the USB camera. To avoid warping of the image, the camera and projector are mounted as close as possible to each other. Calibration of the image is less important here, because slight shifts of position are less disruptive than in tangible detection, in which position is more important.

[2]https://www.microsoft.com/hardware/en-us/p/lifecam-studio/Q2F-00013

[3]Fiducials are predefined patterns that can be recognised in an image in a robust and accurate manner. This is explained in more detail in Section 4.7.

[4]http://arc.optomausa.com/products/detail/Pico-PK320

4.2 Overview

The framework can be represented in a layered view, as illustrated in Figure 4.3. Using the standard layout of data, logic and presentation the components of ADFIS are grouped together in a logical manner.

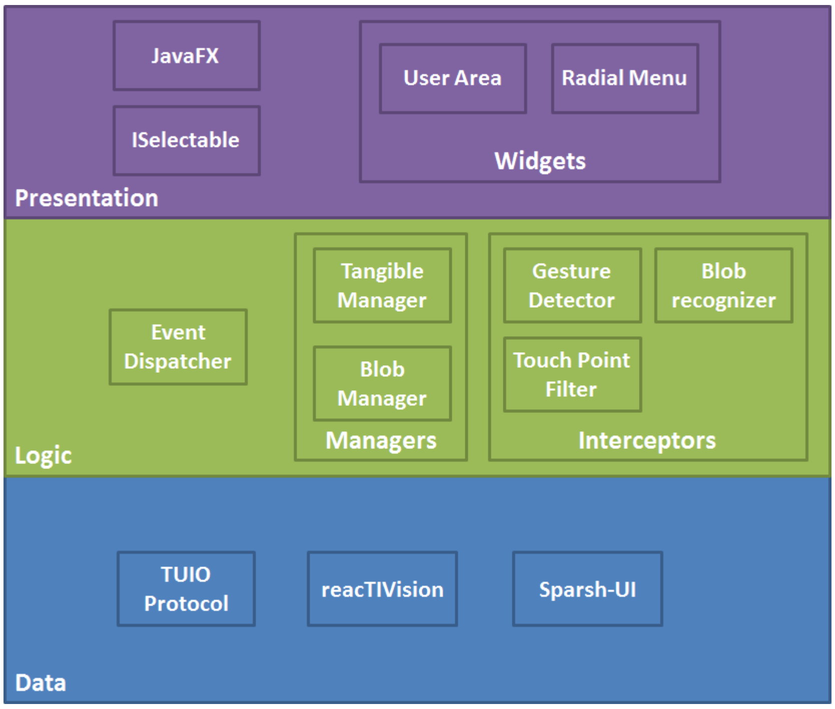


Figure 4.3: Overview of the ADFIS architecture

In the data layer, three components are shown. The TUIO protocol is used to receive touch events from the hardware and is also used to receive tangible events which are dispatched from the reacTIVision framework. The reacTIVision framework dispatches those events by analysing frames originating from a video stream. Sparsh-UI [51] (see Section 4.6 is the gesture recognition component of the framework, which is based on a client-server architecture but runs locally.

In the logic layer, the most important components are the event dispatcher, the managers and the interceptors. The event dispatcher is the central point where input events are sent to and are processed before dispatching. The event dispatcher also finds the correct event target. The event dispatcher is aided by a tangible manager and a blob manager, their function is to maintain a reference to every tangible and blob currently in the system and provide the event dispatcher with information needed for touch point filtering. The touch point filter, gesture detector and blob recogniser are interceptors, they modify the default behaviour of the event dispatcher. The detailed mechanism of the interceptor system is given in Section 4.5.

In the presentation layer, the JavaFX[5] library is used as the facilitating library to create graphical user interfaces. To better provide developers with the necessary abstractions, ADFIS provides some visualisation support on top of JavaFX. The ISelectable interface allows for objects to be selected in the application's context. ADFIS also includes two widgets, the user area and the radial menu. User areas are parts of the surface that are distinct and are designed to accommodate one user. Radial menus are designed to allow for easier menu navigation in multi-touch applications.

4.3 Graphical User Interface Structure

As previously mentioned, a graphical user interface or GUI can be built using JavaFX. JavaFX is a graphics library which is included in the Java programming language starting from JDK/JRE (Java Development Kit/Java Runtime Environment) 7u6 onwards. It is a modern library that provides developers with a Rich Client Platform (RCP) for development of local or web-based applications. JavaFX is designed to provide a lightweight and hardware-accelerated UI platform.

When developing an application using JavaFX, the developer is encouraged to make use of the `Stage-Scene` system. A `Stage` is the name of any window that is created. By default, one `Stage` is created when running any application. More stages can be created programmatically. The `Stage` always shows a `Scene`, which is the container that contains the scene graph (which is explained the next paragraph). Scenes are typically used for one specific view or state of the application. When the application changes state or view, the current scene is replaced by another scene with different UI elements. This is important because accessing the current scene is needed to properly dispatch events. Ignoring this would result in a defective event dispatching system.

JavaFX, like many GUI frameworks, organises its UI elements in a hierarchical structure that can be visualised using a tree diagram. The JavaFX

[5]http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html

event system is based on the use of a *scene graph*. As shown in Figure 4.4, the scene graph system works like any tree, with a root node that has branch nodes that can either be branch nodes themselves or a leaf node.

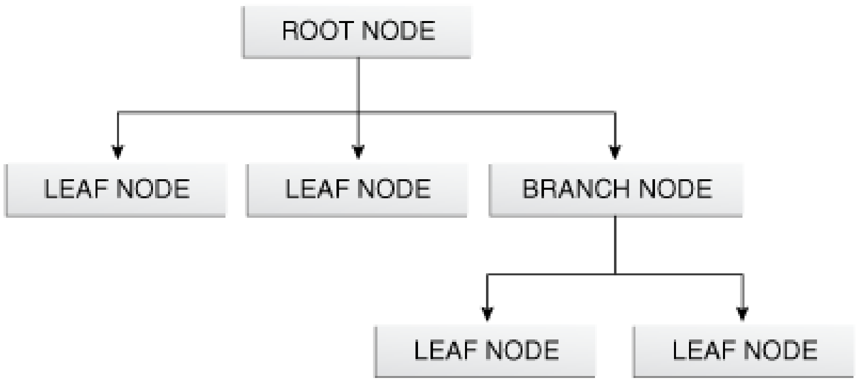


Figure 4.4: Scene graph system
(source: [6])

When put into practice, the scene graph will contain nodes specified by the developer. One of the most commonly used root nodes is the `Group` class[7]. The `Group` class allows components to be placed using an absolute coordinate system. As shown in Figure 4.5, the scene graph presented here has a `Group` as the root node. Two of its branches, the `Circle` and `Rectangle` nodes, are leaf nodes. The `Region` node is an intermediary node that has a `Text` node and an `ImageView` as leaf nodes.

4.3.1 Event System

When events are fired, they are targeted at one of the elements in the scene graph. This process consists of four steps[8], but the essence of this process is that a chain of targets is constructed by chaining all nodes together that intersect with the coordinates of an event. When this event dispatching chain is created, the event travels from the targeted node back to the root node of the scene graph. Each node in the event dispatching chain that has a handler

[6]`http://docs.oracle.com/javase/8/javafx/scene-graph-tutorial/scenegraph.htm`

[7]The others being the `WebView` and `Region` classes. The `WebView` class is used to display web pages, and the `Region` class is a class that has subclasses which manage layout in a more structured manner, less fit for interactive surface interface design.

[8]https://docs.oracle.com/javafx/2/events/processing.htm

for the event can then trigger. Whenever a node is reached, its handler can *consume* the event, meaning that the event will not travel further down the chain to the root node. This system allows for very fine-tuned behaviour to events of different event types.

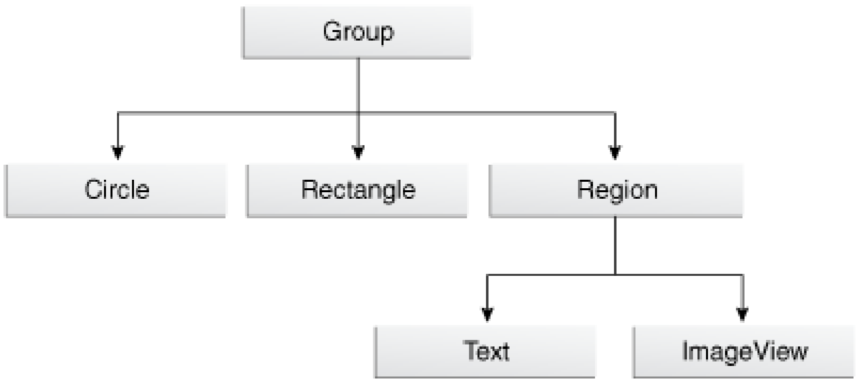


Figure 4.5: Example instantiation of a scene graph (source: [9])

4.4 Multi-touch

As previously mentioned, the multi-touch capabilities of the framework are mainly based on communication using the TUIO protocol in the case that other communication protocols such as Windows 7 touch are absent or disabled.

As shown in Figure A.1 in Appendix A, the framework implements a TUIO listener. This TUIO listener is implemented by the `TUIOConnection` class, which listens to asynchronous events fired by TUIO on the hardware. The `TUIOConnection` class then does two things. First, it translates the coordinates in the TUIO cursor event to coordinates on the screen (touch points are called cursors in the TUIO specification). This is necessary because TUIO cursor coordinates are normalised; they are are floating point numbers between 0 and 1. The `TUIOConnection` scales those coordinates to screen size. When the coordinates are translated, the TUIO cursor event is converted to a JavaFX `TouchEvent`. A new `TouchEvent` is created and its parameters

[9]`http://docs.oracle.com/javase/8/javafx/scene-graph-tutorial/scenegraph.htm`

are assigned[10]. The `TouchEvent` is then sent to the `EventDispatcher` for processing[11].

In the `EventDispatcher` class, a number of preprocessing steps are performed. First, the `BlobManager` will check if the touch point in the `TouchEvent` does not intersect with a blob that is currently detected on the surface. By default, the response of the framework is to interpret this case as an update to the state of the blob. This is because the development hardware implements blobs as a collection of touch points. When different hardware if used that specifies blob detection differently, this behaviour can be changed by modifying the `processTouchEvent` method in the blob manager.
If the blob manager does not claim the touch point, the interceptors have the opportunity to process and/or claim the `TouchEvent`. The inner workings of the interceptor system are detailed in Section 4.5. If neither the blob manager or the interceptors claim the `TouchEvent`, the event is dispatched. Dispatching the event consists of finding and setting the target for the event, and then firing that event to the target.

Finding the event is done through the use of the `getNode` method in the JFXtras library[12]. The JFXtras library contains auxiliary methods that expose parts of the JavaFX library that are otherwise not accessible. The `getNode` method takes as arguments a `Parent` object, x and y-coordinates and a class parameter. It returns the last node that is an implementation of the class parameter, starting the search at the `Parent` object. The class parameter works as a filter for the search. Every element in the scene graph is an instantiation of the `Node` class or a subclass thereof, so providing `Node.class` will return a leaf node in the scene graph (the topmost element on the scene). The workings of this method were necessary to explain to clarify how the location process was adapted in the framework. It is true that `getNode` returns the last implementation of the `Node` class, but the `Group` class is also an subclass of the `Node` class. `Group` objects usually contain more nodes, so it is necessary to recursively apply `getNode` until the result is not a `Group`. When the actual topmost element is located, the `EventDispatcher` fires the event to the targeted `Node` subclass on the scene.

[10]A number of parameters are unused, but they can be set using an interceptor before dispatching if those parameters are required in the context of the application

[11]This dispatcher class is not the same as the `EventDispatcher` type in JavaFX

[12]http://jfxtras.org/

4.5 Interceptor System

In the framework, the event dispatching system's rules for dispatching events can be modified using interceptors. Whenever the dispatcher receives an instruction to process an event, interceptors are notified of the event first. The interceptors then have the choice to retain the event and process it, or release the event to the dispatcher which will dispatch it normally.

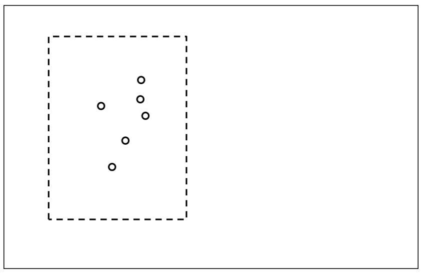
Since interceptors might have a long computation time (such as the blob detector), it is advised to implement interceptors asynchronously, e.g. as a `Runnable` in a separate thread. If computationally heavy interceptors ran on the same thread, they become a bottleneck in the system. There is also a minor issue that releasing events from the interceptors might result in duplicate events being fired by the event dispatcher. A possible solution to this would be to extend the interceptor system with a locking system where releasing the event from each interceptor reduces the lock state of the event, which is then dispatched normally when the lock state is nullified.

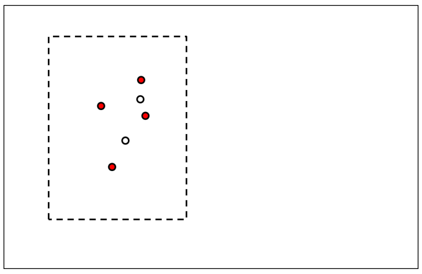
4.5.1 Blob Detection

The prime example of the interceptor system is blob detection. A blob detector will collect touch press events during a certain period of time. Between those intervals, the detector will analyse the touch points that were collected during the last interval. If there are enough touch presses that were not nullified by touch releases, a blob is detected and its properties are calculated such as the center, the dimensions and the outline of the blob. This state is then encapsulated in a blob entry event and sent to the event dispatcher.
The process that allows for blob introduction can be summarised in the following steps:

- The hardware sends touch events over TUIO to the `TUIOConnection`;
- the `TUIOConnection` sends the touch events to the `EventDispatcher` for processing;
- the touch events are intercepted by the `BlobDetector`;
- after a set interval, the `BlobDetector` analyses the contents of its buffer. If a blob was not detected, the touch events are released back to the `EventDispatcher` where they are dispatched normally;
- if a blob was detected, a new `BlobEvent` is created;
- the `BlobEvent` is sent to the `EventDispatcher` for processing;

- the `EventDispatcher` creates a `RegionFilter` that is sent to the `BlobManager`, which provides an ID for the blob. The `RegionFilter` is added to the `BlobDetector` so touch events that intersect with the blob region are intercepted and will update the blob state;
- the `BlobEvent` is dispatched to its target.


(a) A paper document creates a number of touch points.


(b) Non-extreme points are filtered.

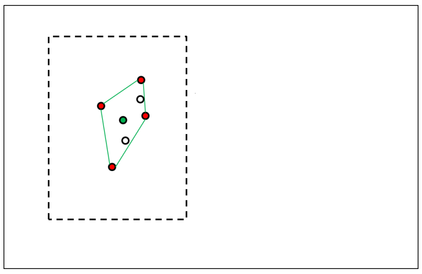

(c) A polygon represents the blob's shape. The center point is calculated and used as the blob's position.

Figure 4.6: The blob creation process

The `BlobEvent` that is dispatched encapsulates a `BlobState` object, which represents the state of the blob. This state is largely based on the parameters of a blob in TUIO. The blob state contains blob coordinates, the *inner state* of the blob (entry, stationary, update, removal), the shape of the blob in the form of a `Polygon` and a list of touch points. The coordinates of the blob are determined by calculating the extreme points in the set of touch points, from which a midpoint is calculated which coordinates are set as the blob's coordinates. Because of this method of coordinate calculation, a minimum of three touch points is required to detect a blob. This method of blob detection is a circumvention of the limitations of the hardware. The blob detection is parametrised; both the number of touch points that is required to consider a blob and the detection frame in which touch points are collected can be modified programmatically. This process is illustrated graphically in Figure 4.6.

4.5.2 Tangibles Accommodation

Another type of interception that is enabled by default is the filtering of touch points beneath tangibles. Because of the non-capacitive touch technology in the hardware, any object that touches or is in extreme proximity of the development surface will trigger a touch point. A system is in place that makes sure that tangibles do not interfere with touch interactions and more importantly, gestures. As mentioned before the event dispatcher can be modified with interceptors. By default, when processing an event the dispatcher will also have the tangible manager check whether a touch point does not intersect with a tangible. This is done by checking for each tangible if it intersects with a certain radius around the tangible. This radius can be modified on a case-by-case basis, but has a default value.

4.6 Gestures

As previously mentioned, the framework implements gesture recognition through the use of the Sparsh-UI framework. In this section we motivate our choice to use Sparsh-UI as opposed to MT4j, another framework written in Java, and we detail the use of Sparsh-UI.

4.6.1 Motivation

MT4j[13] (Multi-Touch for Java) is an open source framework for the development of multi-touch applications. MT4j's main purpose is to speed up the development of multi-touch applications. It provides high level functionality in the form of a toolkit.

[13]http://www.mt4j.org

The MT4j framework implements its functionalities by propagating TUIO events and other input events such as mouse and keyboard to a unified input system which abstracts from the hardware. On the presentation side, MT4j has a custom approach that is reminiscent of how JavaFX works, which is the technology used in this thesis on the presentation side.

Similarly to Tactive, MT4j provides a proper basis for the development of multi-touch applications. On the flip side, it also has the same limitations of Tactive (but it does support tangibles). Multi-touch applications are a subset of interactive surface applications. The project also seems to be abandoned, as the last update to its source code was in December 2012[14]. It also has some gesture limitations such as restricting zoom gestures to one.

Gesture recognition is implemented using the Sparsh-UI framework [51][15], a cross-platform (C++ and Java) framework for multi-touch and was developed at Iowa State University's Virtual Reality Applications Center (VRAC). The project's development seems to have halted since the last modification to the source code was in 2010[16], but the framework is functional and readily available.

Sparsh-UI features gesture recognition though a client-server architecture. The server component is the core component of the framework, it receives touch data as input and transmits touch points and recognised gestures. It supports a set of basic gestures which can be extended by sub classing the `Gesture` class (both in C++ and Java). Clients connect to the server and they are either input sources or Sparsh-UI clients. Input sources transmit touch data. The reception of this touch data is broadcast to clients which can then *claim* the data and mark themselves as destination for touch points or gestures. Sparsh-UI clients can then receive gesture messages and process them according to the application's rules.

Because of this client-server architecture, it is possible for the server to run in Java while the clients are a mix of Java and C++, since communication is done over TCP/IP which also opens up the possibility of distributed applications. In ADFIS, this client-server architecture is circumvented by running the server and clients locally, but it is possible to undo this with minimal effort. By design, the Sparsh-UI integration is very loosely coupled.

[14]https://code.google.com/p/mt4j/source/list
[15]https://code.google.com/p/sparsh-ui/
[16]https://code.google.com/p/sparsh-ui/source/list

As demonstrated in the section on usage of gestures in the framework (Section 5.4), the gesture detector is an interceptor that is added to the event dispatcher.

4.7 Tangibles

As previously mentioned, tangibles are supported through the use of the reacTIVision framework [31]. The reacTIVision framework is a framework for tangible interactions. Created by the same authors as the TUIO protocol, the reacTIVision framework makes use of TUIO for communication between input devices and applications. The framework consists of a standalone application that processes a real-time video stream. Processing the video stream consists of performing visual pattern recognition, which can be done quickly and robustly. The visual patterns that are recognised are predetermined patterns called fiducial markers or fiducials. A more common example of fiducials is the QR-code, which is a block-shaped pattern that can also contain information such as a hyperlink. There are many types and variants of fiducials. The fiducials used by reacTIVision are amoeba-shaped, as they consist of a blob-shaped ellipse containing circles and dots, reminiscent of amoebas. Examples of these amoeba-shaped fiducials are shown in Figure 4.7.

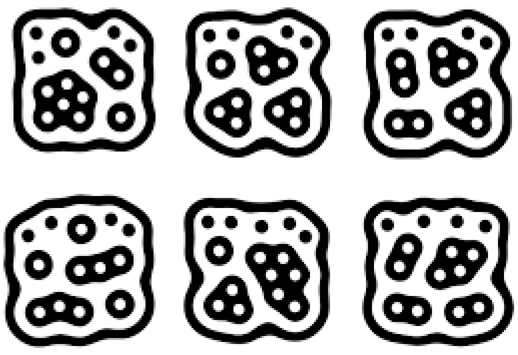


Figure 4.7: Fiducial markers used by reacTIVision
(source: `http://reactivision.sourceforge.net`)

The use of reacTIVision requires a video feed. The optimal hardware platform for this video feed is through the use of a camera inside the interactive surface, which means that the patterns are face down on the surface, which opens up more opportunities to combine tangibles with projection. In our hardware setup we did not have access to such a technology, so we resorted to using the fiducials face up and used a USB camera to register the markers.

ReacTIVision sends `Tuiobject` events over TUIO's default port on UDP, which is listened to by a `TuioConnection` in ADFIS. TUIO object events are processed analogous to how touch events are processed before dispatching. The coordinates detected by reacTIVision are normalised, so a transformation to the application's UI dimensions are applied. It is also possible that the video feed's coordinate space is not equal to the application logic, therefore coordinates can be mirrored around the X or Y-axis before dispatching. When sent to the dispatcher, the dispatcher has the opportunity to handle the events before dispatching, similarly to the touch interceptor system. By default, the dispatcher will filter out touch points that overlap with an estimate area of the tangible at its detected location.

5 ADFIS Functionality

In this chapter the functionalities that are included in ADFIS are detailed. First, in Section 5.1, we discuss the graphical user interface component of ADFIS that can be used to aid development. Second, we detail the multi-touch functionality in Section 5.3, followed by presenting Section 5.4 on the framework's gestures functionality. After this, Section 5.5 details the tangibles functionality. Finally, Section 5.6 details the functionality involving paper documents. In each section, sample code is included that demonstrates how an application built using the framework can make use of its features.

5.1 Graphical User Interface

Rather than implementing its own UI framework, ADFIS makes use of the JavaFX framework to allow for UI development. Some facilitating support is offered by ADFIS which is not included in JavaFX. The following sections explain why the choice was made to integrate JavaFX in the framework as opposed to the older, more supported Swing framework and why JavaFX has an edge over Swing because of its feature set, supported technologies and ease of use.

5.1.1 Discussion

ADFIS uses JavaFX as an enabling technology. By implementing the hardware abstractions on top of JavaFX, developers can implement their applications using the standard way of working in JavaFX. Only a few method calls are required to make use of the hardware abstractions of ADFIS. If needed, more functionality of the individual components of ADFIS can be accessed. ADFIS also includes other visualisation support build on top of JavaFX in the form of widgets (user area and radial menu) and the ISelectable interface. Those elements are detailed in the previous chapter and their use is demonstrated in the next section.

JavaFX has many features that make it beneficial to develop user interface with. One of the major features is JavaFX's integration with web technologies. Amongst those technologies is a web rendering engine that allows for mixing of Java features and web technologies. It also features FXML, which is an XML syntax used to script user interfaces and allows the the use of CSS (Cascading Style Sheets) to style the interface of an application. The support of CSS allows for separation of function and style, as well as adaptive layouts for different screen sizes (made popular by its use in cross-device development of web applications, but this is also relevant for interactive surfaces).

Another key point of JavaFX is its event-driven character. In JavaFX, user input is handled by triggering events of a specific type which is then dispatched to the relevant UI component. The software components of this event system are reused and extended in the framework to allow for dispatching and receiving of multi-touch, gesture and blob events.

JavaFX is still fairly new. Its initial release was in 2008[1] and is still in active development, although it has received some major updates over the years. It is possible to argue that it is better to develop graphical Java applications in Swing[2], the graphics library that is also included in Java. Swing is considerably older than JavaFX and has many third-party libraries, so Swing might include some features that JavaFX does not. JavaFX' modern approach to GUI's and integration of web technologies still make it the library of choice for development of GUIs in the context of interactive surfaces.

[1]http://www.oracle.com/technetwork/java/javafx/1-0-140175.html
[2]http://docs.oracle.com/javase/tutorial/uiswing/

5.1.2 Sample Code

In listing 5.1, the code is given that makes up a very basic JavaFX application. Every implementation of a JavaFX application needs to extend the `Application` class that is found in the JavaFX package. By extending `Application`, a class is required to implement the inherited abstract method `void start(Stage primaryStage)`, which is called when the application is ran. The sample application creates a window in the form of a `Stage` object that is shown with its initial `Scene`[3].
Some IDEs or development environments require that the `main` method calls `launch(args)`, so it is included for sake of completeness.

```java
public class SampleApplication extends Application {
    private final Group root = new Group();
    private double width = 1920;
    private double height = 1080;

    @Override
    public void start(Stage primaryStage) throws Exception {
        Scene scene = new Scene(root, width, height);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Listing 5.1: Basic code for a JavaFX application

5.2 Other Visualisation Support

ADFIS contains a few other functionalities that support user interface implementation: support for element selection and the radial menu, an example of a widget.

5.2.1 Selectable Components

Selection of one or more UI components is an interaction method that is ubiquitous. In the context of interactive surface UIs, selection is possible in a certain territorial context such as a window or a user area. JavaFX supports selection by providing the `SelectableNode` interface, but this does not

[3]More information about how different views are organised in JavaFX can be found at docs.oracle.com/javafx/

include support for user areas. As such, we have updated selection functionality by providing the `ISelectable` interface. This interface is constructed to allow for selection of components in a user area, in such a way that each user area has a different selection.

In listing 5.2, an example is provided that shows the implementation of a selectable rectangle by combining the `Rectangle` class with the `ISelectable` interface. In this example, the `SelectableRectangle` is implemented in such a way that reception of a touch press calls the `setSelected` method (line 17). Calling this method will set a border for the rectangle, providing visual feedback to the user and sets the `selected` boolean to `true`. Other behaviour is context sensitive: on line 27 the method `onObjectSelected` is called, which processes this event. A default handler would set selection on other components to `false`, but more complex interactions can be defined by the developer.

```
 1  public class SelectableRectangle extends Rectangle implements ISelectable {
 2      private boolean selected = false;
 3      private UserArea area;
 4
 5      public SelectableRectangle() {
 6          this.setOnTouchPressed(new EventHandler<Event>() {
 7
 8              @Override
 9              public void handle(Event event) {
10                  SelectableRectangle.this.setSelected(true);
11                  event.consume();
12              }
13          });
14      }
15
16      @Override
17      public void setSelected(boolean value) {
18          if (value) {
19              this.setStroke(Color.BLACK);
20              this.setStrokeWidth(10);
21              this.selected = true;
22          } else {
23              this.setStroke(null);
24              this.selected = false;
25          }
26          if (selected) {
27              area.onObjectSelected(this);
28          }
29      }
30      ..
31      @Override
32      public void setContext(UserArea area) {
33          this.area = area;
34      }
35
36  }
```

Listing 5.2: Use of the ISelectable interface

5.2.2 Radial Menu

Widgets are UI components that are pre-constructed and available to developers. ADFIS currently does not have a library of widgets, but implementing them is relatively easy (depending on the complexity), and by implementing them once a lot of time for future development is saved. As an example of a widget, ADFIS contains a radial menu. A radial menu is a menu that has its elements organised in a circle. This has a distinct advantage over traditional drop-down menus: all elements in a radial menu are equally close to the center of the menu, which is usually also the location of the touch points. Selecting elements in a drop-down menu using touch can be more difficult since elements in a drop-down menu will have a different distance from the touch point's location. The most common varieties of radial menus are either donut shaped or have bubbles representing elements. ADFIS uses the bubble layout, because in a such a layout the elements are separated which ensures that releasing a touch point is never overlapping two options, as would be the case when using a donut layout.

```
    double x = 500;
    double y = 600;
    MenuItem accept = new MenuItem();
    MenuItem decline = new MenuItem();

    RadialMenu menu = new RadialMenu(new MenuItem[] {accept, decline});
    menu.setCenterX(x);
    menu.setCenterY(y);
```

Listing 5.3: Using a radial menu

In listing 5.3, an example of the use of a radial menu is shown. Before initialising a radial menu, menu items must first be created. They can then be provided to the constructor of the radial menu. A menu item is implemented as a `Node`, which allows them to have event handlers. The suggested use is to set the handler with the desired function in the `Node.onTouchReleased` method, but other event handlers can be used as well. When creating a radial menu, the elements will be automatically spaced over the available space in the menu. The radius will also depend on the number of elements, which ensures that there are no graphical problems when using a radial menu.

5.3 Multi-touch

Multi-touch capabilities are included in the framework in two forms. Hardware platforms that support multi-touch on the level of the operating system trigger touch events that can be received by JavaFX directly. All other platforms that support the TUIO standard will have their touch events converted to JavaFX events, so developers do not need to distinguish between events.

The supported touch events are entry, update and removal.

5.3.1 Sample Code

Reusing the code from Section 5.1, an example is given of how touch events can be used in an application in listing 5.4. In this listing, an event handler is set for the `Scene` object that is currently used by the application. When the `Scene` object receives a touch press event, the background will be coloured black.

```
 1 public class SampleApplication extends Application {
 2     ..
 3     @Override
 4     public void start(Stage primaryStage) throws Exception {
 5         ..
 6         scene.setOnTouchPressed(new EventHandler<Event>() {
 7             @Override
 8             public void handle(Event event) {
 9                 scene.setFill(Color.BLACK);
10                 event.consume();
11             }
12         }
13 
14     }
15 }
```

Listing 5.4: Processing touch events

On line 10 the event is also consumed. Event consumption dictates that a certain object that is an implementation of the `Node` class has processed the event, and that no other GUI element will receive the event. JavaFX dispatches its events to the topmost node in the scene graph from the root down, and then works its way back to the root element of the scene graph. In other words, the topmost element will receive the event first, and the root of the scene graph will be the last to receive it. Whenever an element consumes an event, that event will stop making its way back to the root element.

The code sample discussed above is purely JavaFX code and will work in many scenarios, on the condition that there is a software component that dispatches touch events. For instance, from Windows 7 onward, the operating system has multi-touch support built-in[4] which allows developers to implement multi-touch interactions just by adding event handlers. However,

[4]http://windows.microsoft.com/en-us/windows7/products/features/touch

Windows 7 touch is not open source. The default set of gestures can not be extended using source code and its default set of gestures is fairly rigid e.g. a long press is interpreted as a right click which might not be desirable behaviour in some applications.

It is possible that there is no software component present in the operating system that dispatches touch events. In that case it suffices that the hardware platform supports the TUIO protocol to make use of the framework. This is demonstrated in listing 5.5.

```
1  public class SampleApplication extends Application {
2      final Group root;
3      ..
4      @Override
5      public void start(Stage primaryStage) throws Exception {
6          ..
7          EventDispatcher dispatcher = new EventDispatcher(root);
8          TUIOConnection tuio = new TuioConnection(dispatcher);
9          tuio.connect();
10         ..
11     }
12 }
```

Listing 5.5: Processing TUIO touch events

In this sample we can see that an `EventDispatcher` and `TUIOConnection` object are utilised. The TUIO connection is an object that implements a TUIO listenern which starts listening for TUIO events on line 9 using the `connect()` method. The TUIO connection is responsible for converting TUIO events to JavaFX events and is linked to the event dispatcher. The event dispatcher will determine where the events are sent, and adhere to specific rules with regard to grouping, filtering and preprocessing when there are applicable. This is explained in-depth in chapter 4.

5.4 Gestures

Gestures are implemented using the gesture server from Sparsh-UI. Since Sparsh-UI is written in Java, it was possible to re-purpose those components. More information about this is implemented can be found in Section 4.6. The framework currently provides developers with four gestures: rotate, zoom, swipe and double tap. Gestures can be added by extending the`GestureEvent` class found in JavaFX and by dispatching those gesture events from a gesture recognition component such as Sparsh-UI. Sparsh-UI only supports a number of events, so adding a gesture without other gesture recognition software would require adding the recognition code to the Sparsh-UI gesture server software as well.

5.4.1 Sample Code

In the code sample below, a simple application is given that demonstrates the use of the rotate gesture event. Rotate events received the the rectangle component will translate to a rotation of the rectangle.

```java
public class SampleApplication extends Application {
    private final Group root;
    private Rectangle rec = new Rectangle(400, 300); // width & height

    @Override
    public void start(Stage primaryStage) throws Exception {
        ..
        rec.setOnRotate(new EventHandler<Event>() {
            @Override
            public void handle(Event event) {
                RotateEvent rotate = (RotateEvent) event;
                rec.setRotate(rec.getRotate() + rotate.getAngle());
                event.consume();
            }

        });
        rec.setX(100);
        rec.setY(100);
        root.getChildren.add(rec);
        ..
        EventDispatcher dispatcher = new EventDispatcher(root);
        GestureDetector gestureDetector = new GestureDetector();
        gestureDetector.startSparshUI();
        dispatcher.addInterceptor(gestureDetector);
        TUIOConnection tuio = new TuioConnection(dispatcher);
        tuio.connect();
        ..
    }
}
```

Listing 5.6: Processing gesture events

It is important to note that the code sample presented uses a shortcut to handle the rotate event. JavaFX has built-in support for four gesture types, which have a corresponding method that is somewhat shorter. To demonstrate the use of a gesture that is supported by SparshUI but not JavaFx, the following code snippet is given. The `DoubleClickEvent` class was added to the framework and instances are converted from SparshUI and dispatched by the gesture detector interceptor.
Note: setting the background of a scene object should be ran on the JavaFX thread, this is omitted.

```java
public class SampleApplication extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        ..
        scene.addEventHandler(DoubleClickEvent.DOUBLE_CLICK, new
            EventHandler<Event>() {

            @Override
            public void handle(Event event) {
                scene.setFill(Color.BLACK);
            }
        });
        ..
    }
}
```

Listing 5.7: Processing gesture events not native to JavaFX

5.5 Tangibles

The framework also supports tangibles by supporting reacTIVision markers, which can be applied to any tangible. These markers are fiducials that are recognised using the reacTIVision application. This application takes a video feed such as from a webcam and searches for distinct, well-defined patterns. ReacTIVision fiducials have a position and a rotation that can be used in an application. The supported tangible events are entry, update and removal. The use of reacTIVision is detailed in Section 4.7.

5.5.1 Sample Code

The code in listing 5.8 demonstrates how tangibles can be used in an application. In this code sample, a `Rectangle` object is created and added to a `Group` object which serves as the root node for the scene graph. An event handler is added which handles tangible entry events. Tangible events contain the state of a tangible, from which the position and rotation of the tangible can be accessed. Here the rotation of the rectangle is set to the rotation of the tangible.

```
public class SampleApplication extends Application {
    private final Group root;
    private Rectangle rec = new Rectangle(50, 50); // width & height

    @Override
    public void start(Stage primaryStage) throws Exception {
        ..
        scene.addEventHandler(TangibleEvent.ADDED, new EventHandler<
            TangibleEvent>() {
            @Override
            public void handle(TangibleEvent event) {
                TangibleState state = event.getState();
                double rotation = event.getRotation();
                rec.setRotate(rotation);
                event.consume();
            }
        }
        rec.setX(100);
        rec.setY(100);
        root.getChildren.add(rec);
        ..
        EventDispatcher dispatcher = new EventDispatcher(root);
        TUIOConnection tuio = new TuioConnection(dispatcher);
        tuio.connect();
        ..
    }
}
```

Listing 5.8: Processing tangible events

Since reacTIVision uses TUIO as a method of communication, it is also required to create a TUIO connection and event dispatcher. However, a TUIO connection can and should only be initialised once, it handles touch and tangible events simultaneously.

5.6 Paper Documents

Paper documents are supported in the form of blob recognition. In its current form, the framework can recognise blobs by collecting and filtering touch points. When certain criteria are met, touch points will be grouped into a blob (this process is explained in Section 4.5.1). Blob events encapsulate a blob state, which contains the position, the list of touch points in the blob and a polygon representation of the blob. The supported blob events are entry, update and removal which correspond with the detection of paper, moving of paper and removal of paper respectively.

Currently, it is the responsibility of the application to distinguish between blob event types; the developer needs to make sure that events coming from paper documents are handled as such. It is possible to accommodate this as done in the demo application (see chapter 6) using a fiducial.

5.6.1 Sample Code

Listing 5.9 demonstrates how blob events can be processed. In this sample, an event handler is added to the scene that retrieves the shape of a blob and displays it in the form of a polygon.

```java
public class SampleApplication extends Application {
    private final Group root;

    @Override
    public void start(Stage primaryStage) throws Exception {
        ..
        scene.addEventHandler(BlobEvent.BLOB_ADDED, new EventHandler<
            BlobEvent>() {

            @Override
            public void handle(BlobEvent event) {
                BlobState state = event.getState();
                BlobRegion region = dispatcher.getBlobManager().getRegion(
                    state.getId());
                root.getChildren().add(region.getShape());
            }
        });
        ..
    }
}
```

Listing 5.9: Processing blob events

Since blob detection is based on touch points, it is also required to create a TUIO connection and event dispatcher. This is shown in listing 5.10.

```java
public class SampleApplication extends Application {
    private final Group root;

    @Override
    public void start(Stage primaryStage) throws Exception {
        ..
        dispatcher = new EventDispatcher(root);
        double detectionFrame = 50;
        BlobDetector detector = new BlobDetector();
        detector.setDetectionFrame(detectionFrame);
        Thread t = new Thread(detector);
        t.setDaemon(true);
        t.start();
        dispatcher.addInterceptor(detector);
        tuio = new TuioConnection(dispatcher);
        tuio.connect();
        ..
    }
}
```

Listing 5.10: Setup of blob detection

This code sample requires some explanation. The event dispatcher is the same as used in previous code samples in this chapter. As mentioned before, this dispatcher can be modified to interpret event differently before dispatching them. This is done using an interceptor model: a number of interceptors have the chance to handle the events before they are dispatched to UI elements. Blob detection is quite intensive to compute, thus a daemon is created that does these computations in the background. This is explained in more detail in chapter 4.

5.7 User Areas

User areas are a component of the framework that allows interactions to be separated from each other. They are a well-defined areas of an interactive surface that support any interaction that can be performed on the whole surface and also has the potential to support inter-area interactions.

5.7.1 Discussion

The framework provides developers with user areas, which are UI elements that function as interaction areas which are separated but not isolated from the entire surface. This entails that while a user while mostly interaction with the UI components in their user area, interactions between user areas are still possible. An example of this would be to share information represented by a UI component by dragging that component to another users' interaction area. In a user area, interactions such as selection of objects and gestures are not ambiguous, they do not interfere with other users' interactions as would be the case when multiple user actions are allowed simultaneously in the same interaction area.

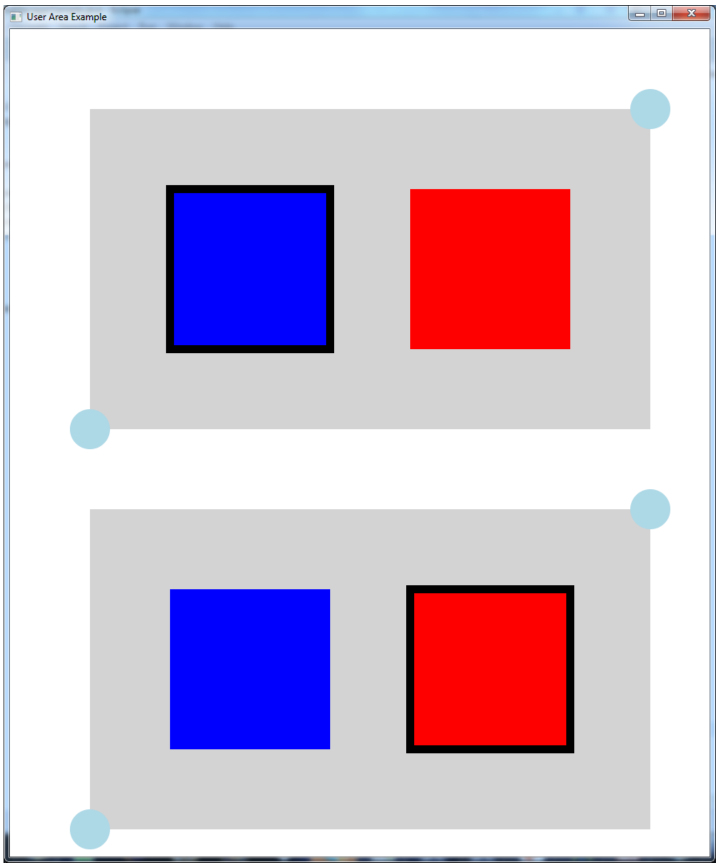


Figure 5.1: An application demonstration the separate selection contexts of two user areas

An example of the use of user areas is shown in Figure 5.1. Two user areas are shown with handles for repositioning and resizing. Inside the user areas, two rectangles are displayed. Both pairs of rectangles can be selected, which triggers a black outline as a visual cue. Selection of either rectangle in one user area will not interfere with another user area, providing developers with a tool to group and separate UI contents for a specific user.

5.7.2 Sample Code

User areas are easily used by simply defining their dimension and position, and adding them to the root node of the scene graph as would be done with any JavaFX node. As many user areas can be added as needed.

```
public class SampleApplication extends Application {
    private final Group root;

    @Override
    public void start(Stage primaryStage) throws Exception {
        ..
        UserArea area = new UserArea();
        area.setX(100);
        area.setY(100);
        area.setWidth(700);
        area.setHeight(400);
        root.getChildren().add(area);

        scene = new Scene(root, 1920, 1080);
        primaryState.setScene(scene);
        ..
    }
}
```

Listing 5.11: Using a user area

Being implemented as a sub class of the JavaFX `Group` class, adding components to a user area is exactly the same as adding components to the root of the scene graph.

```

        ..
        UserArea area = new UserArea();
        Rectangle rectangle = new Rectangle();
        Circle circle = new Circle();

        area.getChildren().add(rectangle);
        area.getChildren().add(circle);
        ..
```

Listing 5.12: Adding components to a user area

5.8 Projection

Projection on documents is used in Chapter 6, where a projected image overlays on top of a paper document. ADFIS supports projection, but does not provide abstractions to accomplish this. In fact, projection is considered to be no different from displaying the UI on a monitor or external display such as the TV used in the development hardware.

As mentioned in Section 5.1, JavaFX makes use of a the `Stage-Scene` system, where stages correspond to windows and scenes contain UI elements. To project an image, it suffices to create another `Stage` that contains the elements which need to be projected. The newly created `Stage` will need to be displayed on the correct output display, which is the responsibility of the developer.

5.9 Summary

In this chapter we presented the feature set of ADFIS and demonstrated their use with code samples consisting of small applications that focus on one feature. The features of ADFIS consist of the most important interaction methods of interactive surfaces, support for paper document interactions and visualisation support on top of the JavaFX framework.

Applications have a strong integration with the JavaFX framework, which allows UI components to receive events of every interaction method type. Visualisation support in ADFIS also includes user areas, radial menus and selectable components. User areas provide separate areas of the surface that allow for a separate interaction contexts for any interaction methods. Selection of components is also supported.

ADFIS supports multi-touch interactions, gesture interactions, tangible interactions and paper document interactions. Touch events can be received over the Windows 7 touch protocol or over the TUIO protocol. Gesture events can be enabled or disabled as needed. Tangibles are supported through support of the reacTIVision framework. Paper documents can be recognised by performing blob detection and provides the application with a shape and location of the document.

6 Demo Application

In this chapter we present the demo application that was built using ADFIS to demonstrate its capabilities.

6.1 Scenario

The general idea of the demo application was to create a small proof-of-concept that demonstrates as many features of ADFIS as possible, but the application should also consist of a practical example that can be used in a real world situation. Although this is a challenging objective, we managed to construct a scenario in which nearly every feature of ADFIS is demonstrated in a meaningful way. The scenario of the demo application is given in the following paragraphs:

> “A user approaches the interactive surface and starts interacting with it by putting a tangible on the surface. This tangible has a fiducial marker on it, which the surface recognises. As soon as the tangible is recognised, the surface displays an interaction area that is personal to that user (it could — for instance — be the same interaction area that they used at work). In the personal

> interaction area, the user places a physical document.
>
> A second user approaches the interactive surface and puts their tangible on the surface. The second user is also accommodated by the surface with their personal user area. The two users begin discussing the contents of the document on the surface. The first user wishes to share their document with the second user. By pressing a *broadcast* button, the second user's area receives a notification that another user is sharing content. The second user can now press their user area, spawning a radial menu that allows them to receive the content or dismiss the notification. Accept the broadcast causes the document to be transferred to the second user's area; the document is copied to the user context of the second user and is shown in their user area as a digital copy. This digital copy can be interacted with using zoom and rotate gestures.
>
> The second user annotates their digital copy and would like to share their annotations. The second user presses a *share* button, which causes the annotations to be shared with the first user. The annotations are now projected on the document of the first user."

This is a scenario that would be fairly time-consuming to implement completely. As such, the components that are not features of ADFIS are simulated with placeholder functionality, since they are the responsibility of the developer to implement and do not contribute anything to the demonstration of ADFIS. Some of those components could be the subject of future work and/or extensions to the framework. In order of use: the user area linking to the tangible is hard-coded; the annotations on the document are contained in an image that is linked to the digital document and copying the file is purely visual and does not actually transfer a file.

6.2 Demonstration

Initially, the surface is blank, devoid of any content or UI elements. The first interaction is user one placing a tangible on the surface. This tangible is detected and recognised, triggering a tangible entry event in the framework.

As can be seen in Figure 6.1, the application responds to this event by creating a user area element at the location of the tangible. The tangible now functions as a handle for the user area; repositioning the tangible will also reposition the user area. The user area can be resized with another handle located at the opposite corner of the tangible. User interactions can now be distinct based on whether they are inside the user area or not.

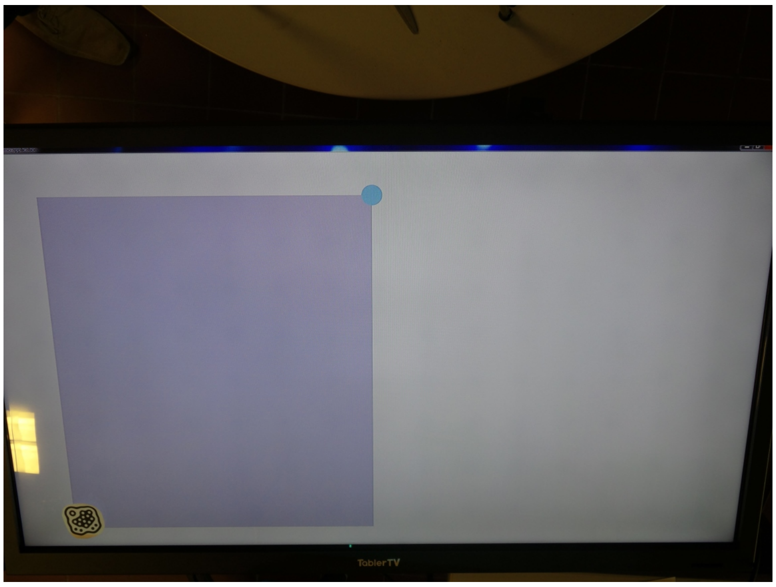


Figure 6.1: State of the application after introduction of a tangible

After the introduction of the user area, a paper document is placed on the surface, inside the user area. The document used in this demo is the second page of Wellner's DigitalDesk [73] paper. Introducing this paper sheet triggers a blob entry event and is detected as a number of touch presses in a specific time frame. This is detailed in Section 4.5.1. As can be seen in Figure 6.2, a rectangle is added to show the detection area of the paper sheet, which has the exact dimensions and position of the area in which touch points are regarded as part of the blob. The paper document can be moved around, but this may not track the document accurately. Along with the rectangular region, a button is shown at the bottom of the region with a broadcast icon. This broadcast functionality was inspired by [23], in which the authors uses broadcast functionality to communicate between different devices such as tablets. Tablets and user areas are comparable in the sense that they usually belong to one user and support multi-touch interactions.

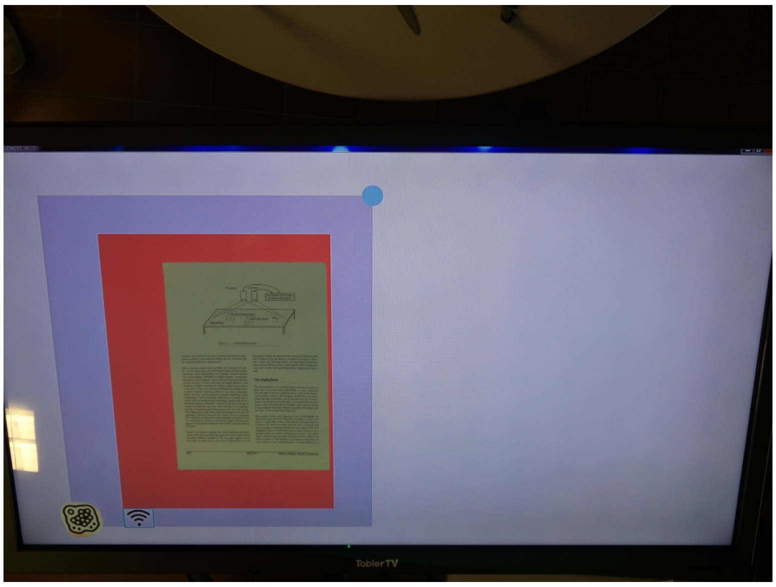


Figure 6.2: Introduction of a paper document is detected

A second tangible is then introduced that spawns a second user area, in the same way that the first tangible creates its own user area. This is shown in Figure 6.3.

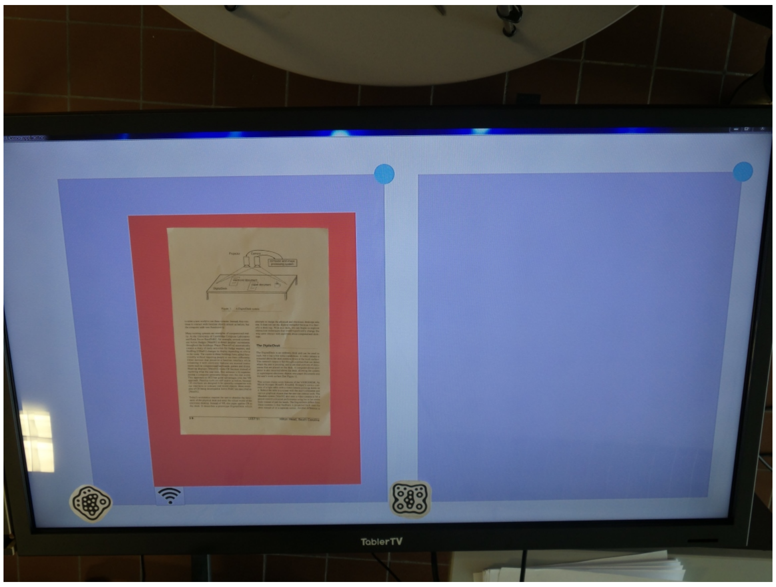


Figure 6.3: A second user area is created

Pressing the broadcast button will now add a notification to the second user area that the first user is sharing the DigitalDesk document. When a touch press is detected in the second user area, the application will create a radial menu that has two options: accept or decline. This is shown in Figure 6.4. If the touch point moves to either option, it will set the option as selected with visual feedback.

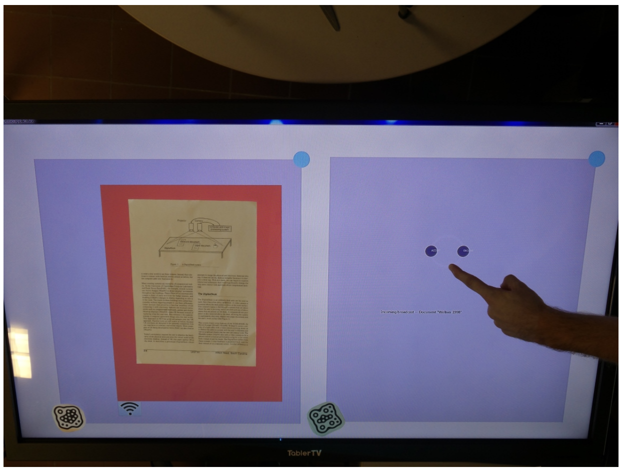


Figure 6.4: A radial menu is created with an accept and decline option

If the touch point is released on top of an option, the radial menu option will trigger its listener that listens to touch release events. The *accept* option is selected, which creates and adds a view of the DigitalDesk document shown in Figure 6.5.

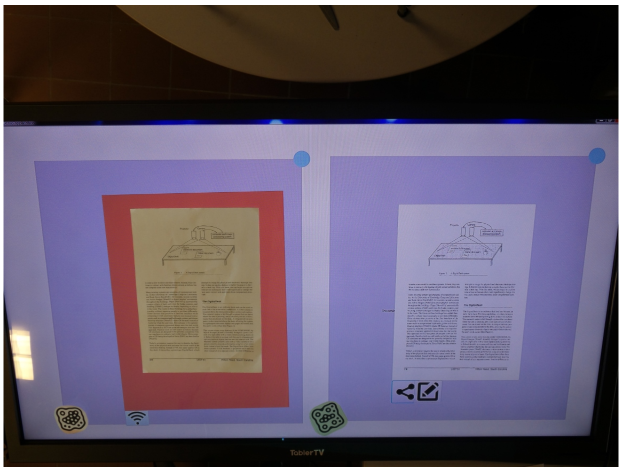


Figure 6.5: A digital copy of the document is created

The view of the DigitalDesk document supports gesture interaction. Zooming and rotating the document is possible using the commonly used two finger multi-touch gestures: moving touch points closer and further respectively decreases and increases the size of the document; moving touch points in a clockwise or counter-clockwise motion rotates the documents accordingly. Included in the UI component that contains the view of the DigitalDesk document is an *annotation* button. Pressing this button allows for the addition of annotations to the document[1]. This is shown in Figure 6.6.

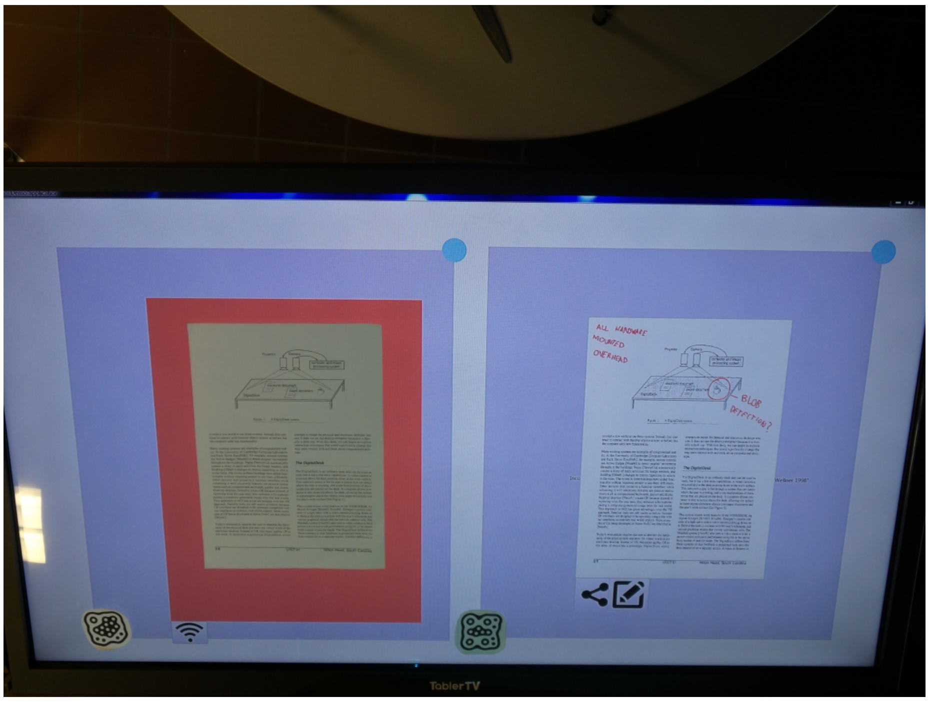


Figure 6.6: The digital copy is annotated

Finally, the annotation can be shared by pressing the *share* button also contained in the view of the document. This transfers the user-created annotation to the original user area. On reception of this transfer, the paper document is augmented using the overhead projector. The overhead projector overlays the annotations on the document.

[1]As explained in Section 6.1, this is mocked.

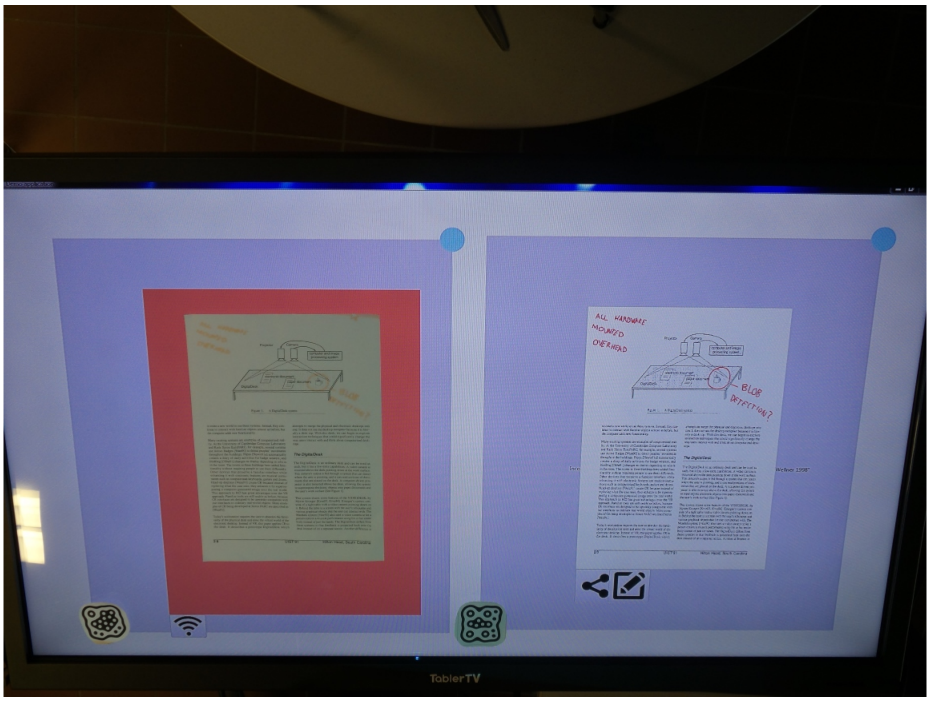


Figure 6.7: Sharing the annotations causes an overlay to be projected on the physical document

6.3 Findings

From the implementation of the demo application certain conclusions can be drawn about how well ADFIS has facilitated development. The basis for those conclusions is comparing the development of the application with ADFIS against a situation where the provided abstractions were not available. For evaluation purposes, an identical technical environment is assumed, development would have to be done using the same hardware.

The use of tangibles in the application was implemented by adding event listeners to certain UI components that are registered to listen to tangible events. Without using ADFIS, this could have been achieved by also using the reacTIVision framework or another comparable framework, but the developer would have to invest time in implementing abstractions for handling the raw input events coming from the reacTIVision framework by implementing a TUIO listener. Implementing a TUIO listener would also have to take into account conversion of normalised coordinates and possible coordinate space inversions in the case that the registration hardware's axes are different from the application's. If non-capacitive touch was used, filtering of touch points underneath tangibles would also have to be implemented.

Implementing multi-touch functionality has become a fairly standard practise in interactive surfaces, which is reflected by its number of available frameworks, discussed in Section 2.1. The multi-touch features in the demo application could have easily been implemented in a different framework, which is preferable to implementing a TUIO listener. When opting to choose Windows 7 touch, developers would be locked into using Microsoft technologies (.NET framework, WPF, etc). However, since multi-touch is just one of the functionalities of ADFIS, the greatest advantage of using ADFIS was the ease of combining different interaction methods. Using different frameworks for each interaction method would require an equal number abstractions to be implemented, or to follow the example of ADFIS and unify the input events in an event dispatching system where event listeners are generic and can be parametrised depending on the interaction method.

In the demo application, gestures were supported by adding gesture event listeners to UI components. It is however necessary to create a gesture detector, start detection and add it as an interceptor to the event dispatcher. Gesture recognition is not enabled by default as it is done in a separate thread, and obscuring this would hinder the developer. Implementation of gestures without ADFIS is possible, but it is recommended to use one of the existing gesture frameworks as they can be time-consuming to implement, especially when allowing simultaneous gestures. The use of Sparsh-UI allows a decent set of basic gestures to be used and offers good extensibility. During prototyping, we found that basic gesture interaction such as a combination of dragging, zooming and rotating work as desired. However, in the implementation of the demo application, rotation and zoom events do not work perfectly. The events are received but are not handled. This is likely because of the fact that Sparsh-UI dispatches events for all enabled gestures as soon as touch data is processed, independently of the value of the gesture (e.g. rotation events can be dispatched with an angle of zero degrees) causing processing issues. A filter that only lets through non-zero events might alleviate this problem.

Because of the hardware platform and the method used, detection of paper documents is one the more problematic features of the demo application. The detection of a paper document relies on the speed of entry of the document, which is not completely reliable due to the hardware's touch point detection technology. Sometimes the entry speed is too low, and the paper sheet's touch points are not grouped. This can also cause problems when multiple users perform touch presses at the same time, although this could be mitigated by adding area constraints to the detection algorithm. Moving a paper sheet is less problematic and translates the movement of the paper fairly accurately, although sometimes touch points are only detected at the edge of the paper sheet. This causes the detected position of the paper sheet to be at the edge of the sheet as well. A possible solution to this would be to add a probabilistic component to the tracking of movement. During prototyping we found that detecting smaller blobs and objects with a more solid surface is less problematic due to the fact that they produce touch points that are closer to each other. Their behaviour when moving the object is also more stable. We have argued before that although non-capacitive touch makes this form of paper document tracking possible, capacitive touch in combination with computer vision achieved with detection inside or beneath the surface is optimal.

Considering that some applications include the notion of territoriality, the availability of user areas in ADFIS is a feature that can save developers a lot of time. Since they are well-integrated in JavaFX, user areas can be treated as any building block of the UI. During the development of the demo application, the use of user areas proved to be very easy to separate touch and gesture interaction from other parts of the surface. Although not used in the demo application, the added support for selectable components also provides developers with much flexibility.

7 Conclusion

In this chapter we conclude this dissertation by summarising its contents. The findings from related work are listed again and were used to substantiate the claim that a framework such as ADFIS are a contribution to the research field of interactive surfaces. Additionally, other contributions are also restated.

7.1 Contribution

Over the course of the previous chapters a number of contributions were made of which the implementation of the ADFIS framework is the most significant, but other contributions were made as well. A short summary is provided below, after which they are discussed briefly in the next sections.

- We created an overview of the research field of interactive surfaces by reviewing related work starting from its origins to modern research, covering various important topics.
- We created a hierarchical classification for interactive surfaces based on hardware characteristics.
- We engineered requirements for a framework for a subset of the classification which is satisfied by ADFIS, a framework for application de-

velopment for interactive surfaces which we implemented.

- We investigated the integration possibilities of several interaction methods.
- We implemented a paper document detection algorithm, achieved with blob detection for IR-based optical imaging technology.

7.1.1 Overview

In the overview of the research field we established that interactive surfaces can be very distinct. They usually have different hardware systems and use different technologies although specialised frameworks or libraries such as TUIO provide some abstraction. Many systems use multi-touch and support different interaction methods complementary to touch interactions. We also concluded that the use of paper documents is not diminishing as predicted, which entails that paper documents are an interaction method that should also be taken into account.

In the introductory chapter, we discussed the problem that application development for interactive surfaces is being hindered by the need to implement hardware interfaces and software components that could be reused from other projects. The most important finding from investigating literature is that software support for interactive surfaces is limited to specific components. Therefore we proposed that there is a need for a framework that provides developers with better support on a higher abstraction level.

7.1.2 Hierarchical Classification

Due to the fact that hardware platform for interactive surfaces are very distinct we propose a hierarchical classification based on the characteristics of the interactive surface. Those characteristics are physical in nature and allowed us to define which systems ADFIS could be used for and what systems it does not support or could potentially support with some modifications or extensions.

7.1.3 ADFIS

In terms of provided features, the feature set of ADFIS includes support for all interaction methods on interaction surfaces, with the exception of the stylus. We have included a discussion regarding stylus- and pen-based interaction in Section 8.4 in the chapter on future work. However, to our knowledge, there is no other framework with the same set of features. More important than the support of individual features is the support of combining them in an application by unifying interaction methods in an event dispatching system that allows the developer to easily implement listeners for those event on different parts of the UI. ADFIS is easy to use yet is still flexible enough to allow developers to add or change functionality as needed.

7.1.4 Interaction Method Integration

Since ADFIS supports several interaction methods, we learned that some interaction methods are more easily integrable than others. This depends on which specialised frameworks are used, they can have a different implementation language or use different communication methods. Some restrictions also apply on the physical level. The use of IR-light for touch detection entails that IR-based approaches cannot be used for other purposes, such as IR-based blob detection. Furthermore, we experimented with using computer vision for paper document detection (which is discussed further in Section 8.1.2 but since we already used reacTIVision, the overhead camera's stream was not available for computer vision.

7.1.5 Paper Document Detection Algorithm

During development of ADFIS we observed that using non-capacitive touch technologies can be an issue when supporting physical objects on the surface of the system. Specifically, physical objects can be detected by the touch technology as being touch points instead of blobs or tangibles. Tangibles on the surface clearly interfere with the touch detection as they should only be registered as a tangible. Paper documents and other objects provided us with the opportunity to keep blob detection in the surface as opposed to using other techniques such as computer vision and IR-detection.

The algorithm that we propose is based on the temporal and spatial aspects of touch points. In summary, during a given interval a collection of touch points can be recognised as a blob if the collection is of a certain size. The detection frame and collection size can be parametrised, although large detection frames will introduce a latency in the detection. From the detected collection of touch points, a blob is recognised and its position and outline is calculated, providing a region of the surface that can intercept touch events. The detailed workings of this algorithm are explained in Section 4.5.1.

7.2 Evaluation of ADFIS

To properly evaluate ADFIS a user study should be performed which compares the use of ADFIS versus using only specialised frameworks and libraries. Such as study was performed by the authors of Tactive [22]. Due to time constraints this was not possible. Instead, we evaluated ADFIS by implementing the demo application in Chapter 6. Other aspects not covered in the discussion of the demo application are covered in the next paragraphs.

Since the application possibilities of interactive surfaces are numerous, extensibility is an important factor of ADFIS. This is achieved in multiple ways. First, the event dispatching system allows for the addition of new interaction methods. It suffices to create a new event type for event listeners. Events of that type then need to provide access to encapsulated state of an interaction method. Second, the interceptor system allows an application to modify the default behaviour of a component of ADFIS, although intercepting events is less relevant to other interaction methods than multi-touch.

The performance of the system is also relevant for the overall quality of the system. During the development of the framework we used several prototype applications as a test bed. We experienced no noticeable input lag when developing a single function, nor are there noticeable delays in the demo application, where multiple interaction methods are combined. Minimal delays are achieved by the use of multi-threading. Other than the UI running on a separate thread, which ensures UI responsiveness, interceptors can be run on a different thread than the application's main thread. This is the case with gesture recognition, and is recommended when using blob recognition, since increasing the detection frame causes delays between introduction and detection of a blob.

8 Future Work

In this chapter we discuss the future possibilities of ADFIS. Those future possibilities include improvements to individual components, as well as integrating the entire framework in other software solutions.

8.1 Improvements

In this section we analyse the different components of ADFIS for which technologies exist that could replace and improve the current functionality of the relevant component.

8.1.1 Gestures

Gesture recognition is currently implemented using Sparsh-UI. Sparsh-UI provides a set of basic gestures that can be extended, but this requires a programmatical approach to gesture definition. Other gesture recognition technologies exist that approach gesture definition differently.

Proton++ [36], GeForMT [34] and Midas [61] are frameworks that replace the programmatical approach to gestures with a descriptive approach. In GeForMT, gestures are defined using short descriptions in a gesture syntax.

Proton++ is a declarative framework where gestures are defined using regular expressions. Midas is also a declarative framework that uses a prolog-like language to define gestures. All of the aforementioned frameworks offer more expressiveness and flexibility regarding definition of gestures and would be hugely beneficial to developing applications that can benefit from complex gestures. Important to note is that using a declarative approach might not contribute to lowering development time as there might be a learning curve involved in using the declarative syntax.

To better support interactive surface applications, Cirelli and Nakamura [11] defined an extended set of requirements that should be met by gesture recognition frameworks. These requirements were based on previous work done by Kammer et al. [34]. Although the authors did not evaluate every framework or gesture recogniser that they mentioned in their work, they included Midas, Proton++, $N-Protractor [3] and MT4j in their comparison using the proposed requirements. They found that Midas meets 11 of the 16 requirements, failing only spatial invariance and easy prototyping. Accuracy was not evaluated due to a lack of data. Other frameworks did not meet at least half of the proposed requirements.

8.1.2 Paper Documents

ADFIS currently supports paper document interaction through blob detection, which only provides positional information about a document, and distinguishes between different documents. This could be extended in a few ways. First, the information that is accessible to the developer can be extended to also include rotation, as is the case with tangibles. Second, the information about the document can also be extended to include information about the content of the document, which implies that documents are no longer just blobs with a certain identification number.

As previously mentioned in Section 2.2, different technologies exist that allow for identification of documents. When trying to identify documents, the main issue is clear: robust tracking requires the identification component to know what it is searching for (i.e. pattern recognition), or it requires a complex computer vision algorithm that can identify the shape and content of a document.

Pattern Recognition

Pattern recognition can be done by familiarising the identification component with the document. We have discussed this before when considering [35], in which the system has a database of images to use as patterns. When using such a technique, we propose a less strict limitation: it should be required to have a digital copy of a document in a format such as PDF, and recognition features should be generated automatically using the same algorithm that extracts features from the captured image.

Due to its dependence on assumed knowledge of documents, using pattern recognition is limiting. The alternative is using computer vision to discover paper documents. The clear advantage here is that paper can be used without any sort of modification or augmentation. The problem here is that the identification algorithm needs to perform multiple tasks. It needs to discover documents and identify them, which is not an easy task and requires much programming effort.

Computer Vision

We attempted to achieve document discovery using OpenCV[1], but encountered some significant problems. Our approach was to use background subtraction to identify moving shapes. The algorithm used was the Improved Adaptive Gaussian Mixture Model [77], referred to by OpenCV as MOG2 (Improved Mixture of Gaussians). After the background was subtracted, shape recognition was applied to the resulting image, searching for rectangular shapes of a certain size. This was done using OpenCV's edge detection and polygon approximation on the detected edges. At several stages there were issues. The background subtraction subtracted too much from the image, detecting only the rough edges of the sheet. Changing the parameters of the algorithm - threshold and learning rate - did not improve the detection quality. Since they are also moving with respect to the background, a user's hands were also detected.

During these experiments we also found that moving UI elements are detected as foreground, which is problematic. A possible solution for this would be to refresh the background model if the UI changes, but doing so would require large scale changes to the UI framework. Shape detection was also problematic since there was no clear way of removing user limbs from the image, which shifts the problem towards recognising and filtering limbs from

[1]http://www.opencv.org

the image before paper sheets could be detected. For the issues mentioned in this section, we abandoned the computer vision approach.

Optical Character Recognition

Optical Character Recognition (OCR) is the name given to a number of technologies that translate printed text to digital text through mechanical or electronic means. OCR dates back as far as 1870 and has known several generations of improvement [18]. Nowadays, OCR technologies are available through a number of frameworks, of which the Tesseract engine[2] is the most accurate and has the advantage of being open source [43]. It is also used by other frameworks as their recognition engine.

OCR can be used in the framework to improve the detection of paper documents, provided that the contents of the document contain printed text (or even handwritten text, but this requires different algorithms). Depending on the algorithm used - such as Tesseract - OCR can be applied even if the source image is rotated. By recognising text, it also becomes possible to identify the document on top of detecting its location, which is a huge advantage.

8.2 Extensions

Some components or features of ADFIS are functional but offer the possibility of extension. In this section we discuss some of the functionalities of ADFIS that can be extended.

8.2.1 Near Field Communication

Some of the applications and interactive surfaces in chapter 2 support the use of devices that contain Near Field Communication (NFC) technology. NFC is a group of technologies and communication protocols that allows for communication between devices over short distances, typically 5 to 10 centimetres. The typical use of NFC is allow smartphones to act as communication point between the user and the system, or between users. Previously mentioned in Section 2.1.4, smartphones are used for file transfers and payments. User identification can also be an application of NFC in interactive surfaces.

[2]https://code.google.com/p/tesseract-ocr/

8.2.2 Territoriality

User areas in their current implementation separate areas from the interactive surface. Currently, a user area is limited to being distinct from another area in that it supports selection, grouping of touch points and the ability to perform gestures separately. Although this opens up a wide variety of possibilities, this is still limited. The shape of the user area is rectangular, which might not be the optimal shape when the interactive surface has to accommodate many users, or when the hardware platform is shaped differently than a tabletop or wall-mounted display. We refer back to the BendDesk mentioned in Section 1.2, which would not be fully compatible with the current implementation of user areas.

Inter-area interactions are currently the responsibility of the developer. It would prove useful to developers if communication between user areas was support in predefined inter-area interaction methods such as drag-and-drop interactions. User areas are currently also identical in function. We suggesting investigation of the use of heterogeneous user areas such as common and personal areas. Moving beyond one hardware device, it is possible that user areas are no longer contained to one specific interactive surface. Exploration of the possibility of distributed user areas would also prove useful to developers.

8.2.3 Multiple Screen Support

ADFIS allows for multiple monitors to be used, limited only by the capabilities of the graphics card(s) used the hardware on which it is ran. However, there is no support for using multiple screens, either for linking them together to create one large display or using multiple monitors to create an interactive environment ran on one instance of the framework.

8.3 Integration

Since interactive surfaces can exist in an environment that contains more devices or other surfaces, the possibility exists for integration of ADFIS in an interactive context. An example of an interactive environment is given in Section 2.1.4, where a presentation by Samsung shows the combination of a tabletop with a large wall projection. Interactions on either the tabletop or the interactive wall affect the state of the other component of the environment. This opens up a wide array of possibilities, but would require some sort of protocol or messaging model that allows for easy and generic sharing of content between devices.

Another integration possibility would be to allow smartphone interactions. Using standard Wi-Fi, a communication model could be designed that allows for multi-user interaction. For example, interactive presentations could be given using a vertical interactive surface, where the audience can be interacted with by asking multiple choice questions which the audience can respond to by submitting an answer.

8.4 Pen-based Interactions

A subject that has not been previously discussed is the use of stylus interactions. Historically, this has been one of the interaction methods used in interactive surfaces, dating back as far as DigitalDesk (see Section 2.1). However, in modern interactive surfaces and devices, the stylus seems to be disregarded as an interaction method and replaced by multi-touch. This is most likely due to the fact that single touch interactions could be performed with a stylus or finger, but multi-touch is a very expressive interaction method, and its expressiveness is comparable to the mouse. A stylus only has the advantage of being more precise and consequently, more fit to input text or drawings. However, in most mobile and interactive surface applications, inputting text or drawings is not a common input method. In those applications, multi-touch and virtual keyboards are more efficient.

Although the stylus as an interaction method is less used, the digital pen has become an interesting new input method which partially replaces the stylus. Digital pens are ballpoint pens that are augmented using a micro-controller and an integrated camera. As presented in [65], augmenting paper with a pattern of tiny dots provides a grid structure that allows the integrated camera to track the pen's position. This grid structure does not interfere with the contents of the page, it merely gives the paper sheet a gray tint. Certain areas of the paper can be interpreted by an application to trigger a certain function. In Signer and Norrie's PaperPoint system, this is used for various presentation interactions such as random access to specific slides. The positional tracking is used to annotate in real-time on both the physical document and the slide currently being presented.

In the demo application for ADFIS (see Chapter 6), annotations are added to a digital copy of a document, and projected back onto a physical copy. Although it is not impossible to conceive a system that automatically updates a physical document in a permanent fashion, it would certainly require specialised hardware. The other way around, adding content to a physical copy and synchronising this with a digital copy can easily be done using the digital pen described in this section. In the context of an interactive surface, modifying a paper document would most likely occur either on or in the close proximity of the surface. In PaperPoint, the digital pen uses Bluetooth for short-to-medium range wireless communication, meaning distance would not be a limiting factor.

A
Sequence Diagrams

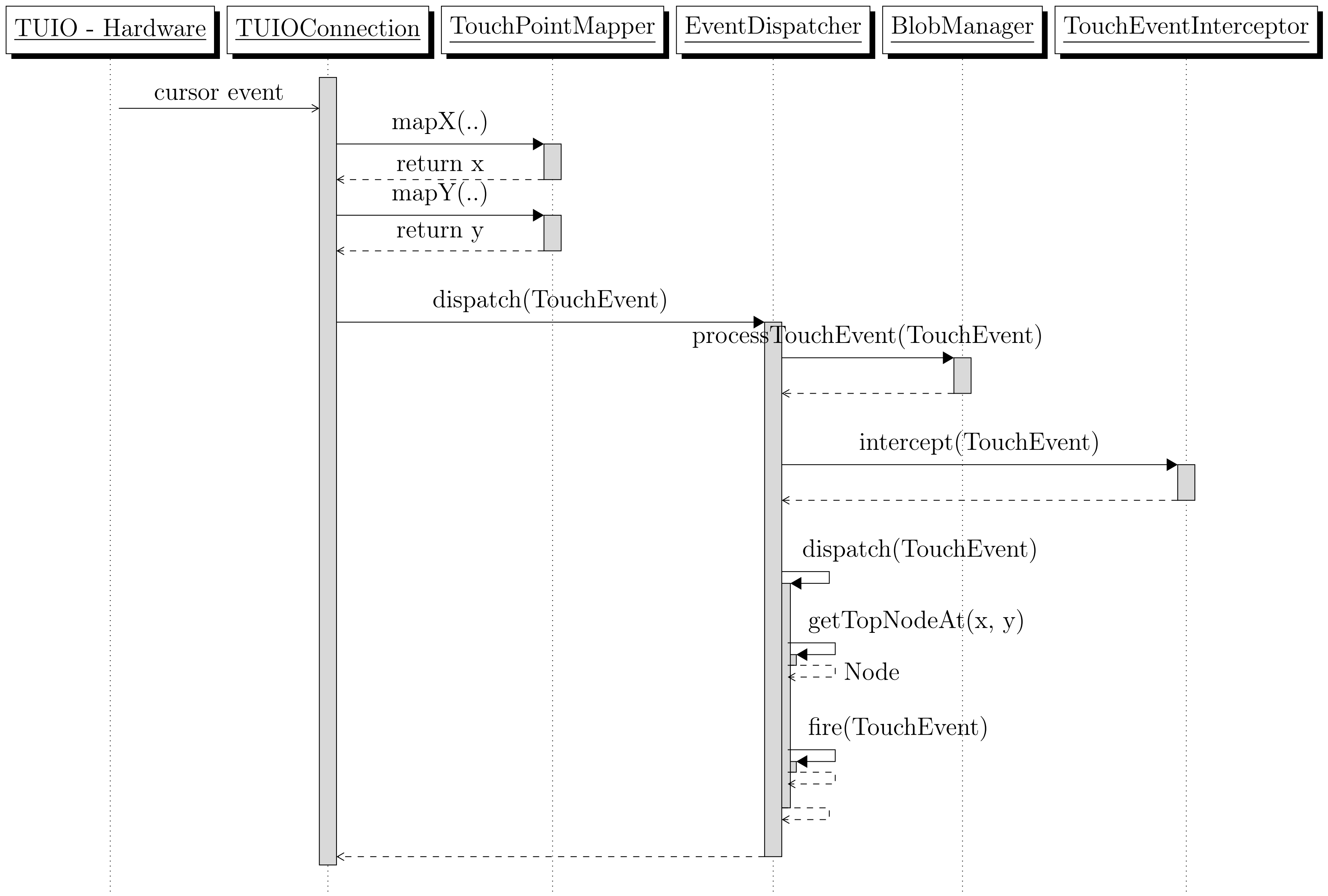


Figure A.1: Sequence diagram of touch event processing from hardware to event dispatching

Bibliography

[1] qTUIO. Retrieved from: qtuio.sirbabyface.net. Last accessed: July 29, 2015.

[2] Michelle Annett, Tovi Grossman, Daniel Wigdor, and George W. Fitzmaurice. Medusa: A Proximity-Aware Multi-Touch Tabletop. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, Santa Barbara, CA, USA, October 16-19, 2011*, pages 337–346, 2011.

[3] Lisa Anthony and Jacob O. Wobbrock. $N-protractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of the Graphics Interface 2012 Conference, GI '12, Toronto, ON, Canada, May 28-30, 2012*, pages 117–120, 2012.

[4] Thomas Augsten, Konstantin Kaefer, René Meusel, Caroline Fetzer, Dorian Kanitz, Thomas Stoff, Torsten Becker, Christian Holz, and Patrick Baudisch. Multitoe: High-Precision Interaction with Back-Projected Floors Based On High-Resolution Multi-Touch Input. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology, New York, NY, USA, October 3-6, 2010*, pages 209–218, 2010.

[5] Sriram Karthik Badam and Niklas Elmqvist. PolyChrome: A Cross-Device Framework for Collaborative Web Visualization. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces, ITS 2014, Dresden, Germany, November 16 - 19, 2014*, pages 109–118, 2014.

[6] Nikola Banovic, Frank Chun Yat Li, David Dearman, Koji Yatani, and Khai N. Truong. Design of Unimanual Multi-Finger Pie Menu Interaction. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2011, Kobe, Japan, November 13-16, 2011*, pages 120–129, 2011.

[7] Hrvoje Benko. Beyond Flat Surface Computing: Challenges of Depth-Aware and Curved Interfaces. In *Proceedings of the 17th International Conference on Multimedia 2009, Vancouver, British Columbia, Canada, October 19-24, 2009*, pages 935–944, 2009.

[8] Matthew Blackshaw, Anthony DeVincenzi, David Lakatos, Daniel Leithinger, and Hiroshi Ishii. Recompose: Direct and Gestural Interaction with an Actuated Surface. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Extended Abstracts Volume, Vancouver, BC, Canada, May 7-12, 2011*, pages 1237–1242, 2011.

[9] Harry Brignull, Shahram Izadi, Geraldine Fitzpatrick, Yvonne Rogers, and Tom Rodden. The Introduction of a Shared Interactive Surface into a Communal Space. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW 2004, Chicago, Illinois, USA, November 6-10, 2004*, pages 49–58, 2004.

[10] Pizza Hut Chaotic Moon Studios. Interactive Concept Table, March 2014. Retrieved from: https://www.youtube.com/watch?v=xvT0MCugb58. Last accessed: July 22, 2015.

[11] Mauricio Cirelli and Ricardo Nakamura. A Survey on Multi-touch Gesture Recognition and Multi-touch Frameworks. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces, ITS 2014, Dresden, Germany, November 16 - 19, 2014*, pages 35–44, 2014.

[12] Peter Dalsgård and Kim Halskov. Tangible 3D Tabletops. *Interactions*, 21(5):42–47, 2014.

[13] Chi Tai Dang and Elisabeth André. A Framework for the Development of Multi-Display Environment Applications Supporting Interactive Real-Time Portals. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS'14, Rome, Italy, June 17-20, 2014*, pages 45–54, 2014.

[14] Alessandro De Nardi. Gesture Recognition mAnagement Framework for Interactive Tabletop Interfaces. Master thesis, University of Pisa, December 2008.

[15] Paul H. Dietz and Darren Leigh. DiamondTouch: a Multi-User Touch Technology. In *UIST*, pages 219–226, 2001.

[16] Pierre Dillenbourg and Michael Evans. Interactive Tabletops in Education. *I. J. Computer-Supported Collaborative Learning*, 6(4):491–514, 2011.

[17] Florian Echtler and Gudrun Klinker. A Multitouch Software Architecture. In *Proceedings of the 5th Nordic Conference on Human-Computer Interaction 2008, Lund, Sweden, October 20-22, 2008*, pages 463–466, 2008.

[18] Line Eikvil. OCR - Optical Character Recognition, 1993.

[19] Michael A Evans and Jesse Jay LM Wilkins. Social Interactions and Instructional Artifacts: Emergent Socio-Technical Affordances and Constraints for Children's Geometric Thinking. *Journal of Educational Computing Research*, 44(2):141–171, 2011.

[20] George W. Fitzmaurice, Hiroshi Ishii, and William Buxton. Bricks: Laying the Foundations for Graspable User Interfaces. In *Human Factors in Computing Systems, CHI '95 Conference Proceedings, Denver, Colorado, USA, May 7-11, 1995.*, pages 442–449, 1995.

[21] Food and Agriculture Organization of the United Nations. 2013 Global Forest Products Facts and Figures, 2014. Retrieved from: `http://www.fao.org/forestry/statistics/80938/en/`. Last accessed: August 22, 2015.

[22] Ombretta Gaggi and Marco Regazzo. *Tactive*, a Framework for Cross Platform Development of Tabletop Applications. In *WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies, Volume 2, Barcelona, Spain, 3-5 April, 2014*, pages 91–98, 2014.

[23] Peter Hamilton and Daniel J. Wigdor. Conductor: Enabling and Understanding Cross-Device Interaction. In *CHI Conference on Human Factors in Computing Systems, CHI'14, Toronto, ON, Canada - April 26 - May 01, 2014*, pages 2773–2782, 2014.

[24] Jefferson Y. Han. Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology, Seattle, WA, USA, October 23-26, 2005*, pages 115–118, 2005.

[25] Jefferson Y Han. Multi-Touch Interaction Wall. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Emerging Technologies, Boston, MA, USA - July 30 - August 3, 2006*, page 25, 2006.

[26] Richard Harper and Abigail J Sellen. *The Myth of the Paperless Office.* MIT Press, Cambridge, MA, USA, 2001.

[27] Steven E. Higgins, Emma Mercier, Elizabeth Burd, and Andrew Hatch. Multi-Touch Tables and the Relationship with Collaborative Classroom Pedagogies: A Synthetic Review. *I. J. Computer-Supported Collaborative Learning*, 6(4):515–538, 2011.

[28] Ideum. GestureWorks. Retrieved from: http://www.gestureworks.com. Last accessed: July 18, 2015.

[29] Hiroo Iwata, Hiroaki Yano, Fumitaka Nakaizumi, and Ryo Kawamura. Project FEELEX: Adding Haptic Surface to Graphics. In *SIGGRAPH*, pages 469–476, 2001.

[30] Yvonne Jansen, Thorsten Karrer, and Jan O. Borchers. MudPad: Tactile Feedback and Haptic Texture Overlay for Touch Surfaces. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, Saarbrücken, Germany, November 7-10, 2010*, pages 11–14, 2010.

[31] Martin Kaltenbrunner and Ross Bencina. reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction. pages 69–74, 2007.

[32] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. TUIO - A Protocol for Table Based Tangible User Interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation GW 2005*, pages 1–5, 2005.

[33] Dietrich Kammer, Mandy Keck, Georg Freitag, and Markus Wacker. Taxonomy and Overview of Multi-Touch Frameworks: Architecture, Scope and Features. In *Engineering Patterns for Multi-Touch Interfaces 2010. A workshop of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, June 2010.

[34] Dietrich Kammer, Jan Wojdziak, Mandy Keck, Rainer Groh, and Severin Taranko. Towards a Formalization of Multi-Touch Gestures. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, Saarbrücken, Germany, November 7-10, 2010*, pages 49–58, 2010.

[35] Jiwon Kim, Steven M. Seitz, and Maneesh Agrawala. Video-Based Document Tracking: Unifying Your Physical and Electronic Desktops. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, Santa Fe, NM, USA, October 24-27, 2004*, pages 99–107, 2004.

[36] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton++: A Customizable Declarative Multitouch Framework. In *The 25th Annual ACM Symposium on User Interface Software and Technology, UIST '12, Cambridge, MA, USA, October 7-10, 2012*, pages 477–486, 2012.

[37] Daniel Klinkhammer, Markus Nitsche, Marcus Specht, and Harald Reiterer. Adaptive Personal Territories for Co-located Tabletop Interaction in a Museum Setting. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2011, Kobe, Japan, November 13-16, 2011*, pages 107–110, 2011.

[38] Daniel Leithinger and Hiroshi Ishii. Relief: A Scalable Actuated Shape Display. In *Proceedings of the 4th International Conference on Tangible and Embedded Interaction 2010, Cambridge, MA, USA, January 24-27, 2010*, pages 221–222, 2010.

[39] Roberto Martínez Maldonado, Andrew Clayphan, Christopher James Ackad, and Judy Kay. Multi-Touch Technology in a Higher-Education Classroom: Lessons In-the-wild. In *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures - the Future of Design, OZCHI '14, Sydney, New South Wales, Australia, December 2-5, 2014*, pages 220–229, 2014.

[40] Nicolai Marquardt, Johannes Kiemer, and Saul Greenberg. What Caused That Touch?: Expressive Interaction with a Surface Through Fiduciary-Tagged Gloves. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, Saarbrücken, Germany, November 7-10, 2010*, pages 139–142, 2010.

[41] Nicolai Marquardt, Johannes Kiemer, David Ledo, Sebastian Boring, and Saul Greenberg. Designing User-, Hand-, and Handpart-Aware Tabletop Interactions with the TouchID Toolkit. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2011, Kobe, Japan, November 13-16, 2011*, pages 21–30, 2011.

[42] Tobias Meyer and Dominik Schmidt. IdWristbands: IR-based User Identification on Multi-Touch Surfaces. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, Saarbrücken, Germany, November 7-10, 2010*, pages 277–278, 2010.

[43] Ravina Mithe, Supriya Indalkar, and Nilam Divekar. Optical Character Recognition. *International Journal of Recent Technology and Engineering (IJRTE)*, 2:72–75, 2013.

[44] Christian Müller-Tomfelde and Morten Fjeld. *Tabletops: Interactive Horizontal Displays for Ubiquitous Computing*, volume 45, pages 78–81. 2012.

[45] Takashi Nagamatsu, Masahiro Nakane, Haruka Tashiro, and Teruhiko Akazawa. Multi-Push Display using 6-axis Motion Platform. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces, ITS 2014, Dresden, Germany, November 16 - 19, 2014*, pages 65–68, 2014.

[46] Kosuke Nakajima, Yuichi Itoh, Takayuki Tsukitani, Kazuyuki Fujita, Kazuki Takashima, Yoshifumi Kitamura, and Fumio Kishino. FuSA Touch Display: A Furry and Scalable Multi-touch Display. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, pages 35–44, New York, NY, USA, 2011. ACM.

[47] Robert J. Orr and Gregory D. Abowd. The Smart Floor: A Mechanism for Natural User Identification and Tracking. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems, CHI Extended Abstracts '00, The Hague, The Netherlands, April 1-6, 2000*, pages 275–276, 2000.

[48] James Patten, Hiroshi Ishii, Jim Hines, and Gian Pangaro. Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. In *Proceedings of the CHI 2001 Conference on Human Factors in Computing Systems, Seattle, WA, USA, March 31 - April 5, 2001.*, pages 253–260, 2001.

[49] Roman Rädle, Hans-Christian Jetter, Nicolai Marquardt, Harald Reiterer, and Yvonne Rogers. HuddleLamp: Spatially-Aware Mobile Displays for Ad-hoc Around-the-Table Collaboration. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces, ITS 2014, Dresden, Germany, November 16 - 19, 2014*, pages 45–54, 2014.

[50] Raf Ramakers, Davy Vanacken, Kris Luyten, Karin Coninx, and Johannes Schöning. Carpus: A Non-Intrusive User Identification Technique for Interactive Surfaces. In *The 25th Annual ACM Symposium on User Interface Software and Technology, UIST '12, Cambridge, Massachusetts, USA, October 7-10, 2012*, pages 35–44, 2012.

[51] Prasad Ramanahally, Stephen Gilbert, Thomas Niedzielski, Desirée Velázquez, and Cole Anagnost. Sparsh-UI: A Multi-Touch Framework for Collaboration and Modular Gesture Recognition. In *ASME-AFM 2009 World Conference on Innovative Virtual Reality*, pages 137–142. American Society of Mechanical Engineers, 2009.

[52] Matthias Rauterberg, Morten Fjeld, Helmut Krueger, Martin Bichsel, Uwe Leonhardt, and Markus Meier. BUILD-IT: a planning tool for construction and design. In *CHI 98 Conference Summary on Human Factors in Computing Systems, CHI '98, Los Angeles, California, USA*, pages 177–178, 1998.

[53] Sarah E Reed, Oliver Kreylos, Sherry Hsi, Louise H Kellogg, Geoffrey Schladow, M. Burak Yikilmaz, Heather Segale, Julie Silverman, Steve Yalowitz, and Elissa Sato. Shaping Watersheds Exhibit: An Interactive, Augmented Reality Sandbox for Advancing Earth Science Education. In *AGU Fall Meeting Abstracts, San Fransisco, California, USA - December 15 - 19, 2014*, volume 1, page 1, 2014.

[54] Jun Rekimoto. SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces. In *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems: Changing our World, Changing ourselves, Minneapolis, Minnesota, USA - April 20 - 25, 2002.*, pages 113–120, 2002.

[55] Stephan Richter, Christian Holz, and Patrick Baudisch. Bootstrapper: Recognizing Tabletop Users by Their Shoes. In *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 1249–1252, 2012.

[56] Markus Rittenbruch, Andrew Sorensen, Jared Donovan, Debra Polson, Michael Docherty, and Jeffrey I. Jones. The Cube: A Very Large-Scale Interactive Engagement Space. In *The ACM International Conference on Interactive Tabletops and Surfaces, ITS '13, St Andrews, United Kingdom - October 06 - 09, 2013*, pages 1–10, 2013.

[57] Volker Roth, Philipp Schmidt, and Benjamin Güldenring. The IR Ring: Authenticating Users' Touches on a Multi-Touch Display. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology, New York, NY, USA, October 3-6, 2010*, pages 259–262, 2010.

[58] Samsung. Interactive Full Room Wall, January 2014. Retrieved from: https://www.youtube.com/watch?v=8fJ2ORNuwKI. Last accessed: July 27, 2015.

[59] Dominik Schmidt, Ming Ki Chong, and Hans Gellersen. HandsDown: Hand-contour-based User Identification for Interactive Surfaces. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction 2010, Reykjavik, Iceland, October 16-20, 2010*, pages 432–441, 2010.

[60] Dominik Schmidt, Raf Ramakers, Esben Warming Pedersen, Johannes Jasper, Sven Köhler, Aileen Pohl, Hannes Rantzsch, Andreas Rau, Patrick Schmidt, Christoph Sterz, Yanina Yurchenko, and Patrick Baudisch. Kickables: Tangibles for Feet. In *CHI Conference on Human Factors in Computing Systems, CHI'14, Toronto, ON, Canada - April 26 - May 01, 2014*, pages 3143–3152, 2014.

[61] Christophe Scholliers, Lode Hoste, Beat Signer, and Wolfgang De Meuter. Midas: A Declarative Multi-Touch Interaction Framework. In *Proceedings of the 5th International Conference on Tangible and Embedded Interaction 2011, Funchal, Madeira, Portugal, January 22-26, 2011*, pages 49–56, 2011.

[62] Stacey D. Scott, M. Sheelagh T. Carpendale, and Kori M. Inkpen. Territoriality in Collaborative Tabletop Workspaces. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW 2004, Chicago, Illinois, USA, November 6-10, 2004*, pages 294–303, 2004.

[63] Teddy Seyed, Chris Burns, Mario Costa Sousa, Frank Maurer, and Anthony Tang. Eliciting Usable Gestures for Multi-Display Environments. In *Interactive Tabletops and Surfaces, ITS'12, Cambridge/Boston, MA, USA, November 11-14, 2012*, pages 41–50, 2012.

[64] Teddy Seyed, Mario Costa Sousa, Frank Maurer, and Anthony Tang. SkyHunter: A Multi-Surface Environment for Supporting Oil and Gas Exploration. In *The ACM International Conference on Interactive*

Tabletops and Surfaces, ITS '13, St Andrews, United Kingdom - October 06 - 09, 2013, pages 15–22, 2013.

[65] Beat Signer and Moira C. Norrie. PaperPoint: A Paper-Based Presentation and Interactive Paper Prototyping Tool. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction 2007, Baton Rouge, Louisiana, USA, February 15-17, 2007*, pages 57–64, 2007.

[66] SmartPixel.tv. Multitouch Application for Real Estate Agents Presented on an Interactive Table, October 2014. Retrieved from: https://www.youtube.com/watch?v=IonqPE7jPs8. Last accessed: July 27, 2015.

[67] Takram. On the Fly Paper, June 2015. Retrieved from: http://www.engadget.com/2015/06/05/intel-takram-paper-ui/. Last accessed: July 23, 2015.

[68] TMSolutions. Multi-Touch Interactive Restaurant Table, August 2013. Retrieved from: https://www.youtube.com/watch?v=gPHC9MkqHpE. Last accessed: July 27, 2015.

[69] Sandra Trullemans and Beat Signer. From User Needs to Opportunities in Personal Information Management: A Case Study on Organisational Strategies in Cross-Media Information Spaces. In *IEEE/ACM Joint Conference on Digital Libraries, JCDL 2014, London, United Kingdom, September 8-12, 2014*, pages 87–96, 2014.

[70] Brygg Ullmer and Hiroshi Ishii. The MetaDESK: Models and Prototypes for Tangible User Interfaces. In *ACM Symposium on User Interface Software and Technology*, pages 223–232, 1997.

[71] Xin Wang, Yaser Ghanam, and Frank Maurer. From Desktop to Tabletop: Migrating the User Interface of AgilePlanner. In *Engineering Interactive Systems, Second Conference on Human-Centered Software Engineering, HCSE 2008, and 7th International Workshop on Task Models and Diagrams, TAMODIA 2008, Pisa, Italy, September 25-26, 2008. Proceedings*, pages 263–270, 2008.

[72] Malte Weiss, Simon Voelker, Christine Sutter, and Jan O. Borchers. BendDesk: Dragging Across the Curve. In *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, Saarbrücken, Germany, November 7-10, 2010*, pages 1–10, 2010.

[73] Pierre Wellner. The DigitalDesk Calculator: Tangible Manipulation on a Desktop Display. In *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology, UIST 1991, Hilton Head, South Carolina, USA, November 11-13, 1991*, pages 27–33, 1991.

[74] Pierre D. Wellner. Interacting with Paper on the DigitalDesk. *Communications of the ACM*, 36(7):86–96, 1993.

[75] Andrew D. Wilson. PlayAnywhere: A Compact Interactive Tabletop Projection-Vision System. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology, Seattle, WA, USA, October 23-26, 2005*, pages 83–92, 2005.

[76] Takuto Yoshikawa, Buntarou Shizuki, and Jiro Tanaka. HandyWidgets: Local Widgets Pulled-out from Hands. In *Interactive Tabletops and Surfaces, ITS'12, Cambridge/Boston, MA, USA, November 11-14, 2012*, pages 197–200, 2012.

[77] Zoran Zivkovic. Improved Adaptive Gaussian Mixture Model for Background Subtraction. In *17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004.*, pages 28–31, 2004.