VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

A USER-FRIENDLY AND DEVICE-INDEPENDENT INTERFACE FOR CROSS-DEVICE INFORMATION EXCHANGE

ARNO DE WITTE

Academic year 2016-2017

Promoter: Prof. Dr. Beat Signer
Advisor: Reinout Roels
Science and Bio-Engineering Sciences

VRIJE
UNIVERSITEIT
BRUSSEL

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

EEN GEBRUIKSVRIENDELIJK EN APPARAAT ONAFHANKELIJKE INTERFACE VOOR INFORMATIE UITWISSELING TUSSEN VERSCHILLENDE APPARATEN

ARNO DE WITTE
Academiejaar 2016-2017

Promotor: Prof. Dr. Beat Signer
Advisor: Reinout Roels
Wetenschappen en Bio-ingenieurswetenschappen

Abstract

Users these days possess a lot of different devices. However, transferring information between these devices is not always an evident task. In order to solve this information exchange problem, the INFEX framework was introduced along with a tangible tabletop interface. On this interface devices could be placed, users can list files on a device, inspect the file in detail and transfer it to another device. However the current implementation of INFEX does not allow interaction with input-only and output-only devices. A more general problem is how to represent these devices, which typically are bound to a specific location, in a tangible tabletop interface. Another feature that the INFEX framework lacks is that users cannot connect their tabletop session to another remote instances session. This also introduces a general problem: how to show device ownership in a tabletop interface. Because many users do not have access to an interactive tabletop, the INFEX framework is also implemented as a classical desktop browser application.
Along with these issues, which are each a new feature that has to be added to INFEX, there are also some usability issues with the current INFEX framework. The framework in general is not user-friendly enough to be used in the proposed environments.
In this thesis we will introduce some related work to these problems, discuss how we designed solutions to these problems in a user-centered design process. The result of this was the introduction of graphical user interface which implements both fixes for the usability problems and implements the features for the specific problem statements. We will also discuss the implementation of these solutions and discuss the executed evaluations which include a user test that was executed to confirm proposed designs.

Acknowledgements

I would like to thank Reinout Roels for his guidance with this thesis and his insightful comments. Also, I would like to thank Prof. Dr. Beat Signer for the opportunity to do this thesis and the proofreading.
I would also to thank my friends and the assistants at the WISE lab for the informal evaluation moments. Another special thanks goes to my girlfriend for the help with both proofreading and support during the writing of this thesis.
Lastly I would like to thank all the participants of the user test.

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.

Contents

1 Introduction

1.1 Information Exchange

Users nowadays own a lot of different devices containing different types of information. In this day and age almost everybody owns a smartphone and almost all families own a laptop, desktop computer, tablet and a camera. Another ongoing trend is the move towards the cloud. More and more (free) cloud services allow for users to store their information in the cloud.
However transferring information from one device to another, from device to service or from service to service still requires a lot of intermediate steps. For example, if a user wants to transfer a picture from their smartphone to their tablet they will have to either do an intermediate step and transfer it using a laptop or through a service. Often users resort to using email to send their information from one device to another. This solution is far for ideal and, for example imposes limits on the file size.

1.2 INFEX Framework

To solve this information exchange problem, the currently unpublished INFEX (INFormation EXchange) framework was created at the WISE lab[1].

[1]This paper is currently under review and thus a reference is impossible.

The INFEX framework is a general solution for this problem, it allows for information exchange between different devices and online services using different protocols.
The user interface introduced with the INFEX framework is a tangible tabletop interface. Users can put their devices or tangibles on the tabletop, where simple interface elements allow the users to list the files on a device, inspect files on the table and to transfer files between devices. Devices are recognised by the framework using discovery plugins and can connect to the framework through communication plugins. This makes the framework extensible as any developer can create a discovery or communication plugin that supports communication to new devices.

Figure 1.1: The original interface of the INFEX framework

However, the INFEX framework is not restricted to local devices placed on the tabletop, and allows for external sources to be connected. For example it is possible to connect an FTP server, Dropbox or other cloud services. Because there is no physical device, these remote sources are represented by physical cubes on the tangible tabletop interface. Thus all information sources and targets are represented by tangibles, either the physical device or by such a cube.

1.3 Problem Statement

The INFEX framework however lacks some functionality. INFEX as introduced is not user-friendly, for example to add a device, users have to fill out a configuration file. It also lacks features that might prevent its use as a

information exchange system. This user-friendliness problem can be divided into two subcategories: current usability issues with the INFEX framework and features missing from the INFEX framework or its user interface. For the latter we can define three general problem statements that each define a missing feature in INFEX. These do not only apply to INFEX specifically. One of the goals of this thesis aims to gain insight and try to solve these problem statements. Solving these problems will contribute to improving the user-friendliness of INFEX.

PS1: *How to transform an INFEX tangible tabletop-based interface into a browser-based desktop application?*
An average user does not have access to an interactive tabletop. However, these users still face the issue of information exchange. Therefore it is proposed to create a browser application. This has no impact on the core functionality of the framework: the communication to the each device is handled by a desktop computer. However using a more limiting interaction medium might impact the user experience and usability. To reduce this impact, a user-centred development approach is used and an evaluation (on all parts of the user interface) is performed.

PS2: *How to represent device ownership in a distributed tangible tabletop interfaces?*
A feature that the original INFEX framework lacks, is the ability to connect to another instance of the framework, for example if we want to share our holiday pictures with relatives. When connecting to a different tabletop interface it is however hard to know which devices in the interface belongs to whom. If we are connected to three other relatives, it is evident that the camera (which is tangible for us) on the tabletop is ours, but it is harder to identify the owner of represented the virtual devices.

PS3: *How to implement and represent input-only and output-only devices (which might not physically fit on the tabletop) on a tabletop interface?*
The original INFEX framework does not support input-only and output-only devices. We will call these devices source and endpoint devices respectively, these devices are devices that only support a single output or input action. For example a printer only supports the print output action, webcams only allow to import images or video streams. To support these two extensions will have to made: support for streams within the framework and a way to represent these devices on a tabletop interface. These cannot be put on the tabletop as they are either

too big (e.g. a printer or a television) or their location is a part of their functionality (e.g. a baby monitor webcam).

1.4 Contributions

To solve the problems listed in the previous section, we introduce a new graphical user interface for the INFEX framework. The interface is run on the tabletop to solve problem statements two and three, and on a laptop for problem statement one. To support these solutions, modifications to the existing INFEX framework had to be made. In order to design this user interfaces, a study was performed of different existing guidelines. An iterative user-centered design methodology was used, in which a user-test was executed which focussed on the two main aspects of the user-friendliness problem of INFEX: the problem statements and the usability issues with INFEX. The result of this design process is a prototype interface that implements solutions to the problem statements and fixes the usability problems that the INFEX framework faced.

1.5 Thesis Structure

In this chapter the INFEX framework that aims to solve some the shortcomings associated with information exchange was briefly introduced along with the problem statements for this thesis. Chapter 2 discusses similar systems that try to solve the information exchange problem and systems that touch on similar problems in tabletop and tangible interfaces. The next chapter discusses some background on design, implementation and methodology decisions. The design of the modifications to the core model of INFEX and the proposed solutions are discussed in Chapter 4. Chapter 5 discusses the implementation details both on the GUI module (frontend) and the INFEX framework (backend). The next chapter is about the evaluation and user study which influenced some of the design and interface decisions. Finally a conclusion of the work, including some limitations and future work is discussed in Chapter 7.

2 Related Work

2.1 Information Exchange Systems

Many users face the information exchange problem when trying to transfer files from one device to another. When trying to transfer files, many unconventional methods are used. Some users try to use email to transfer files. This method sets a number of limitations, first the file size is limited and secondly the receiving device must have access to an email client. Many media devices (e.g. cameras) do not meet these requirements. However many of these devices have wireless connection options. For example a GoPro does not have an email client but does have WiFi connectivity[1]. It is clear that there exists a connectivity mismatch between such devices, which interferes seamless information exchange. This problem has been identified by other researchers as well [40, 54].

Weiser introduced the idea of ubiquitous computing (later shortened to UbiComp) [71]. His vision, published in 1991, stated that with the advancements in both hardware and software, computing will become such a part of our living environment that we would stop noticing it. He compared this with the ability to write spoken language down, something we find so evident that it is difficult to imagine life without it. In later work [72], Weiser identified

[1] `https://github.com/KonradIT/goprowifihack`

some problems with reaching the ubiquitous computing vision. The information exchange problem we introduced, is an example of a problem that has to be overcome to come closer to the ubiquitous computing vision [44].

The related work for information exchange can be divided into two main problems: discovering other devices and end-user interactions.

2.1.1 Device Discovery

Before information exchange can take place, a device has to be discovered. Discovery in wired connection is trivial. However, in wireless connections the task becomes more complicated. Some wireless protocols such as Bluetooth provide functionality to discover other devices[2]. But Bluetooth has also some drawbacks such as bandwidth and latency. In some other technologies such as IP there is no built-in solution for discovering other devices. For the IP example the only way to discover other devices within a network is to scan every possible IP address.
For some applications simply discovering devices is not enough. In tabletop applications (such as INFEX) the exact spatial position relative to the table of a device has to be known as well. This has to be correct to be able to render the user interface elements which enable interaction. Discovering devices has been an issue in spontaneous interaction systems which is a part of the ubiquitous computing concept [44, 19].
A number of techniques are used to solve these discovery problems. The first technique is the most common. This technique is the use of optical sensors. HuddleLamp [54] is a system that allows for cross-device interaction through connecting the multiple devices to a server. Each device runs a mobile browser and the server provides a JavaScript API. Devices are discovered, as illustrated in Figure 2.1, using a RGB-D camera which can identify screens. HuddleLamp thus only provides spatial discovery, the connection to a server has to be done manually.

Another optical technique to discover spatial location is Dynamic Tiling Display introduced by Li et al. [38]. This technique uses the front-facing cameras of mobile devices to spatially locate themselves. There is a marker above the devices, this allows through visual pattern recognition for devices to get their relative position, distance and orientation. It is implemented as a host-client architecture, where one device acts as a host and other devices act as clients, receiving data from the host. Communication is done through UDP and TCP. It uses UDP broadcast to identify other clients. This system

[2] `https://www.bluetooth.com/specifications/assigned-numbers/service-discovery`

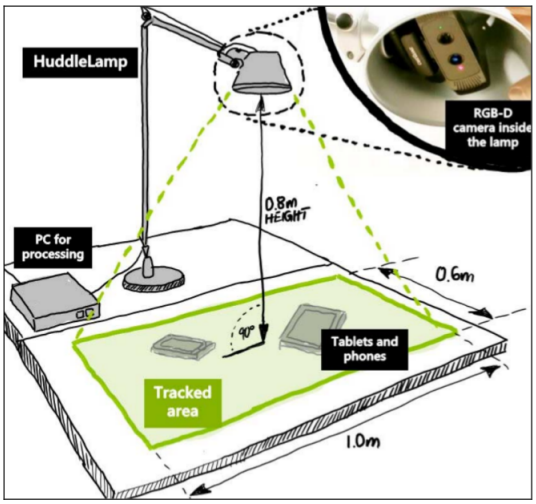


Figure 2.1: The HuddleLamp setup

is thus limited to a single protocol (UDP) where devices should be on the same network.

A common technique for optical fiducial tracking (and thus optical spatial discovery) is called reacTIVision [33]. ReacTIVision is used as the visual engine for the reactables tangible tabletop interface which is discussed in Section 2.2.2. In reacTIVision there is an overhead camera which tracks visual patterns which are placed on top of objects. ReacTIVision uses amoeba patterns which allow subpixel accuracy. This framework also allows for the tracking of fingers which makes it ideal for use in tabletop interfaces.
BullsEye [35] further explores the core idea of reacTIVision, but improves upon it. This allows for higher efficiency and accuracy (down to a tenth of a pixel).

Figure 2.2: The fiducial of BullsEye (left) and reacTIVision (right)

Previous solutions for device discovery were focused on discovering devices in a single setup (e.g. in a lab or on a desktop). The RELATE framework was introduced to allow for peer-to-peer and thus infrastructure-independent device discovery and positioning [19]. Older techniques often relied on beacon

technology or other sensors to identify whether a device is in a certain general position (e.g. inside of a shop). The framework is focused on fulfilling the spontaneous interaction requirement of ubiquitous computing. The goal of the framework is to identify the relative location of devices, such as smart surfaces, close to a single user. This can be done through either optical (as discussed earlier in this section) or through ultrasonic positioning [22]. The ultrasonic positioning is done through a special USB dongle which can sense the ultrasonic signals of other devices and emits an ultrasonic signal. This dongle has to be inserted into each device. The information from the USB dongles is passed on to the RELATE framework which can than calculate different devices relative positions.
Another non-stationary device discovery system is called GroupTogether [41]. Rather than tracking individual devices, GroupTogether tracks persons and their orientation towards each other. For example when people are close to each other and face each other, some interactions are possible with the devices they are holding. For example people standing next to each other can trigger a file transfer by tilting their tablets. The system also allows for big screens such as interactive whiteboards and televisions. The tracking is done through an overhead Kinect. This approach allows for natural standing interactions.

Apart from the spatial discovery, devices must also be able to connect to each other. Therefore some discovery frameworks are also created in research. A first system is called Connichiwa. Connichiwa is a framework that allows for cross-device web applications [60]. It provides a JavaScript API similar to that of HuddleLamp. A local web application is run on each device such that through the API, they can communicate to other devices. The cross-device communication is done over IP. To discover other devices IP addresses, Bluetooth Low Energy is used.
A similar but more elaborate system is Bluetable [76]. Bluetable allows for cross-device communication on a tabletop interface. Devices are discovered by a host system (which directs all communication) through a visual sensor, then a Bluetooth connection is attempted. When this Bluetooth connection is successful, an IR signal will be sent from the device. Communication between host and device is done over the Bluetooth connection. Bluetable imposes quite some hardware requirements on each device.

Discovery can also be done through Radio-Frequency Identification (RFID), electromagnetic sensing and Near Field Communication (NFC). These technologies are limited to a close range. Devices should thus be placed close to the sensors. An example of such a system is Sensetable [53]. Sensetable allows for objects to be recognised when put a specific tabletop (3 different

implementations are described). These tabletops can sense differences in the electromagnetic spectrum as illustrated in Figure 2.3. The system provides low latency and high precision at close range. However it requires special hardware in the objects placed on the tabletops. The tabletops should also be equipped with spatial sensors, however this constraint is comparable with other interactive tabletops.

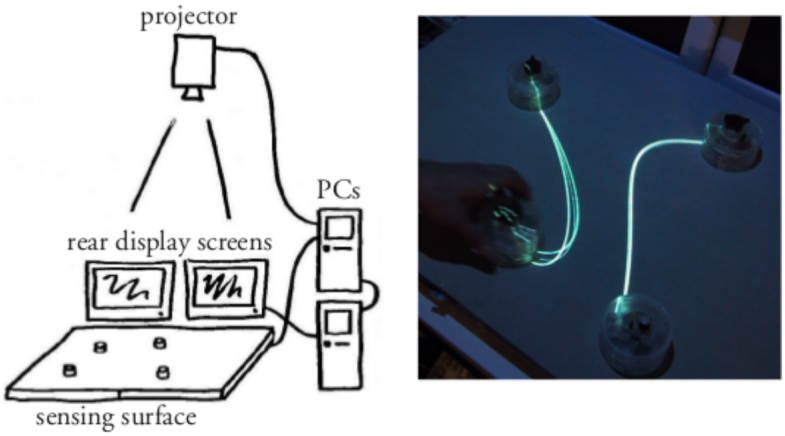


Figure 2.3: On the left Sensetable setup and on the right a demo application of Sensetable

Another possibility for device discovery is using fixed QR codes. This method has been applied by Frosni et al. [18] in their framework. The framework allows for easy development of applications with a distributed user interface, for example having a museum walkthrough with multiple devices and a guide. A QR code is used to get the backend engines connection information. User devices will connect to the engine themselves and initialise. In this case device discovery involves a manual step, scanning of the QR code with the camera of the device, by the user.

2.1.2 End-User Interaction

In this section systems that already implement cross-device information exchange in some way are discussed. Most of the systems require devices to be close to each other. These types of proxemic interactions were identified by Marquardt et al. [40] in the context of ubiquitous computing.

Pick-and-Drop [56] allows users to use digital pens to transfer objects from one device to another. The paper imagined that in a world of ubiquitous computing it should not matter on which device an object is stored, it should be easily transferred from one device to another. The paper describes

(a) Embodied Data Objects (b) DroPicks

Pick-and-Drop as a natural extension of the drag-and-drop action which was and still is common in WIMP (Windows, Icons, Menu, Pointer) interfaces. Feedback to the user is given through a shadow. Objects can also be taken from physical resources such as booklets, a camera than tracks the position of the pen and passes it to a database. Transfers are done through TCP/IP and the application is written in Java. Objects can only be taken from the application and dropped within another instance of the application. This system was further expanded using a projector which projected extra information about objects placed on a tabletop [55].

Embodied data objects allow users to make digital information physical by linking a certain physical object, which is equipped with an RFID tag, to a document [69]. When a physical embodied data object is read by another computer, it can retrieve the digital information through a SOAP service.
A very similar system called DroPicks allows users to physically represent digital pieces of information [23]. DroPicks is more focused on representing Internet content instead of files, which is the focus of embodied data objects. A user can leave a DroPick at a certain location where other users can take it, read it and view the content as shown in Figure 2.4b. DroPicks also uses RFID for the physical objects, but uses Bluetooth to transfer the actual information.
Another very similar system is called Touch & Share [59]. In Touch & Share users can use their NFC equipped mobile device to see digital information that is linked to a RFID tag. Information is stored on a server and is transferred over HTTP.

Dippon et al. described a seamless integration for mobile devices and a interactive surface. In their demo application they had an android appli-

cation that allowed users to share pictures. Users had to put their device on the interactive surface (which has a RGB-D camera above it), then connect to the interactive surface by showing a unique pattern on their screen. Once connected they could share pictures to other users connected to the interactive surface.

Gestures are used in many different user interfaces. For information exchange Dachselt et al. [14] introduced a technique where information can be "thrown" towards a large display. This technique uses the built-in accelerometers of smart devices to detect the gestures. Users can view a picture and then make a throw motion with their smart device in their hand as shown in Figure 2.5. The smart device will then contact a server with sensor data using UDP and send, when the gesture is detected, the file over TCP.

Figure 2.5: Throwing motion makes the image appear on the screen.

Several walk-up interactive surfaces have been developed. These systems allow their users to meaningfully share content and enhance user-to-user interaction. For example WeSpace allows users to share screenshots and other data whilst in a meeting [75]. There are also more stationary systems that provide similar functionality such as Dynamo [31] and IMPROMPTU [8]. Both of these application support a central big screen where resources can be shared, but have to be shared through the application with a fixed protocol.

A different approach is ActivitySpace [25]. ActivitySpace allows users to keep information that is linked to a single activity such as a research project. On each device you can see the files related to that activity. Files can be managed in a useful fashion (e.g. pin important files). It is also possible to send files to other devices that a user owns. Each device is either isolated, master or slave. Isolated devices do not allow cross-device file transfer for activities, masters own all files and slaves can be used to send files for a certain activity. A REST interface is used for communication between master, interactive surface and slaves. This is combined with a broadcast service to indicate new activities.

Another interactive surface-based system is WatchConnect [24], which focuses on different interactions involving smartwatches. WatchConnect is a

hardware and user interface toolkit that allows developers and researchers to prototype interactions such as file transfer between smartwatches, interactive surfaces and other devices. An example application allows users to move data from an interactive display to their watch and then transfer it to another device, this way the smartwatch becomes a *wearable mediating storage device*. Communication in the toolkit is done through a REST interfaces to which all devices should be connected. Typically ran on the same computer as the interactive surface.

2.2 Tabletop Interfaces

Many tabletop interfaces are developed in research. Interactive tabletops have several benefits, they allow for multiple users and help improve collaboration, motivate users and improve performance [10]. Some of them discussed similar problems as stated in the problem statement. In this section we divide three different tabletop interfaces: ordinary tabletop interface, tangible tabletop interfaces and connected tabletops.

2.2.1 Ordinary Tabletop Interfaces

A wide range of tabletop interfaces with various goals have been created in the past. In this section we will look over some tabletop interfaces that show a relevant connection to the INFEX Framework and the problem statements.

A commercial tabletop interface technology is Microsoft Pixelsense[3]. The Pixelsense is currently no longer available as a flat interactive tabletop but has been re-branded under the Surface name. Pixelsense is a touch table that provides object recognition. This recognition can be used for detecting devices but it is also possible to buy accessories that can be placed on top of the touch interface. These accessories can provide a tangible way to interact with applications on the table.

A classical application of interactive tabletop interfaces is a picture gallery. SharePic [6] is a picture browser created for elderly. SharePic is a multi-touch and collocated multi-user system. It uses a DiamondTouch [15] tabletop which can detect touches on a table through a specific electrical field which is generated by transmitter. Each user has a special receiver wired to their seat which can detect the changes in the electric field. In DiamondTouch the image is projected using an overhead projector. Other objects (e.g. coats) do not cause a touch event to be triggered. In SharePic users can view, save and share pictures. Each user has their personal space where they can store

[3] `http://pixelsense.com`

certain pictures. The architecture focusses on some distinct actions such as *select*, *move* and *copy*. The system however is focussed on the user interaction with the elderly. It does not provide a (device) source of the pictures. Pictures can only be shared once in the interface.

Focus is a tabletop file explorer [13]. Classical file explorers are meant to be used with a mouse and keyboard by a single user. Focus tries to bring file access to tabletop interfaces. It is independent of the used tabletop technology. In Focus each user can define similarities themselves, for example how similar the author is to a document. When a document is focussed on the tabletop, all similar files from all the users at the table are available as seen in Figure 2.6. Just as SharePic, Focus also has the black hole concept to discard certain files. This system focusses on collaboration, context of use, orientation, table use and removing clutter. However it lacks a plugin structure so there is limited file type support. Files are transferred from user computers to the system using HTTPS and only Mac OS X computers are supported.

Figure 2.6: The Focus tabletop interface

Morris et al. [43] have previously discussed the social dynamics of multi-user tabletop interfaces. While designing a tabletop interface, it should be noted that group dynamics can play a role in the interaction. For example when multiple users try to interact with the same interface elements or tangibles on the interface. They propose some example solutions: tear where the interface element breaks up into pieces, duplicate where duplicate interface elements are created and rank users and allow the highest ranked user to execute the action.

Another aspect of social behaviour on tabletop interfaces is territoriality [61, 68]. When different users are at a table handling certain digital or

physical objects, they may use certain parts of the table for group work while using other parts of the table for personal work. Scott et al. [61] described three types of territories in ordinary tabletop interfaces: personal territories, group territories and storage territories. In group territories group actions are executed, such as collaborative work. Storage territories hold all currently unused resources and the personal territories are for personal work, for example writing something down. In their work they also described some design implications which are covered in Chapter 3.

2.2.2 Tangible Tabletop Interfaces

Tangible interfaces are interfaces that are enhanced physical artefacts. These artefacts provide either input or output modalities. On top of the digital feedback loop, which is already present in most user interfaces, tangible interfaces add a physical feedback loop [62]. In tangible tabletop interfaces the artefacts are typically placed on top of or next to the horizontal surface of the tabletop.
The idea and vision for tangible interfaces was first introduced in the paper *Tangible Bits* where Ishii et al. [30] introduced the concept of using the physical world as a user interface. There is also research that tries to investigate the difference between a pure touch-based tabletop interfaces [67, 4], which concluded that tangible tabletop interfaces impose different types of usage (e.g. the use of one or two hands and implications for sequential actions). Hence the usage of tangibles in tabletop interfaces should improve the user interface in a meaningful way. However good use of tangibles in these touch tabletop environments can benefit the user performance and user experience.

One of the best known tangible tabletop interface is the reacTable introduced by Jordà et al. in 2007 [32]. ReacTable is a tangible tabletop interface that allows users to create music using tangibles. ReacTable uses the visual reacTIVision engine mentioned in Section 2.1.1 to recognise tangibles. ReacTable highlights the use of computer vision for tangible discovery and tracking such as orientation, more points of interest detection and finger recognition. Music is created by different types of objects, each with a distinct shape. Some objects generate sounds, others can have various effects. These objects can be placed on the round tabletop and connected to create sound. Artists have used this setup during live music performances[4].

[4] `https://www.youtube.com/watch?v=xdcEJFHId1I`

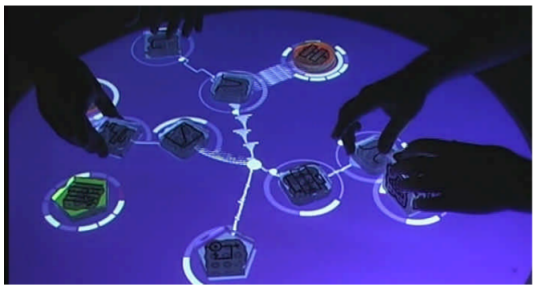


Figure 2.7: Two people using a reacTable

2.2.3 Connected Tabletops

Most of the tabletop systems discussed in previous sections were either single user or co-located multi-user situations. Which means that one or more people can use the interface on the same physical interactive tabletop. However some tabletop systems are connected through the Internet. These types of user interfaces, where not every user is in the same location, are called mixed presence groupware [66]. Both for regular user interfaces as for tabletop interfaces this introduces some design challenges [68, 57], how to cope with these challenges in tabletop interfaces will be discussed in Chapter 3. In this section we will talk about some of the connected tabletop systems that have been created in research.

Wang et al. [70] introduced AgilePlanner which is a tabletop interface for agile development teams. Because many development teams are spread across the globe, tabletops in multiple locations can be connected to each other. This is done through a central server which handles all events and notifies every other client.

One of the problems in mixed presence groupware is the visualisation of remote users actions. Digitable [12] tries to solve this by sending visual information to the remote client. In Digitable users have a vertical screen where they can see their remote collaborators on and a horizontal interactive tabletop for various applications as seen in Figure 2.8. Through computer vision, Digitable can send visual information to remote clients. This can be used to track arm movements or tangibles which are placed on the interactive surface.

An earlier system that supports distributed tabletop interaction is the DoubleDigitalDesk [73]. The DoubleDigitalDesk allows users to superimpose documents that are on another desk, documents on a desk are also superimposed on the other desk. This is done with a projector and camera. Multiple desks are connected over an ethernet connection. As seen in Figure 2.9 users

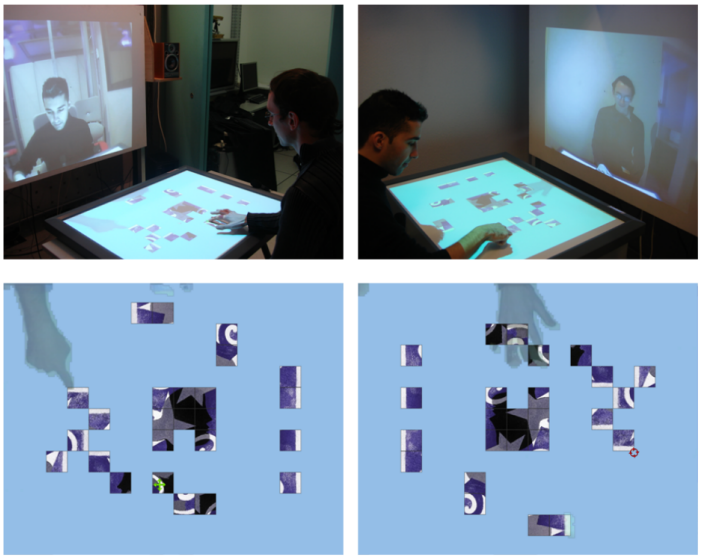


Figure 2.8: Digitable setup and use case

can also select text and this selection is mirrored on the other desk, selections can be moved around and pasted somewhere.

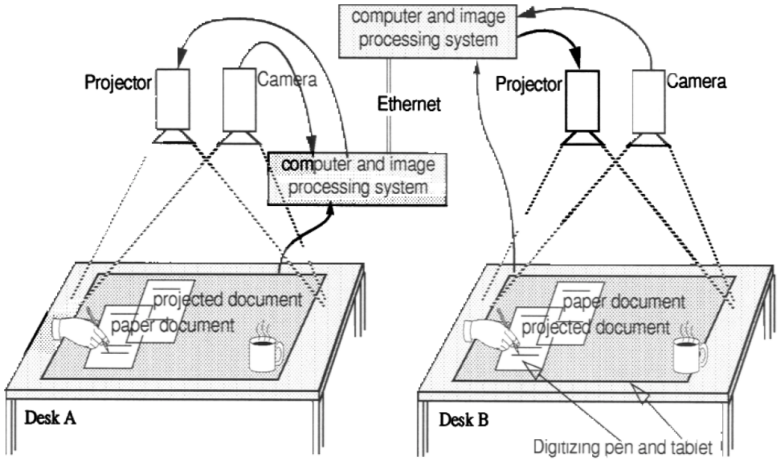


Figure 2.9: The DoubleDigitalDesk

2.3 Discussion

Many of the introduced device discovery systems provide adequate means of discovering different kinds of devices. However many of the introduced

systems (e.g. BlueTable) depend on specific hardware inside of or attached to the devices to be able to function. Other systems, such as HuddleLamp and GroupTogether, can only operate in a limited space. These requirements limit the possibilities for systems that want to use device discovery. This is something that the INFEX framework solves by using modularity, which can be used to combine the introduced methods. It also removes communication restriction for devices, as multiple discovery methods can be combined, which is a step towards ubiquitous computing, as users need not to be aware of the capabilities of a certain device or system.

The introduced interaction mechanism for information exchange faces two problems. The first being that their main goal is not information exchange. These systems are all based on interaction between different devices. They introduce many techniques of achieving cross-device information exchange, which can be useful for INFEX as well, as seen in the RELATE model. However, the second problem is that these systems only work using one or a specific combination of communication protocols. On top of this, many systems require specific software to be installed. Some systems try to support a variety of different devices by using a browser or programming the application in Java. However, this is not enough, as many devices only support information exchange over a single, often proprietary, communication protocol. Many cameras, such as GoPro, face this issue. These cameras do not support a web browser or Java applications. As will be explained in the next chapter, INFEX is flexible and supports these devices as well.

Many different tabletop applications have been created and much research has gone into this topic, which has resulted in many guidelines that can be applied in the design of the prototype created for this thesis. However some problems have not been explored. Many remote connected tabletops have also been created, there are even guidelines created to help researchers and developers [57]. These do describe users having different interaction elements on the table, either virtual or in the form of tangibles, however these are almost all used in a cooperative way. None of the actors in the interface own any of these interaction elements, but they inherently belong to the interface. When having information exchange with multiple users and devices, this becomes a problem, as each piece of information is owned by a certain person. This relation between information and its owner needs to be able to be represented and protected within tabletop interfaces.
Tangibles are usually used to materialise a concept. In reacTable, tangibles materialise the different audio sources and filters. These systems introduce a lot of concepts on how to track and interact with tangibles. However there is no tangible system at the time of writing where tangibles represent physical

devices. These devices might be in the line of sight of the user, but their functioning is bound to a particular location (e.g. security camera). How this can be done best and whether this is a helpful way to represent these devices, remains the question.

3 Background

3.1 INFEX Framework

In the previous chapter we introduced many systems, some that provided solutions for the information exchange problem. We also pointed to some issues that were not addressed by these systems. In the unpublished INFEX paper, it is pointed out that many of the systems require that special software has to be installed on the devices to allow for information exchange. Others might even extend these requirements and require devices to have a screen. Even when no special software on the devices is needed, some setup is required. For example when a device wants to be connected, the setup requires users to enter the IP address of a central server. Another issue that many systems face is that they are not extensible. Few systems provide a plugin interface that allows for extensibility, thus limiting them to a single protocol or user interface.

To overcome these problems the INFEX framework was introduced. The framework was developed according to eight requirements:

List, Inspect and Transfer Content
The INFEX framework provides 3 core user actions. Users can *list* files that are on a device, *inspect* files, for example view a video, and *transfer* files between different devices.

Customisable Yet Consistent User Experience
As will be introduced in Section 3.2, consistency is important for the usability and user experience of a user interface. The framework is however customisable such that different interfaces or interaction techniques can be used on top of it.

Reduced Hardware Constraints
Because the INFEX framework wants to support many different devices, the constraints for these devices should thus be very light.

Support for Remote and Non-mobile Devices
Many users store their information not only on their devices, but also use cloud services such as Dropbox and Google, but also using simpler ways such as FTP or git.

High-level Abstractions
The three core actions are an example of a high-level abstraction, developers of applications do not need to worry about technical details of the framework.

Broad Connectivity and Access Support
INFEX aims to provide connectivity support for as many as possible communication protocols. These protocols might even be based on other supported protocols. For example RESTful HTTP relies on TCP/IP sockets. The implementation of these protocols should thus be reusable.

Bi-directional Mediation
Different devices or services might use a different protocol. For example a phone communicates over Bluetooth and Dropbox uses HTTP. Then INFEX will be a mediator and allow for bi-directional communication between the phone and Dropbox.

Extensibility and Reusability
The extensibility of INFEX allows for new protocols to be easily added to the framework. As mentioned in the connectivity requirement, the implementations of different protocols should be reusable. The INFEX framework should also allow any type of content to be transferred.

According to these requirements, three goals have been defined for the framework. The first being that the framework should allow for the easy development of applications for user-driven information exchange. Secondly

INFEX aims to support information exchange between devices or combinations of devices that are currently not supported. For example devices using a currently unavailable protocol. The last goal is to provide developers with a framework that allows for the development of different interfaces.

The original INFEX architecture is depicted in Figure 3.1, a large version of this image is available in Appendix D. There are three pluggable components (indicated by the plugin icon) in this architecture: *Device Detection*, *Communication* and *GUI Module*. The first component is responsible for detecting devices that want to participate in information exchange. Different example methods of device detection were introduced in Section 2.1.1. Each discovery plugin can detect whether or not a device is joining an information exchange session. Additionally detection plugins can detect physical location for example on a table, which can be used by user interfaces to draw certain interface elements. The discovery plugins are run continuously, thus when a devices leaves the framework can be notified and poll for new devices.

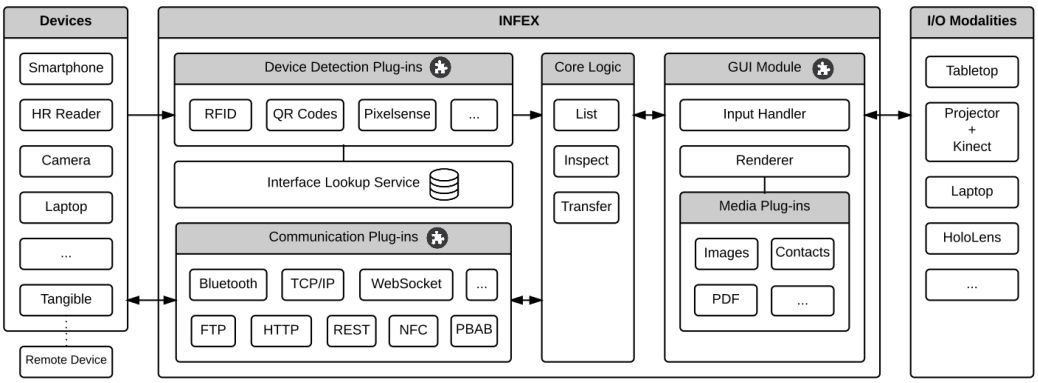


Figure 3.1: INFEX architecture

For devices to be able to participate in information exchange using the INFEX framework, a one-time setup is has to be performed. The setup is required such that communication addresses such as Bluetooth MAC or IP addresses can be stored in the framework. The setup can also be done through RFID as is done in [59]. It is also required that the frameworks knows what types of content a device can exchange. On detection the stored information is used to identify if a device is already known and thus is ready for use, or whether it needs to go through the setup. Once the framework recognises a device, it can proceed to use one of the three core actions: *list*, *inspect* or *transfer*. To do this communication plugins have to implement three methods: *write*, *read* and *list*. These output data to the framework. The data is formatted in a generic data format. If this is not the case by

default (e.g. Google returns contacts in a specific XML syntax), plugins are responsible to convert the data to for example JSON.
The third pluggable interface of INFEX are the Graphical User Interface (GUI) modules. INFEX allows for different types of graphical interfaces to be used. Different I/O Modalities can be used to project and enhance GUI modules. For example a depth camera (as used in Huddlelamp [54]) can be used to locate devices in a projected interface. To allow for reusability in GUI modules, media plugins can be created. Which can be done in the Web domain with HTML components can be used. Using these components it is possible to define a media plugin for inspecting PDF files. This can be reused in other web-based GUI modules. However if developers want to create a GUI using QML or other technique, these media plugins can also be created for these frameworks. For this thesis we will create such a GUI module, which is a tangible tabletop interface. The INFEX framework already comes with such an interface, however this interface is implemented as a proof-of-concept and lacks some of the functionality which are defined in Section 4.2.1.

To illustrate how the INFEX framework might work for a user, we will use the example shown in Figure 3.2. In this example we see a dumbphone and a smartphone. The dumbphone might connect over Bluetooth PBAP (Phone Book Access Protocol) and the smartphone might connect through a WiFi connection. This is however irrelevant to the user, INFEX acts as a protocol mediator which abstracts these protocol differences from the end-user. The three core actions are depicted by the numbers 1 (List), 2 (Inspect) and 3 (Transfer). A GUI might choose to differentiate different content-types as is seen in the example as different tabs for contacts, music and images are available.

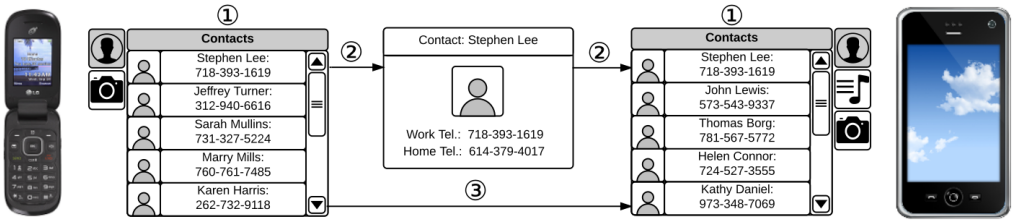


Figure 3.2: An example of a possible INFEX GUI could look like

The INFEX framework is implemented in Java. Connection and detection plugins are implemented as Java JAR files. These can be put into a certain directory, and will then be detected. A plugin can provide certain classes that implement the *CommunicationPlugin* or *DetectionPlugin* interfaces. As stated earlier, INFEX has a proof-of-concept GUI which is implemented as

a web application using HTML5 and JavaScript. To connect to the INFEX core actions, a WebSocket was used. The proof-of-concept GUI is shown in Figure 1.1

3.2 Guidelines and Heuristics

Past research has introduced many guidelines and heuristics for designing and developing user interfaces. In this section the relevant guidelines and heuristics for the usability extensions required to solve the problem statements, will be discussed. However using guidelines and heuristics (even for evaluation) will not guarantee a good user interface [48]. To ensure that the user interface has no usability problems, many different inspection methods can be used [47]. However, none will guarantee that there are no usability problems with the user interface. These will also not come up with a solutions to identified problems.

For various types of user interfaces design guidelines and heuristics have been defined [29, 28, 5, 17]. One of the most used heuristics for regular user interfaces, are defined by Nielsen et al. [42]. These are aimed to enhance human-computer interaction and describe various heuristics. For this however thesis the following are most relevant:

- Simple and natural dialogue
- Be consistent
- Provide feedback
- Error prevention

Nielsen compared different sets of heuristics using known usability problems in [46]. He concluded that consistency is the top heuristic to explain all usability problems that are present in a certain user interface. However in his work Nielsen concluded that identifying these problems is not guaranteed using these heuristics. In earlier work Nielsen also introduced some methods to create consistency in user interfaces [45]. *Style-based consistency* and *Shared code* are some methods that can easily be implemented by the user interface for this thesis as discussed in Section 5.2. *Hardware support for consistency* is a method that is not feasible in tabletop interfaces, as many different types of interactive tabletop hardware exists. However there is a software solution for this. It is also not very important for our prototype.

A common technique that is used in user interfaces is the use of animations. Animations allow for interface elements to move or be highlighted in

such a way that the user is aware of the changes in the interface. An example animation is motion blur, the benefits of which are illustrated in Figure 3.3. The use of animations can have a positive effect on the user experience and decision making [21, 64, 11]. However some guidelines are provided, for example animations should be smooth, affect a whole graphical representation and be consistent. For this, frameworks such as [26] have been developed that allow for easy animation development. Some research has already been done

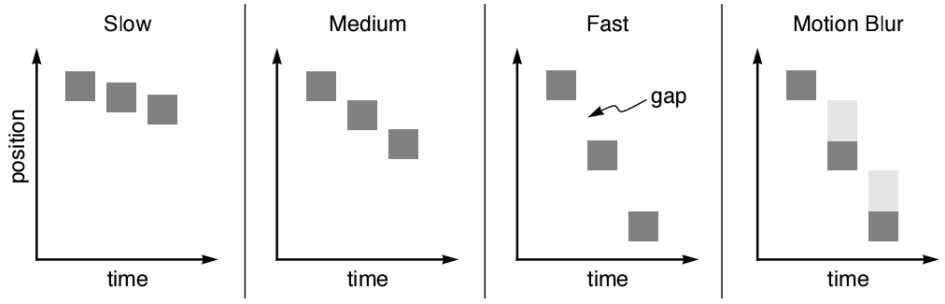


Figure 3.3: An example of how motion blur highlights fast moving objects

to establish guidelines for tabletop interfaces. Apted et al. [5] introduced some heuristics for tabletop evaluation. Some important heuristics that are discussed in the paper are:

- Design independently of table
- Support reorientation
- Minimise human reach
- Use large selection points
- Support private and group interaction

The *support reorientation* heuristic [37] has been researched thoroughly which resulted in the introduction of the Rotate´N Translate interaction mechanism. Rotate´N Translate allows users to manipulate elements on a tabletop interface is a natural way by rotating and moving UI elements in way that is similar to how physical objects would act. It has been implemented by for example AgilePlanner [70]. The importance of orientation (and the ability to rotate elements on a tabletop interface) is pointed out by Kruger et al. [36]. Because users can use an interactive tabletop from any side of the table, the interface elements should be oriented towards the user in order to use them.

Kruger et al. also provided some design guidelines to prevent orientation issues in tabletop interfaces.

An important aspect in multi-user tabletop interfaces is territoriality. As mentioned in Section 2.2.1, we will introduce some design guidelines for this problem. Tuddenham et al. [68] introduced two implications. The first is to add visual boundaries between different territories in order to aid coordination. Their second implication is that changes in the interface should be correctly localised, meaning that it users should be aware of which user (or which hand) caused a certain interaction. These visual changes should be continuous and incremental. An example of this last implication is Rotate´N Translate. They also noticed different coupling styles between different users such as *View engaged* where one user watches another user executing a task. In [61] some similar design guidelines are introduced:

- Provide functionality in the appropriate locality
- Provide visibility and transparency of action
- Provide appropriate table space
- Allow casual grouping of items and tools in the workspace

The first point is very similar to the implications found by Tuddenham et al. [68]. These design guidelines are introduced along with the different territorial spaces, which are briefly discussed in Section 2.2.1. The last point is aimed to allow users to stack different tabletop tangibles in their storage space.

When designing a system for human-computer interaction, a designer should be aware of the infrastructure problem [16]. This problem has three facets, the first being *constrained possibilities*, which is a problem that occurs when certain design or infrastructure decisions make certain desirable user-computer interactions impossible. An example in tabletop interfaces is the zoom problem. The zoom problem occurs when trying to project a large screen with high resolution, on a small screen. Either text will look very small or the screen can only project a small part of the other screen. The zoom problem was encounter in AgilePlanner [70] where they projected only a small part of the bigger screen.
The other facets of the infrastructure problem, are *interjected abstractions*, where abstractions become part of the interface (e.g. applications relying on public key encryption), and *unmediated interaction* where users have to tamper with the infrastructure without using the user interface (e.g. network hardware failures).
The INFEX framework attempts to cope with these infrastructure problems.

For example the *reduced hardware constraints* and *extensibility and reusability* requirements are introduced to prevent the *constrained possibilities* problem.

4 Design

4.1 INFEX Architecture

The INFEX architecture as introduced in Section 3.1 lacks some components to allow the implementation of solutions for the introduced problem statements. In the next section some of these shortcomings will be identified. Next, we will introduce some modifications to the INFEX architecture are necessary to implement these solutions. Then some solutions to the problem statements and how they can utilise this modified architecture will be discussed.

4.1.1 Shortcomings

The INFEX framework in its original form does only support a single session. It is not possible to connect the current working session to a remote session or even multiple remote sessions. Multiple instances of the INFEX framework should thus be able to join a single INFEX session. A remote connection should be able to be created with any other instance of the INFEX framework, thus will be using IP. There are a few requirements to be able to establish such a connection. The first one is that there needs to be a bidirectional connection between the different instances of the framework. Such bidirectional communication is needed to allow other clients to listen

to changes such as new devices entering a session, transfer requests and GUI specific events. To establish this connection it is required to posses the IP address of the remote client. However to prevent the *constrained possibilities* infrastructure problem introduced in [16], the method of establishing this connection should be designed with user experience in mind.
Remote connected instances also introduce a different problem namely access rights. A user might not want all their files to be available to all connected users. For some applications, access rights can become very complex and require role models [51]. However for the usage of this thesis this can be handled rather simply, users can either accept or deny file transfers from and to their devices.

To add a device to the current implementation of INFEX, users have to add a configuration file to the system. The implementation does not provide a GUI to do this, which is a clear usability problem. There is also no architecture component responsible for handling this action. Other similar actions are currently unsupported such as editing a device's connection information and removing devices. In the current architecture this is handled by the *Interface Lookup Service* and the *Communication Plugins*.

The third problem statement introduced in Section 1.3, identifies the problem that the INFEX framework currently does not support source and endpoint devices. Such as printers, scanners, webcams and televisions. These devices do not support the core action *transfer*, as they can only provide or receive information. The *list* action is unavailable for these types of devices as well, for example information is spontaneous provided for source devices, either by request (e.g. webcam) or on a specific event (e.g. scanner).

The current GUI implementation does not provide a lot of features as it is implemented as a proof-of-concept, it thus does not require a lot of functionality from the INFEX framework itself. In the current complementation, it only requires the 3 core actions from the framework. However when a GUI module wants to implement a more end-to-end experience to the users, for example allowing the user to add devices themselves and letting users connect to remote instances through the interface, it becomes more complex for a GUI module to call different components. This might introduce additional complexity, which might introduce problems related to the *constrained possibilities* HCI infrastructure problem.
INFEX also has the *High-level Abstractions* architectural requirement. In their paper Roels et al. specify:

> In order to support developers it is important that a framework provides a consistent high-level interface for accessing its functionality.

4.1.2 Architectural Modifications

To cope with these shortcomings, we introduce a modified INFEX architecture which is shown in Figure 4.1. There are three added components to the architecture: the session manager component and the device manager component and a GUI indirection layer. There are also two new actions in the core logic component.

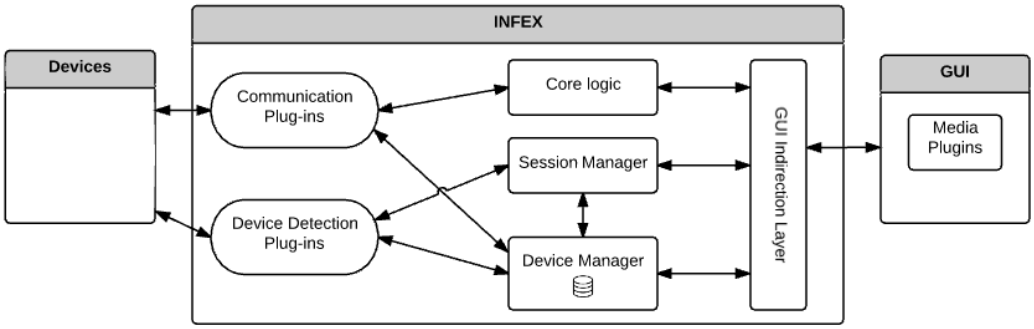


Figure 4.1: Modified INFEX architecture

Compared to the architecture introduced in Section 3.1, we introduce the *produce* and *expedite* core actions to the core logic. These are added to support information movement to endpoint devices and from source devices. *produce* is used to provide information from a source, this action will notify a GUI that a piece of information is available and will then provide the data. *expedite* is used to move information to an endpoint device, this is simpler as it is similar to the transfer action. However the Produce and Expedite actions also allow for streams of data to be handled. For example if a user wants to stream a webcam live feed to a television.
Just as with the List, Transfer and Inspect actions, communication plugins should implement the Produce and Expedite actions. However not each actions can be supported by each device. For example a printer protocol can only implement the Produce action. A GUI should thus be aware of which actions are supported by a device.

To support remote connected INFEX instances, a *Session Manager* is added. The *Session Manager* component will establish the connections with remote INFEX instances. It is also responsible for the management of the devices that currently connected to the framework. The detection plugins will notify this component, which will check whether the device is already setup and then add it to the current session. GUI modules can store some extra information such as the position on a tabletop in this session as well.

In the inverse direction, the *Session Manager* will also notify GUIs of new devices that are detected.

The Device Manager will store all information about the devices that are connected to the framework. The component will handle all CRUD (Create, Read, Update, Delete) operations on these devices. For each device, the supported communication plugins should be kept along with the specific connection information (e.g. Bluetooth MAC address).

The last architectural modification is the addition of the GUI indirection layer. An indirection layer allows for the extraction of information from a system when this information might contain many dynamic invocations and data flows [77]. The indirection layer allows for simple access to the information stored in the INFEX framework. It also separates the internal implementation (which might be subject to changes) from the GUI modules which makes these modules more robust.

4.2 Towards an Improved User Experience

In this section we will introduce the used methodology to form the solutions and then introduce the solutions for each problem statement. All of the solutions were implemented as a GUI module for the INFEX framework. Along with solutions for the problem statements, this GUI module also aims to provide a complete experience for the user. This means that a user can, in a user-friendly way, use all the features that the INFEX framework provides.

4.2.1 Methodology

To establish solutions for the aforementioned problem statements, we used a design methodology which is similar to that for creating user interfaces. As these problem statements are solved by creating a user interface prototype on top of the modified INFEX framework. The methodology used is based upon the course User Interface Design at the VUB and the corresponding book *User interface design and evaluation* by Stone et al. [65].

The first thing that has to be done in a good user interface design, is identifying the users. User classes are usually created in this step of the design process. However for this prototype the only distinct user characteristic is that the user posses more than one device. Thus defining a user class is trivial. Then usability and functional requirements had to be gathered. These were however simple for the problem statements as these had to solve them for the users. Because we wanted to create a prototype that provides the user with

a complete experience, we had to create some requirements that are related to the general use of the system. These are listed below:

- Add new device
  - Give a device a name
  - Check multiple communication plugins
  - Be able to edit details about multiple communication plugins
- Edit device
  - Edit the device name
  - Edit communication plugins settings
- Delete a device
- See a list remembered devices
- List files on a device
  - For each media type the files on a certain device should appear in a list
  - This list should be searchable
  - Filename, filesize and a thumbnail or type icon should be visible in the list
- List streams on a device
  - For each stream type, streams or streamable objects (music files which you can stream) should be visible
- Transfer a file to another device
- Expedite a stream to another device
- Inspect a stream in the interface
- Inspect a file in the interface
- View file transfer status, source and expediter
- View stream source and expediter
- Connect with remote tabletop

  - Setup connection using IP address
  - Save IP address as friend
  - Users can identify their own devices
  - Users can identify the owner of other devices when more than two users are present
  - Users can confirm a file transferred from their devices
  - Users can confirm a file transferred to their devices

- Virtual tabletop
  - Do all of the above
  - Set up a session
  - Add devices to a session
  - Close a session

- Groups of up to 4 people can use the interface at a single physical location

Note that these requirements, as well as all other derived resources, were created in an iterative fashion. When new requirements were identified during the creation of mockups, implementation or evaluation, these were added and the mockups and implementation was adapted accordingly.
A Concurrent Task Tree [52] was not created as the task structure is rather simple: all tasks can be done concurrently as they should be usable in a co-located setting. After the definition of the requirements, we proceeded with the design of the user interface. We started by creating user interface design mockups. These mockups represent how the interface should look like and identify the first usability problems. The mockups, along with a short description of what they model, can be found in Appendix A.
We then proceeded to create a working prototype on the interactive tabletop, which was used to perform a user study. Because the nature of the development, this prototype was not fully operational during the whole design process. Some of the functionality was implemented to be evaluated in a "Wizard Of Oz"-type of way. The goals, setup, execution and results of the user study can be found in Chapter 6.

4.2.2 Feature Design

This section is split according to the different problem statements that were introduced. Each of these has specific solutions and represents a feature that is implemented as part of the GUI prototype.

How to transform an INFEX tangible tabletop-based interface into a browser-based desktop application?

A user might want to transfer their information in an ad-hoc environment. For example when a user is at a family meeting and want to share some of their pictures taken at a trip using their GoPro. In this environment, we cannot assume that there is an elaborate setup, a user might only have access to their desktop or their laptop. In this use case they do not have access to tangibles nor an interactive tabletop. Users work either at their desk or using a mobile device (e.g. laptop, tablet or smartphone), so they are working either alone or in a over the shoulder environment where one person is interacting in the presence of other people.

The setup for this environment is thus different from the setup used in the original INFEX framework. However, it still requires that a computer is available to do the communication. The available detection tools are limited to those that are available in a typical desktop or laptop computer as we assume that no external tools are available for this. Detection of devices is therefore limited to determining whether or not a device is available for communication.
A setup like this still provides enough communication possibilities. Laptop, mobile and desktop computers nowadays provide support for a wide range of communication methods: all internet protocols, WiFi, Bluetooth and a set wired connections are supported by all of these devices. Some mobile devices even have IR and NFC capabilities.

To solve the problem of projection the tangible tabletop interface onto the usability-limited browser environment, we based ourselves on the usability principle of consistency. Users who have used a tangible tabletop interface should be able to easily adapt to the virtual interface. As mentioned in the previous chapter, consistency is one of the biggest usability problems in current user interfaces [46].

For the interface we analysed what the affordances of tangible tabletop interface are for a tangible tabletop interface and translated them to the classical user interface conventions users likely already master. These are two different concepts as discussed in [50], confusing them might lead into sloppy design which might confuse users.

A first affordance is that users can see the 3D shape of a tangible (in INFEX these are devices and representations of remote sources, the later will be introduced later in this section). The shape of a tangible might aid the user in various ways. In [62] these are described as follows:

> "*These do not directly contribute toward the overall goal (they are not functional), but help exploring options, keeping track of previous paths taken, and support memory.*"

Because the browser environment does not have these tangibles available (users might be mobile etc.) and we want to strive consistency, we chose to represent these tangibles virtually. This is done, as seen in the screenshot depicted in Figure 4.2, by an image of the device. Users should be able to add their own image, but in an interactive tabletop environment these images can be taken through the Kinect, which is part of that setup, and saved to the INFEX framework for use in the browser environment. These images might not have the affordances tangibles have. However the image might also aid recognition and support memory.

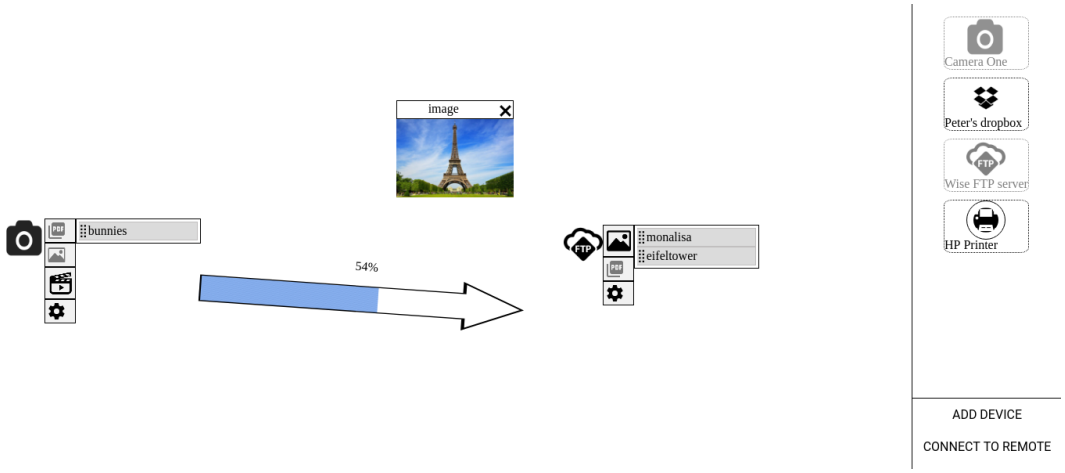


Figure 4.2: The desktop application implementation

Another affordance of tangibles is that they can be picked up and put anywhere on the interface. To cope with this we designed a separate space in the interface where the tangibles are represented (with images). The space can be seen as a representation of a drawer where the tangibles are stored. To convert the placing affordance, these virtual tangibles can be dragged around a virtual tabletop (which is the rest of the screen). We use the drag cursor icon to utilise this desktop convention as shown in Figure 4.3.

A benefit of physical devices is that you can more easily differentiate between physically similar tangibles. For example when there are two Ipads, people can easily identify which one belongs to whom: a simple tap on the home button will show the background and some other identifiable properties on the screen. In a browser application however this is impossible. A solution might be to prohibit the use of the same image for similar devices, however

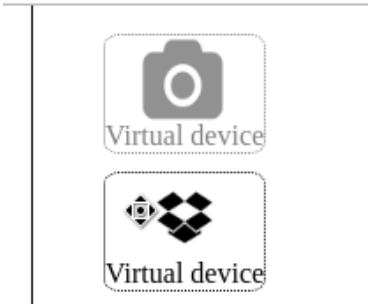


Figure 4.3: Example of two virtual tangibles and the drag cursor

this might interfere with the solution of taking images with the Kinect which was introduced earlier. The solution we pursued was the ability to give devices a name, which is rather simple. We could also use encodings (e.g. give them a colour), but this might require the user to remember which colour belongs to which phone. The use of a name instead, will be easier to recognise for users.

Users can also touch the interactive tabletop, which is not so much an affordance as a convention, users have learned (either by being told, observing or experimenting) they can touch the interface. To project this to the browser, we designed two solutions. The first is to keep the touch actions of the tabletop. Thus when the interface is opened on a tablet or other touch capable device (e.g. touchscreens are now available for laptops), the same interactions are still present. This contributes to the overall consistency of the system. The second solution is to leverage the cursor conventions as much as possible. For example by changing the cursor icon as mentioned previously in this section. Some gestures, such as these defined by Rotate´N Translate [37], which is used on tabletop interfaces to prevent usability issues with orientation, are unnecessary in the browser environment as the orientation is usually fixed (the screen stands vertically) and the user works alone, with a single input device.

How to represent device ownership on distributed tangible tabletop interfaces?

In Section 4.1.2 we already introduced how a user can connect to a remote tabletop on an architectural level. However connected tangible tabletops introduce another problem: device ownership. When a user connects their tabletop to one or more remote tabletops, it becomes increasingly difficult to differentiate which tangible belongs to whom. In our case these tangibles are devices or representations (introduced later in this section). It becomes

increasingly difficult since devices that belong to a remote user, cannot be represented as a tangible. Thus these tangibles have to represented virtually, similar to how devices are represented in the desktop application.

We introduce two different solutions to this problem. The first solution is based on the territorial behaviour on tabletops which has already covered in Section 2.2.1. The proposed solution consist of having virtual seating positions as recommended by [57], each remote user is virtually sitting at a side of the tabletop. Along with the virtual seating positions, each remote user has a personal zone where their tangibles (devices) are placed. The ownership of a device is thus implicit as users only have to identify who is positioned where around the table. However users are required to position their devices within their own personal zone, which might not be the case when a remote connection is established. Users might be in the middle of a session where devices are scattered around the tabletop. An animation is introduced to make clear that they have have to move their devices into a personal zone. We created a mockup that models this, as is seen in Figure 4.4.

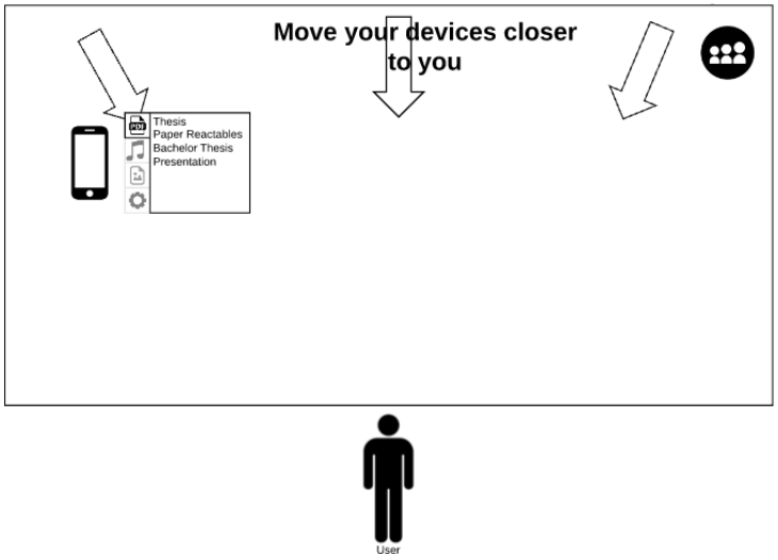


Figure 4.4: Mockup of the property animation

A second solution which was designed for this problem is the use of an encoding. An example of an encoding is colour or numbers, when using colour each tangible (device) is encoded with a certain colour which identifies the owner of the tangible. The encoding method is also used in [63] to identify ownership over notes using colours. This solution is quite easy to implement in virtual environments, but requires more thorough design for user on interactive tabletops as the aesthetics of a tangible cannot be changed (except

when using an overhead projector). For our design we thus chose to user a colour gradient. All the interface elements of a device, have an colour gradient overlay according to whom is the owner of the device. To model presence in this solution, we also used a colour gradient at each side of the tabletop as seen in Figure 4.5, each side thus represent a virtual seating position. Colour gradients are also used in [41] to model presence and interaction between different tablets.

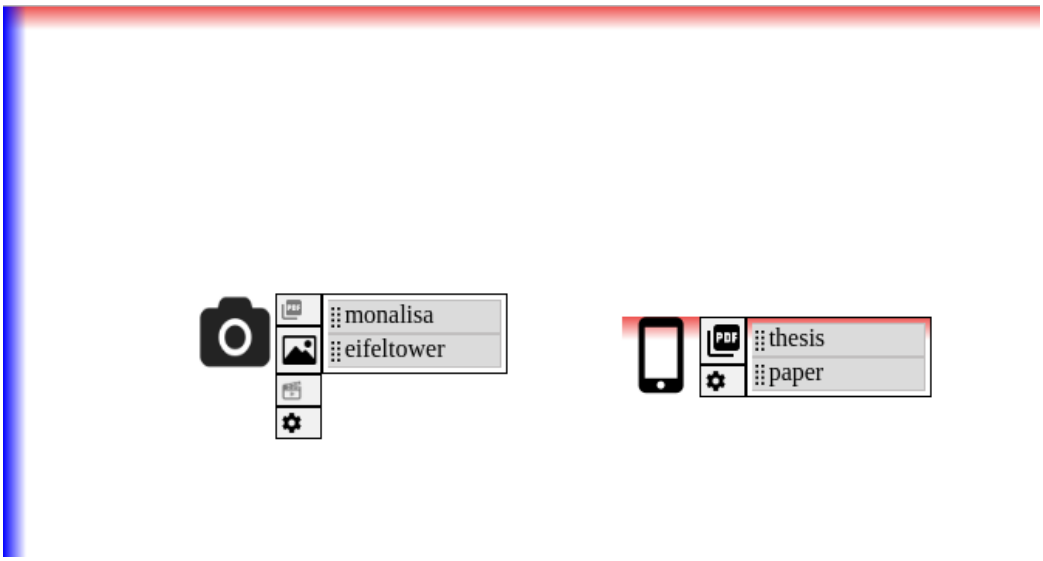


Figure 4.5: Two colour gradients indicating remote presence, a device with the same gradient indicates the remote ownership.

Note that both solutions introduced, limit the number of connected remote INFEX instances to three. This is because the tabletop has four sides, one of which the local user is standing. Only three usable sides remain on the table to show either a colour gradient or a personal zone. However, the consequences of this are rather limited as having more than three remote connected users is not expected. It also introduces more problems as it might confuse users.

These two solutions were compared in a user study (see Chapter 6). The two solutions both have to implement a simple access rights mechanism. Users need to feel in control of their files or they might be afraid to use INFEX. For this prototype we designed a simple access rights system that prompts the user with a dialogue whenever they want to transfer to or from a device that belongs to them.

How to implement and represent source and endpoint devices (which do not physically fit on the tabletop) on a tabletop interface?

For this problem statement we introduce two different solutions. The first solution is to use virtual representation of these devices. Instead of a tangible device, these devices would be represented as virtual devices similar to how remote devices are represented. However this solution has some issues, first could this solution confuse users, as there are both virtual devices which are remote and virtual devices which are local. This would also be inconsistent with the current implementation of INFEX where tangibles are used along with devices to represent remote information sources (e.g. Dropbox and FTP server), each information source (and endpoint) in INFEX is thus represented by a tangible. The second issue with this is that tangibles are loose 3-dimensional objects which enhances recognition and provide additional affordances as stated earlier in this section.
The second solution, which is the only one which was implemented and evaluated, is to create tangibles for these source and endpoint devices. These tangibles can be 3D printed or handcrafted. Many 3D models of everyday objects and devices are available for free on the Internet[1] which provides some additional enhancements. The shape of an object is easier to recognise in a 3D space and different encodings can be used. It is for example possible that in an office environment multiple of the same printers are available, we can use encodings to identify which tangible represents which printer. An example of a few tangibles which were created is shown in Figure 4.6.

[1] `https://archive3d.net/?tag=printer`

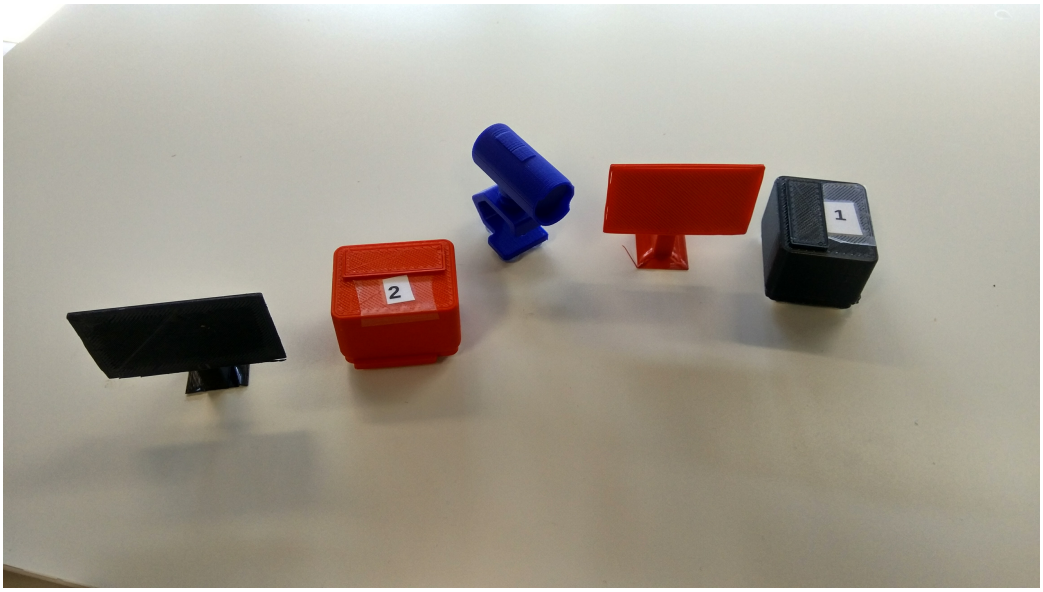


Figure 4.6: 3D printed tangibles

5 Implementation

5.1 Overview

To demonstrate the solutions introduced, a prototype was created. The prototype provides solutions for the three problem statement. It has an implementation of the personal zone and colour encoding for the problem of device ownership, detection of the tangibles and 3D printed tangibles are available and a browser version was created for use on a laptop. The prototype is built on top of the modified INFEX framework. The implementation details of the modifications will be discussed in Section 5.3. The focus of this thesis however lies in the implementation of the solutions to the problem statements, which is done in a GUI module which interacts with the INFEX framework that enables these interactions. In Figure 5.1 the prototype interface is shown on the tabletop. The implementation details of the GUI module is discussed in Section 5.2.

The implementation was tested using a Windows machine, using a TablerTV[1] interactive tabletop. The tracking of tangibles on this tabletop is done with a Kinect. However the implementation allows for different operating systems, tabletop and tracking hardware. This is due to the chosen technologies. A web application is implemented as GUI module and the INFEX framework

[1] `https://tabler.tv/touch-screen-table.html`

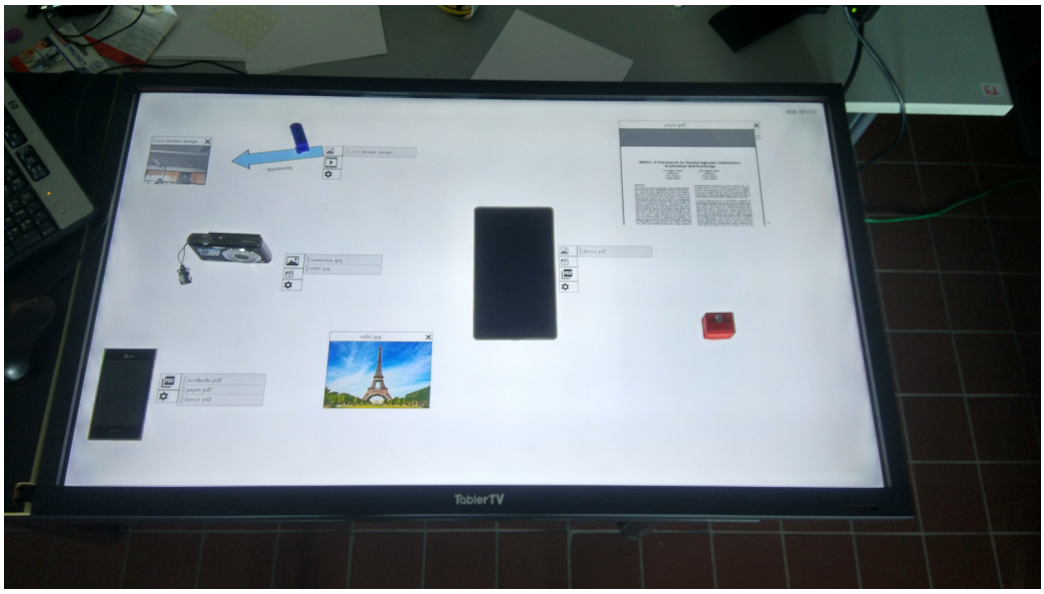


Figure 5.1: The prototype in action on the tabletop with devices and tangibles

is implemented as a Java application. A Java implementation allows for execution on all types of devices for running the INFEX framework and using the GUI module. Note that this need not to be done on the same hardware. For example the INFEX framework can run on a mini pc and a tablet can be used as interface. However in our tabletop setup, the GUI module and the INFEX framework are both ran on the same computer. A different tabletop can also be used. This is because of the use of TUIO [34] which has many implementation across different tabletop platforms[2].

5.2 Frontend

For the GUI module that is created, we chose to implement a web application. One of the reasons, as already mentioned, is that many devices have a browser which increases the portability of the GUI. We also chose to use the same code base for different devices, both on the tabletop and on devices that have the browser interface. It should however be noted that the application is not a responsive design [39], since we chose to map the tabletop interface in our virtual interface as discussed in Section 4.2.2. This choice was made to further increase consistency, as one of the guidelines in [45] suggests.

[2] `http://www.tuio.org/?software`

Because the magnitude of this GUI module, we chose to use a frontend web framework. Several possibilities presented, two major frameworks are currently in use ReactJS and AngularJS[3]. Both provide features that enable easy frontend development, such as templating, data binding and components. We however chose to use Web Components [2], a newly introduced W3C standard. This standard allows for the definition of new HTML-elements called web components. For our project these components are defined using ES6 [1] class syntax. ES6 is also used throughout the GUI module implementation as it provides a simpler syntax and some other features, for example the use of arrow functions which allows for a functional programming style. Because some components also require some specific CSS styling and specific JavaScript libraries, we used HTML imports [20] to create a clean dependency structure. When a component is created, it can be combined with its stylesheet and libraries in a single imported HTML file.
Because the used technologies are relatively new at the time of writing, many of them are not supported by many web browsers, some browsers are still considering implementation[4], others are refusing implementation due to disagreements about the standard[5]. We thus chose to target a single browser, namely Google Chrome. At the time of writing Chrome provides support for all of the used technologies[6][7]. The used version is Chrome version 58. However for these features there are also polyfills available[8], these are JavaScript scripts that mimic the default browser behaviour. Thus theoretically cross-browser support is feasible, however this is untested.
Web components have the advantage of reuse of code. This is important to INFEX as through these components, media plugins can be defined and reused. A media plugin is a plugin that allows for a certain mediatype to be inspected in the interface. To see the contents of a PDF, a PDF media plugin has to be implemented. The GUI module implementation provides an abstract media plugin class, called media component, which provides some auxiliary methods, for example a method that draws a header and a *_load*[9] method that loads media, while providing load percentage to a load callback

[3] `https://github.com/showcases/front-end-javascript-frameworks`

[4] `https://developer.microsoft.com/en-us/microsoft-edge/platform/status/htmlimports/`

[5] `https://hacks.mozilla.org/2014/12/mozilla-and-web-components/`

[6] `https://kangax.github.io/compat-table/es6/`

[7] `https://developers.google.com/web/updates/2014/05/Web-Animations-element-animate-is-now-in-Chrome-36`, `http://caniuse.com/#feat=imports`

[8] `https://github.com/WebComponents/webcomponentsjs`

[9] Because JavaScript does not provide built-in private methods, we use the convention that methods starting with '_' are private.

which is used to visualise the progress. An example of some media plugins are shown in Figure 5.2. Developers can easily add, extend and overwrite media plugins by creating their own web component class that overwrites this media component class, as displayed in Listing 5.1 where a media plugin is shown that displays images. The component should then be registered with the browsers built-in *customElements* object or by adding it into the *register.html* file, which provides the same functionality. A naming convention does apply to the media plugins, the tag name of the custom web component (which a media plugin is) should be of the form "*TYPE-component*", e.g. *PDF-component*.

```
1  class ImageComponent extends MediaComponent {
2
3      render(loadingCallback) {
4          if(!this.url && !this.dataSource) {
5              console.error("Cannot render this image, dont
                   have a datasource");
6          } else {
7              if(this.url) {
8                  this._drawHeader("image");
9                  var containerdiv = document.createElement('
                       img');
10                 this._load(u => containerdiv.src = u,
                       loadingCallback);
11                 containerdiv.style.maxWidth = "150px";
12                 containerdiv.style.height = "auto";
13                 this.appendChild(containerdiv);
14             } else {
15                 // Only display 1 image, a image cannot be a
                       stream
16             }
17         }
18     }
19
20     static getElementName() {
21         return "image-component";
22     }
23
24     constructor() {
25         super();
26     }
27 }
```

Listing 5.1: Image media plugin implemenation

Another reason why the reuse of web components is beneficial, is that

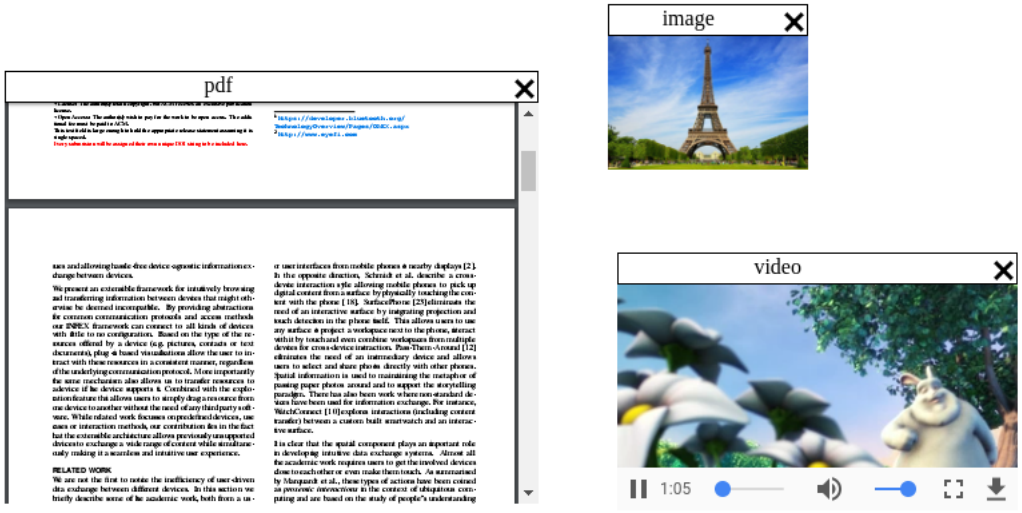


Figure 5.2: PDF, image and video media plugins

components can be shared with other developers[10]. Components can be installed through Bower[11], which is a package manager that packages HTML, JavaScript and CSS projects. To use them, a simple HTML import is required, which allows for a more consistent web usage. For example Google has defined a style guide called Material Design which it uses across all its services. Google also provides web components (implemented using Polymer 2.0) that implement this design guideline. An example of where this is used is the GUI module, where it is used for consistent dialogs and buttons. Web components thus also allow for quick development, which is very useful as we used a working prototype in our design process.

The architecture of the GUI modules is shown in the UML diagram, depicted in Figure 5.3. It is a UML diagram created according to the course *Object-oriented modelling* taught in the second bachelor computer science at the VUB[12]. Each class that is not abstract represents an element that can be added to the interface. Almost all of the interface elements are a subclass of the *TableComponent* class which provides certain features like orientation and positioning. These are needed to allow for fluid movements on the tabletop as is described in [5, 37]. In the architecture, two of the most important subclasses are shown. The first is the *DeviceComponent* which is the parent of both the *ActualDeviceComponent* and *VirtualDeviceComponent*. These classes represent the interface elements that have to be drawn to represent

[10] `https://www.webcomponents.org/`
[11] `https://bower.io/`
[12] `https://www.vub.ac.be/en/opleiding/fiches/90081`

devices in the interface. Because a lot of the functionality overlaps between these objects, an abstract class is created that contains functionality such as for example a circle animation which is shown when a file is hovered over a certain device, to indicate that it can be dropped. The only actual difference between virtual and actual devices is the way they are rendered. A virtual device needs an image to be shown, whilst an actual device has the tangible to represent itself.
The other important architectural part of the GUI module, is the *MediaComponent* which is the abstract parent class of all of the media plugins. The naming convention is also already introduced. This is required because when a certain file is dropped, the INFEX framework in its current form cannot prescribe which media plugin should open it. The implementation thus takes the type of file, which is also used to classify the files in tabs, and derives the suited media plugin which is then used.

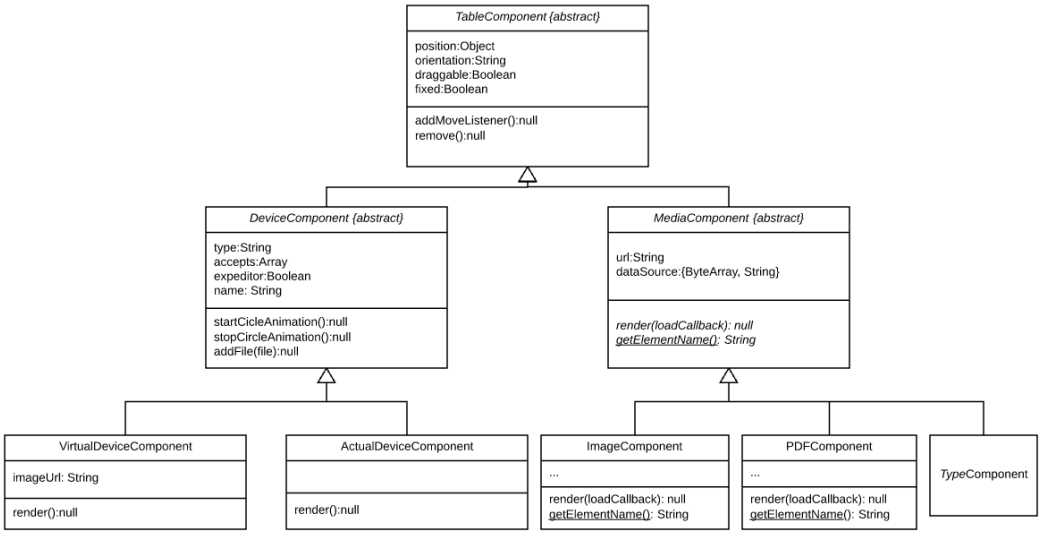


Figure 5.3: GUI module UML model

Because animations can improve the user experience [21, 64, 11], we tried to implement as many useful animations as possible. From the evaluations, we noticed that elements that appear and disappear might confuse the user. Thus we introduced a few animations to make these effects more noticeable and understandable. The first animation we added after an evaluation is when a user presses on a tab to view files, we added an animation that highlights the new files are shown. An animation is also shown when a user drags a file and hovers over possible targets, to indicate that the a drop of the file will start a file transfer or expedite the file. Another animation is shown when a user wants to connect to a remote and they have to place their

devices in their personal zone. Moving arrows will indicate that they have to move their devices into their personal zone which is shown in the coloured zone. Devices that are not already in that zone will wiggle to attract the attention of the user, as shown in Figure 5.4.

These animations are implemented using another new W3C standard called

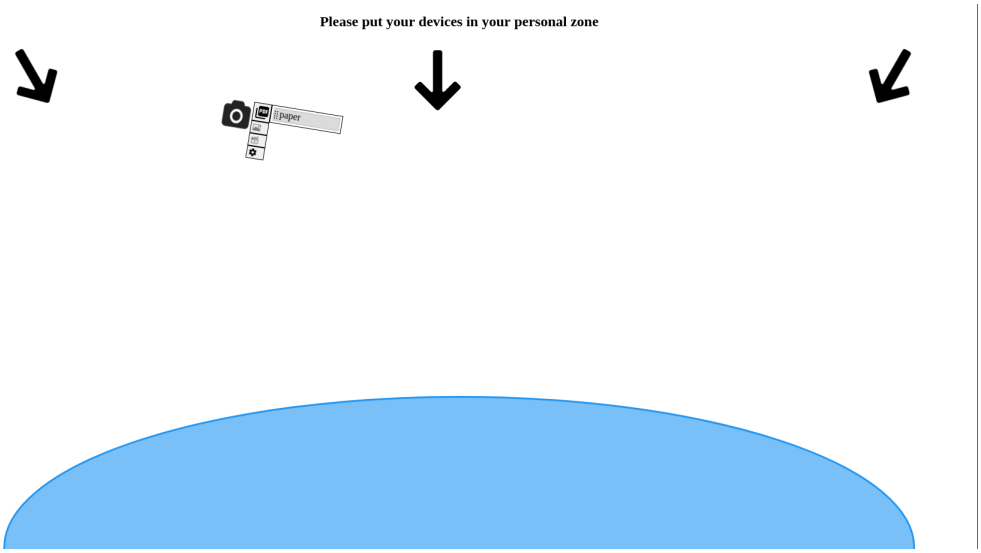


Figure 5.4: Animation shown when connecting to a remote instance.

Web Animations [3]. The Web Animations standard allows developers to create animations using keyframes. It was already possible to do this using the CSS Animations standard [27], which recently became an W3C Editors Draft, which means it is more mature and thus more likely to be implemented by a wider range of browsers. W3C Working Drafts can still be subject to change, which means that it is less likely browsers will implement it. The Web Animations standard, which is still a Working Draft, allows to use these types of animations in JavaScript. It also provides some playback functionality such that animations can be paused and replayed. This allows for integrated use with web components. Note that we also used CSS animations, which was done using the animate.css[13] library and used for the wiggle animation. Each of these animations is implemented either as integrated part of an interface component or as a separate class. For example the circle animation shown when a user can drop a file onto a device, is unique to device components and thus integrated within this component. The animation to show when a remote connects is not unique to a single component and thus integrated as a separate class.

[13] `https://daneden.github.io/animate.css/`

To allow for easy drag and drop of different elements in the interface (e.g. files, virtual tangibles, etc.), InteractJS[14] was used, which is a lightweight library that allows for drag and drop of the components. In the development cycle, we first chose to use the native HTML drag and drop API[15]. However, this implementation is lacking as it did not support touch-based drag and drop. There is however a polyfill[16] that supports touch based drag and drop, but this did not allow for multiple users to drag and drop at the same time. InteractJS however posed another problem: whenever a tangible is placed on top of the table, the library would assume that a hold action is ongoing and block all other interactions. To solve this issue we made the background draggable. However, this is not a permanent solution, a custom implementation for these interactions is thus required.

Between different interface components, there is also some behaviour that needs to be handled. For this different managers are created. The `DragManager` will handle all drag events. For example when a file is dragged, different targets should be notified of the finger or the mouse of the user (and thus the dragged file), to be able to show different drop zones. When the file is dropped, by releasing the mouse button or removing the finger from the surface, the `DragManager` will determine where the file is dropped and either show an inspect or trigger the transfer, expedite or produce actions from the INFEX core. The target (e.g. a VirtualDeviceComponent) should be calculated in real time as users need instant feedback of their actions, this is done using a JavaScript implementation of the Separating Axis Theorem[17] which is a fast algorithm that can calculate collisions between points, circles and polygons.

The `StateManager` manages the state of the tabletop. It will track all devices on the tabletop, current file transfers taking place and remote connections. Such tracking is needed such that other managers know what devices are currently on the tabletop, whether these devices are remote and to whom they belong. Remote devices may require different interactions.

To allow evaluations early in the development process, a `DemoManager` was implemented that mimicks some of the functionality that required a connection to the INFEX framework. Some functionality that this manager was responsible for, were for example file transfer, the different designed remote connection interaction techniques and a request for file transfer from a remote. New devices are added through the `NewDeviceManager`, this manager

[14] `http://interactjs.io/`

[15] `https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API`

[16] `https://github.com/Bernardo-Castilho/dragdroptouch`

[17] `https://github.com/jriecken/sat-js`

goes through the different steps required to add a new device to the system. A user first needs to enter a name for the device (using a virtual keyboard component on the tabletop interface), then needs to select one or more protocols that the device supports, as shown in Figure 5.5 and then needs to enter the specific settings for this device.

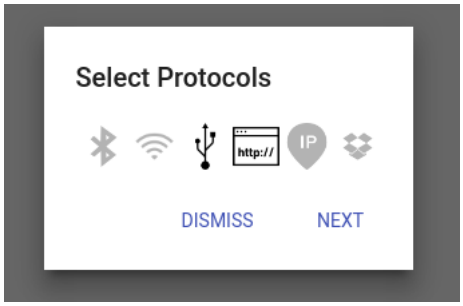


Figure 5.5: Selection of protocols

To identify the location of the tangibles in the tabletop interface, we have created a `TouchManager`. The goal of this manager was to identify the tangibles which was done by placing small rubber feet under the tangible in a perfect square. A perfect square is a figure that is hard to mimic using figures and thus hard to trigger by accident. However the accuracy of the used tabletop was not high enough, resulting in many unrecognised tangibles. Thus this approach was abandoned.
The `ConnectionManager` is responsible for all calls with the INFEX framework. These are done through a WebSocket. Each action (e.g. transfer) sends a message to the server over the socket. Because WebSockets support bidirectional communication, we cannot wait for a response like with HTTP calls. Therefore callbacks can be registered within the manager. These callbacks will be executed when a specific message is received from the INFEX framework. These callbacks can be execute once (e.g. for a list event) or many times (e.g. when streaming).
A `RemoteManager` is also created which handles the establishing and management of the remote connected INFEX instances within the interface.

The classical browser and tabletop interfaces are implemented using the same components and code, the only difference is the virtual tangible drawer on the browser interface. Because it is thus simpler to serve the same *index.html* and then differentiate between the classical browser interface and the tabletop interface, there needs to be a way to identify which interface has to be served. To do this a HTTP GET parameter is used. This is a parameter which is added to the URL of the request. To get the browser interface, the client has to add *?virtual=true* to the URL.

Another problem is the method that is used to connect to a remote INFEX instance. We can just asks users to fill in an IP address, but this is as stated in [16] not an optimal solution as it requires users to tamper with the infrastructure. But because we want to implement a peer-to-peer connection, without a central authority (e.g. connection server as used in the original version of Napster). There is a possibility to use a combination of URL and virtual hosts (e.g. *http://INFEX/connectToMe/?ip=123.45.67.89*), however this is not a cross platform solution, as different configurations are required for various operating systems[18]. We thus chose to implement a friend system similar to the one-time setup of devices in INFEX. Users have to enter their friends IP address and this will be remembered under the name of that friend. When a user wants to connect to another friend they can just select that specific user as is shown in Figure 5.6.

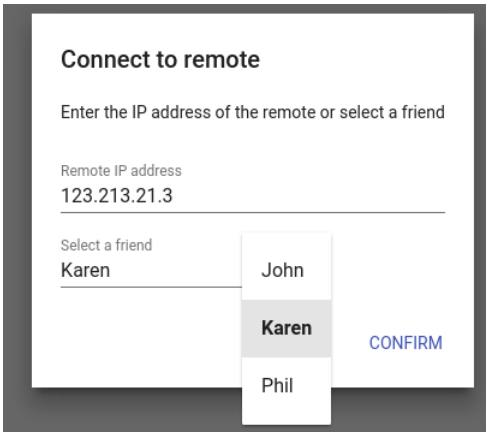


Figure 5.6: Users can select a remembered friend

The used tangibles were 3D printed using a Ultimaker 2+. 3D printing allows for many different designs, colours and sizes. Many devices have 3D models that are either created by the manufacturer or by other users, this allows for good representations of the actual device. However, there exist many different types of extensions. Cura, the software used with the Ultimaker 3D printers, only supports models in the *STL* format. Therefore conversion tools had to be used, some of these are available for free as web applications[19].

[18] `https://httpd.apache.org/docs/2.4/vhosts/examples.html`, `http://stackoverflow.com/questions/2658173/setup-apache-virtualhost-windows`

[19] `http://www.greentoken.de/onlineconv/`

5.3 Backend

The INFEX framework had to be modified as discussed in Section 4.1.2. The pre-existing implementation of INFEX was a Java implementation. There were implementations a HTTP, FTP and file system communication plugins. However these were implemented in a classical sequential fashion, which means that when a request from a client was received, methods were called in a sequential order, there was no possibility to attach code to a certain event in the implementation. For example for debugging purposes, it might be interesting to know when a communication plugin starts and finishes its read command. To implement general actions such as a transfer, where a read is followed by a write, a single method had to be written. This type of implementation does not allow for easy multi-threading.

To overcome this issue, an *EventBus* was introduced. We used the EventBus implementation provided by Guava[20]. An eventbus allows for events to be published on a central bus. All events are children of the *PubSubEvent* class, although it is not required they are direct children, for example we defined the *TransferEvent* abstract class, which is the parent of all events related to the transfer action. Different listeners can register to specific or more general events. The eventbus was used to implement a logging mechanism for the INFEX framework, the different INFEX core actions and the GUI indirection layer. A logging mechanism is an event listener that listens to the *PubSubEvent* class, which is the parent class of all events, thus this listener is executed on all events. Each event has to provide a message, which is printed to this log.
Because each of the existing actions in INFEX were implemented using sequential method calls, we had to reimplement these. Communication plugins have to send events like *ComPluginReadStart* when their read, write, list, stream and expedite methods are invoked, the latter two of these will be discussed later in this section. An inspect core actiontask will then listen to the *ComPluginReadEnd* event and publish an *InspectEnd* event. The events published by the list, transfer and inspect actions are then used by the GUI indirection layer, which is subscribed for these, to respond to clients requests. The event based method invocation, is shown in Figure 5.7. The figure shows that all method invocation in INFEX is done through these events, except for the core actions which are launched by the GUI indirection layer and launch the communication plugins. Communication plugins publish events regarding the state of their read, write, stream (discussed further in this

[20] `https://google.github.io/guava/releases/20.0/api/docs/com/google/common/eventbus/EventBus.html`

section) and expedite actions. The state can include progress report, error state, started state and end state. When streaming a communication can be instructed to stop through a *StopStream* event.
The *EventBus* used is made multi-threaded using the *AsyncEventBus* class, also provided by Guava, combined with an *Executer* which spawns a new thread for each listener matching an event.

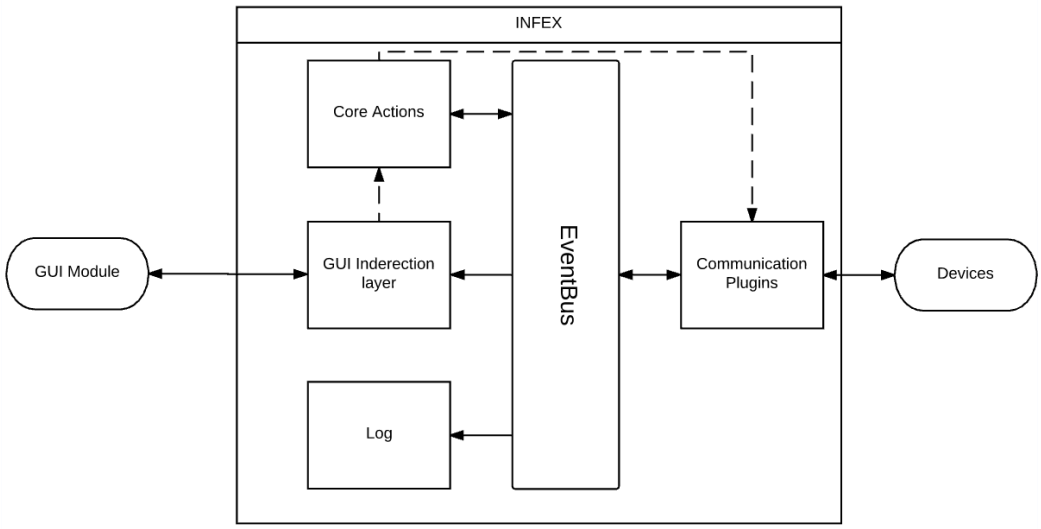


Figure 5.7: Method invocation structure in INFEX

Communication to clients is done through a WebSocket which allows for bidirectional communication. Client can send commands to the INFEX backend, these will invoke certain actions and when the right events are published, the client will receive data from the GUI indirection layer. Because sending data over WebSockets requires additional interface implementation, for example to handle buffering, we allowed GUI modules to add an extra parameter to the inspect action. The parameter allows for GUI modules to request that the inspected file is hosted on a Web server. Hosting files on a Web server allows Web clients to utilise the browsers built-in functionalities. For example if a user wants to inspect a video file on a tablet, the implementation of the GUI module can send the inspect command to the INFEX backend with the hosting parameter, the INFEX backend will then read the video file from the tablet and host the file on its local hard drive, where it can be accessed through HTTP by the GUI module.

To allow for source and endpoint devices, some modifications had to be made. First two new core actions were introduced *produce* and *expedite*, these allow for data transfer to output-only and from input-only devices.

The *CommunicationPlugin* interface was extended with two new methods: *produce* and *expedite*. Communication plugins are also no longer required to implement all of the methods (*produce*, *expedite* and *write*, *read* and *list*) defined in this interface.
It is also possible to pass streams of data through these methods. This is possible in two ways. A plugin may be able to host the stream itself and pass an URL to either the interface or another communication plugin, which is the case when an expedite task is used. If this is not possible, communication plugins can provide chunks of data by publishing *ComPluginStreamChunk* events. These chunks are contained in a *ContentUnit* class, which is a general format within in the INFEX framework and its also used for the results of *read* and *list* methods. The *ContentUnit* class contains some meta-information about the data it contains, such as the type (e.g. image or video) and file or stream name.

Three additional communication plugins were created to support the new functionalities of the INFEX framework. The first plugin is a printer plugin. This plugin only supports the *expedite* method which prints the data provided in the *ContentUnit*. The printer plugin is implemented using the Java *PrintServiceLookup* which is provided in Java SE[21].
A WebCam plugin was also implemented. This plugin provided both the ability to provide streams as files. The streams were provided as chunks to the rest of the framework. A video stream requires real time video encoding which is complicated to implement and requires a lot of resources. Each chunk is an image taken by the plugin. The plugin provides fifteen images per seconds. An external Java library was used[22].
The last implemented plugin is the TV plugin. Because we want to support the use on all types of televisions, a RaspberryPi was used. The RaspberryPi can be connected to all televisions which have a HDMI port. Communication plugins which do not require additional hardware. For this Smart TV applications could be created, but these are platform dependant. The use of wireless display (e.g. WiDi or Chromecast) is also possible. On the RaspberryPi we created a small web application which will display the data which is provided by the plugin. We reused the existing mediaplugin infrastructure discussed in Section 5.2 for this.

[21] `https://docs.oracle.com/javase/7/docs/api/javax/print/PrintServiceLookup.html`

[22] `https://github.com/sarxos/webcam-capture`

6 Evaluation

6.1 Goals

During the iterative design cycles, evaluation is important. It provides useful feedback to the developer which can enhance the user experience. At different stages of the design, different kinds of evaluations were performed. Because finding users to execute a user test is difficult and sometimes even expensive [65, 47], we opted to only do a user test to confirm that the proposed solutions solved the problem statements that were introduced. However during earlier and later stages of the design process, smaller evaluations were executed. These were done in several ways. One was to present mockups, task descriptions and the prototype to some of the researchers that work at the WISE lab, including the supervisors for this thesis. Some of the mockups and early prototypes were also shown to small sets of users which were easy to reach for an informal evaluation moment and during these informal evaluations, some usability problems were already discovered. Another way was to use the heuristics introduced in Section 3.2.

To test the solutions, we had to implement our solutions into a prototype. At the time of the evaluation, the prototype was still a stand-alone web application, which allowed for more rapid development since no modifications to the core INFEX framework had to be made. All devices were mimicked virtually and actions had no real effects. When the browser was

refreshed, all actions were undone. However all the solutions to the problem statements were present, this evaluation was a formative evaluation. The tangibles that represented various devices in the lab were available, including different encodings. Two different implementations to represent device ownership in remote tabletop interfaces, encodings and personal zones were implemented. The browser interface was implemented. We later discovered and addressed some shortcomings to the prototype, as will be discussed in Section 6.3.

Before designing the evaluation of this prototype, some goals were set. Two main goals were defined. The first goal was to get general impressions (e.g. what do users think about the implementation of the list, inspect and transfer actions) about the working prototype. The prototype already implemented some features such as the ability to add devices, but this had no real impact. Some of the core functionality such as transferring and listing files were mimicked. Getting feedback on the graphical implementation of these features could identify usability issues with these, which decreases global development time.
The second goal of this evaluation was to get confirmation that the proposed solutions accurately solve the problems stated in the problem statement. For the case of device ownership, where we proposed two solutions, we wanted to decide on which of these solutions is better suited for use in our prototype. The two competing solutions for device ownership were the encoding implementation and the personal space implementation. Another problem where we compared two solutions is the problem of representing source and endpoint devices. In this part of the evaluation we compared different encodings to each other.

6.2 Methodology

To identify usability problems with a user interface, many different tests exist as pointed out by Nielsen [47]. However for the evaluation of the prototype described in the previous section, we opted to execute a user test. Nielsen stated that a combination of both user test and the use of heuristics will identify most usability problems.

The user test was executed at the WISE lab at the VUB. Users were asked to participate voluntary and received no financial compensation for this user test. Before users started the user test, they were given a consent form which had to be signed. The consent form along with the questionnaire that was given to the users after the test, can be found in Appendix C. Users were recorded (both audio and video) during this user test. This was done

because there was only one observer. Because of the nature of the user test, not all observations could be made during the user test. These recordings were thus used to get all the data from the user tests.

Because the prototype was a stand-alone interface, many of the features had to be simulated. Therefore a Wizard of Oz setup was used. In this type of setup, interactions are simulated by the observer of the user test. For example remote connection requests were simulated in this way. To do this, a secondary screen was connected to the desktop computer that runs the interface. A keyboard and mouse was also connected. On this secondary screen the developer console of the Chrome browser was run, which allowed the observer to initiate certain interactions using the JavaScript interactive prompt. These commands were all using the `DemoManager`, which was introduced in Section 5.2. The complete setup is shown in Figure 6.1.

Figure 6.1: Evaluation setup. On the right the secondary screen for the simulating interactions. The user gets a short introduction about the goal of the user test.

Before the start of the user test, users were given a short introduction about the INFEX framework and the state of the current prototype. It was explained what the goals of the experiment were and that it was a test of the interface and its solutions and not a test of them, as some users might be scared to make mistakes when someone is observing them.
The user test was split in two parts. The first part was using the interactive tabletop. To meet the first goal, testing the general interface of the prototype,

the users were asked to execute a few tasks that were unrelated to the problem statements. Users were asked to identify a device, move it around in the interface, list all the images on a camera, transfer files and add a device to the system. Users were encouraged to think aloud and give comments, these were noted down by the observer along with perceived struggles and bugs that occurred. For example if a user had difficulty inspecting an image, which was necessary since not all users were expressive when they faced difficulties.

To evaluate the solutions to the problem statements and thus meet the second goal of the evaluation, users were asked to execute a number of tasks using the designed solutions. To correctly assess the results of these tasks and have a formal definition for each, usability requirements were created for each task:

Identify external device

- **Usability Requirement** Users can identify the tangible representing an external device
- **Measuring concept** Recognition
- **Measuring Method** Task scenario, success of identifying the right device and the time it takes to complete.
- **Planned level** Successful in 5 seconds.
- **Setup** Two printers with a certain encoding (colour or numerical) will be positioned in the lab and users have to identify the tangible that represents a specific printer. A number of different tangibles is available to the user.

Expedite file to an endpoint device

- **Usability Requirement** Users can expedite a file to an external endpoint device using the tabletop interface.
- **Measuring concept** Ease of use
- **Measuring Method** Task scenario, the time it takes to execute.
- **Planned level** Successful in 5 seconds.
- **Task scenario** Expedite the thesis PDF from a tablet to printer B.

- **Setup** A tablet computer with a number of PDF files will be on the tabletop interface. Two printers will be available and the user has to pick the right file and the right printer. Users will be able to try different implementations. Only their first attempt will be timed to prevent other attempts influencing the timing results.

Connect to a remote tabletop

- **Usability Requirement** Users can easily connect to another remote tabletop.
- **Measuring concept** Ease of use
- **Measuring Method** Task scenario, the time it takes to execute.
- **Planned level** Successful in about 25 seconds.
- **Task scenario** Connect to John Doe's tabletop.
- **Setup** The tabletop interface will be available. The user will be given John Doe's connection information.

Transfer to a remote tabletop

- **Usability Requirement** Users can easily transfer files to another connected tabletop.
- **Measuring concept** Ease of use
- **Measuring Method** Task scenario, the time it takes to execute.
- **Planned level** Successful in about 15 seconds.
- **Task scenario** Transfer the Eiffel tower picture to John Doe's phone.
- **Setup** The tabletop interface will be available. Users will be able to try different implementations. Only their first attempt will be timed to prevent other attempts influencing the timing results.

Approve file transport

- **Usability Requirement** Users are able to approve or disapprove file transport to another connected tabletop.
- **Measuring concept** Ease of use
- **Measuring Method** Task scenario, successfulness.

(a) Colour encoding (b) Numerical encoding

Figure 6.2: Two different encodings to identify a printer

- **Planned level** Successfully.
- **Task scenario** John Doe will ask to transfer a picture of the Mona Lisa. The user has to approve this transaction. Then John Doe will ask to transfer the PDF "top secret" which the user has to disapprove.
- **Setup** The tabletop interface will be available. At a certain moment the transfer will be requested and the user has to act as stated in the task scenario.

Users were asked to execute these tasks on the interactive tabletop in the same setup as for the first part of the tabletop evaluation.

For the task of identifying an external device, we chose the printers, as there are more than one printer in the lab which are visible form the location of the experiment. However, in a real-world example, this task would be easier as users would typically know which devices are available in their workspace. In Figure 6.2 the setup is displayed for both numerical and colour encodings.

The second part of the evaluation focused on the classical browser application. Users were asked to sit behind a laptop. To assess this part of the interface, users were asked to go through a cognitive walkthrough. In a cognitive walkthrough users are asked to execute an action sequence, while they have to explain what they are doing and why [74]. The main goal of the cognitive walkthrough method is to assess the ease of learning of a user interface.
Users were asked to do the same tasks as on the tabletop, but on the laptop browser interface. However, they were asked to executed some extra actions. The first action they were asked is to describe what they saw. We asked this question to test whether the consistent design actually impacted their recognition of the interface and to see if they can recognise the virtual drawer with the virtual tangibles as is. Then they were asked to put a new device

on the interface and transfer files from it. This action will reveal their understanding on how this works. The cognitive walkthrough method is here that the thought process is easily revealed and might gain key insights in design.

After the part on the laptop using the browser application, the users were told that the experiment was over. The recordings were stopped and the users were asked to fill in a questionnaire. The questionnaire contained two parts. The first was a standardised usability test. The test chosen was the SUS usability test [9], which was chosen based on its length and reliability.
After filling in the SUS test, users were asked to write down some positive, negative and general remarks about the system and the user study. All of the textual data that was collected during these user tests can be found in in Appendix B. In the appendix, the notes of the observer and the calculated SUS score, can also be found. Note that these have been made anonymous.

To ensure that users are not influenced by previous tests, we did a between group experiment. In our evaluation, users only get a single solution (e.g only the personal space solution). Because we have enough users, twelve, this is still relevant as five users for a single experiment is regarded as the minimum amount of people [65, 50]. All of the evaluations were conducted in Dutch but all participants had sufficient understanding in English, which allowed them to understand all text in the interface and questionnaire. In practice we gave the first six users a colour-based encoding for representing the printer and the last six a numerical encoding. The even (user number two, four, etc.) users got a colour encoding for the device owner problem and the uneven users got the personal space based solution for this problem.

6.3 Results

In total twelve users were observed during the user tests. All of the testers were students at the VUB between the age of 19 and 23. Half of the users were male and the other half female. Half of the users have a background or are currently studying computer science which makes them generally computer experts. Another half of the users had an other background and described themselves as "*Okay with computers*".

The average SUS score was 75,4, which tells us that the prototype would rank above average and could be interpreted as a good score as defined by [7]. For a prototype in development, this is an excellent result, as going into the evaluation it would be expected that there are still usability problems with the interface.

During the evaluation of our prototype, a number of problems with the user interface were identified. A problem all users faced, was that they tried

to press on a file in order to open it. However the interface only allowed for users to drag files around. When dropped on another device, a transfer would be initiated and when dropped anywhere else, the user would transfer it to the tabletop and thus be able to inspect it. However this was not clear to user and is not consistent with other user interfaces, where pressing on an icon or link opens an item. Related to this issue, many users tried to drag inspected files to devices to transfer them, which was not possible but has been added since.
Another problem which the users faced with the personal zone implementation was that the zone was not graphically represented (different to what is shown in Figure 5.4), which caused some confusion. A zone was thus added.
It was also noted that using a colour gradient in combination with the interactive table has little effect. Because the lighting conditions in the lab are not ideal, many reflections occur on the table, these gradients are not noticed by the users. The use of a colour encoding is hereby hindered as the colours confuse the users.
Some users also faced difficulties with the tabs that show up next to devices. In these tabs users can find the different files that are stored on a single device. The tabs thus graphically represent the *list* action of the core INFEX framework. These users did not notice that new files would display in the list. Therefore an animation was added and the tabs were positioned differently to ensure that users would notice these effects.
Because of the size of the tabletop and the used resolution, many users faced difficulties with the size of the buttons. As mentioned in [5, 61], interface elements should be large enough, which was not the case. This has also been improved in the next iteration of the prototype.

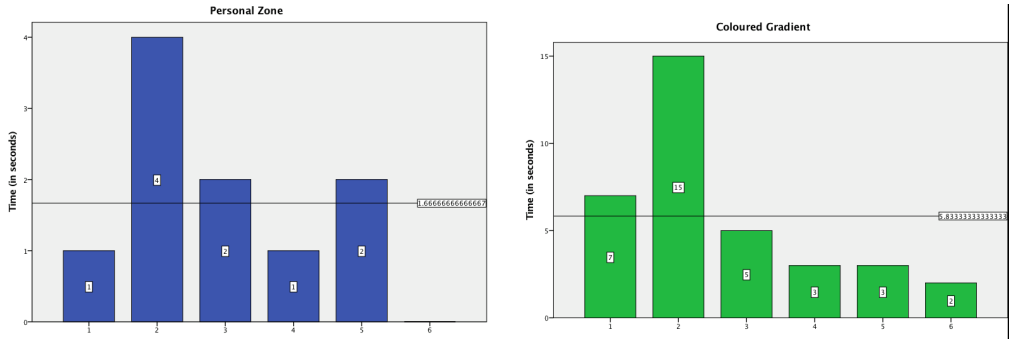
Users were also asked to give some positive, negative and general remarks about the system. Many users found that the idea behind INFEX was really good and indicated that they would use a system like this if it would be available. They confirmed that they had encountered the information exchange problem. Most of the users said that the application was easy to get used to, however some stated that some interactions (like the dragging instead of clicking) could be a little bit more easy. Users also said that the interface is fluent and responsive. However many users remarked that the prototype was unattractive and quite colourless. These are remarks on the aesthetics of the system, which can be modified in the future and is not the primary goal of the prototype. However, having an nice interface can influence usability, as many users prefer to work with aesthetically pleasing user interfaces, which is confirmed by Norman [49].

From the cognitive walkthrough we learned that the virtual drawer con-

cept is quite understandable. However some users (three out of twelve users) find it unclear. They are not able to explain the functionality of it without an introduction. Thus we added a help button which guides users about the use of the sidebar. Another solution to this problem would be to give the users an overlay with this explanation when they first use the system. However the interface is designed to be stateless, it does not require that users register an account, thus this is quite difficult to implement in practice. Users also stated that the use of the cursor icon was very helpful in the browser application, as they recognised it and were able to infer that the devices were draggable.

For the use of the tangibles, most users were able to successfully identify the tangible that represents a certain device. The encoding (numerical or using colours) did not impact the success rate much, when using numbers two users did not successfully identify the printer, while only one did not identify the printer using a colour encoding. However this could be a coincidence. The average time it took to identify using a colour encoding was 4.75 and the average time it took with a number encoding is 3.4, however as said with the number encoding the observer had to intervene twice, so these times were discarded. We thus consider that these encodings are both equal from a usability perspective. In this user test the difference was not big enough as we only have twelve total users. However one could argue that the colour encoding is worse as it impacts colour blind people.

Some time results were also gathered to compare the implementation of the personal zone and the colour gradient. All users were successful in understanding what happened when a remote user wanted to connect to their table. However when recognising who owns which devices there was a clear difference between the two designs, which is shown in Figure 6.3. The personal zone implementation shows device ownership more clearly than the colour gradient implementation. Users can recognise device ownership faster than when a gradient is shown. Therefore we chose this implementation when progressing in the development.



(a) Personal zone

(b) Colour gradient

Figure 6.3: Device owner recognition timing results of the user test

After this evaluation we continued development, but with the following conclusions that were provided by the evaluation:

- Using the personal zone implementation, with graphical zones to show device ownership.
- Making interface elements larger on the tabletop interface for easier touch interaction.
- Use colour encoding for tangibles representing source and endpoint devices.
- Keep using the tangible drawer but add a help menu to help users who do not immediately understand its purpose.
- Keep the dialogues used for adding devices.
- Make text on the arrow indicating a transfer bigger, keep this arrow as users indicated it helped them understand their actions.
- Added animations to the files tab to highlight new files showing up.

7 Conclusion

7.1 Summary

In this thesis we have implemented and evaluated a new front-end user interface as well as various backend modifications to address existing shortcomings and introduce additional features into the INFEX framework.

We started by introducing the information exchange problem in Section 1.1 and the impact onto regular users workflow was highlighted. The INFEX framework is introduced. We identified that the INFEX framework faces some problems regarding user-friendliness. To solve this two categories of problems had to be solved: usability problems with the core functionality of the INFEX framework and features that were lacking in the INFEX framework. In this thesis we introduced three problem statements that each represent a missing feature: How to represent source and endpoint devices on tangible tabletop interfaces, how to represent device ownership on a connected tangible tabletop interface and how to represent a tangible tabletop interface to a classical browser application on a desktop. These problem statements are not limited to the INFEX framework and contribute to the field of tangible tabletop user interfaces.

Some related work was introduced. Some frameworks, systems and methods were introduced that solved the problem of discovering devices. Systems that tried to solve the information exchange problem or provided features

that might facilitate this problem were introduced. We also presented different tabletop interfaces. We focussed on tangible tabletop interfaces and connected tabletop interfaces as these face similar problems as those that are defined in the problem statement. These presented some design clues and ideas how to tackle the introduced problems. To come up with solutions to the problem statements, we introduced the INFEX framework in detail. Related guidelines and heuristics were also discussed. These were then used, along with the related work, to design solutions to the problem statements.

For each of the problem statements we designed one or more solutions. To solve the problem with projection a tangible tabletop on a desktop application, we identified the affordances of a tangible tabletop interface and translated these into UI conventions for these desktop environments. We thus implemented a virtual tangible drawer to store virtual tangibles in. Two solutions were designed to solve the device ownership problem on connected tabletop interfaces. A solution using colour encodings and a solution using territorial zones. The latter was deemed more effective after a user test. Furthermore we introduced 3D tangibles to represent source (input-only) and endpoint (output-only) devices. These tangibles could be placed onto the interactive tangible tabletop interface and used as any other tangible in INFEX.

A user-centered design approach was used to create a new GUI module for the INFEX framework. A working prototype approach was used. The prototype was implemented using modern Web technologies and allows for extensibility for different media plugins which show different types of media content. In order to implement the proposed solutions to the problem statements, some modifications to the INFEX framework were required. Therefore a modified INFEX architecture was designed and implemented. Along with these, some additional communication plugins were implemented. Such as a printer and webcam plugin. The implemented modifications to the core framework allowed the use of streams of data, device management, compatibility with source and endpoint devices, a decoupled GUI through an indirection layer and the possibility to connect to a remote INFEX instance.

To evaluate the different solutions that were designed for both the problem statements and the usability issues that INFEX faced, user tests of different magnitudes and during different phases of the development process were executed. These were used to enhance the working prototype which is the result of this thesis. The results and the impact of these results of these tests were introduced and discussed.

7.2 Discussion

The limitations of the implemented prototype are bound to the limited implemented communication plugins. Because of the flexibility of the INFEX framework, it is possible to use a wide range of device with the prototype. However this requires that many different communication plugins are developed. We can also implement new and additional detection plugins to further improve detection.

A limitation that the implementation surrounding remote connected tabletops is that we can only connect up to three remote tabletops. This is due to the amount of sides the table has. Other research is also limited to a small number of users at a single tabletop.

The implemented GUI module can be further extended to provide more functionality. For example a more elaborate access rights system can be implemented. Where users can define groups, access levels, etc. This would however require thoughtful design, as this might make the interface more complex. Another feature that could be implemented is the ability to allow GUI modules to edit and delete files on the devices. This would only require two more core actions, a *delete* and a *edit*, which have to implemented by the communication plugins. However one could argue that this is not part of the information exchange problem and thus lies beyond the scope of the INFEX framework.

Something that is a part of the information exchange problem is synchronisation. Users might want to automatically store files on different devices and when a file is added to one device, it should be automatically stored on another device as well. A way to automatically synchronise is currently not possible from within INFEX. One could create a GUI module that implements this through the GUI indirection layer, but this requires that this GUI module is running. A more elegant way would be to incorporate this into the INFEX framework. However this also has an impact on the graphical interface.

In the current implementation of the GUI module depents on callbacks to handle events and asynchronous method calls. However this has some drawbacks such as less readable code, hard to catch exceptions and callback hell[1]. To solve these issues, the implemenation could be updated such that it uses the promise API[2].

To enable user-friendly remote connection, the current friend list implementation is not ideal as it still requires users to enter an IP address. Re-

[1] `https://strongloop.com/strongblog/node-js-callback-hell-promises-generators/`

[2] `https://developers.google.com/web/fundamentals/getting-started/primers/promises#promise-api-reference`

search can be done to incorporate solutions involving RFID, NFC, QR and Bluetooth. Users might give their friends a special RFID tag which enables them to connect to their INFEX instance. A similar exchange can be done through QR codes. This might increase the usability of this feature.

A

Appendix A: Mockups

Connecting the Raspberry
Connecting the Raspberry
Connecting the Raspberry
Icon Pressed

Figure A.1

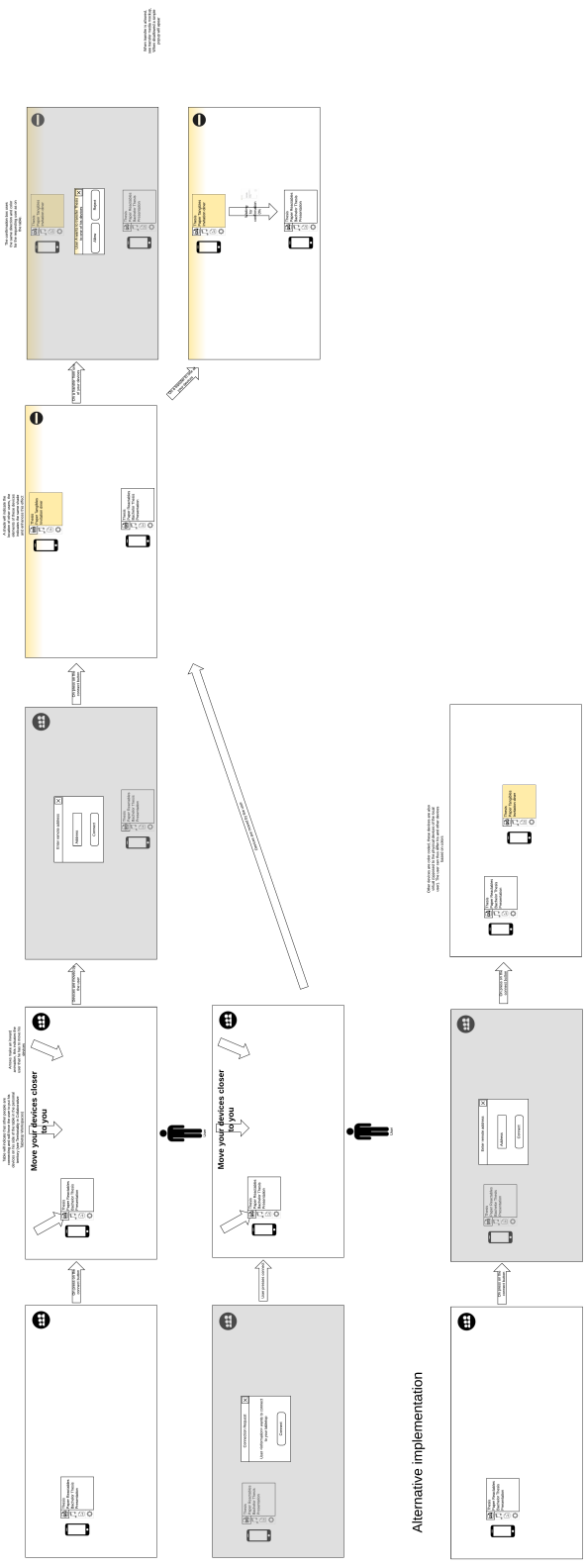

Move your devices closer to you
Move your devices closer to you
Alternative implementation

Figure A.2

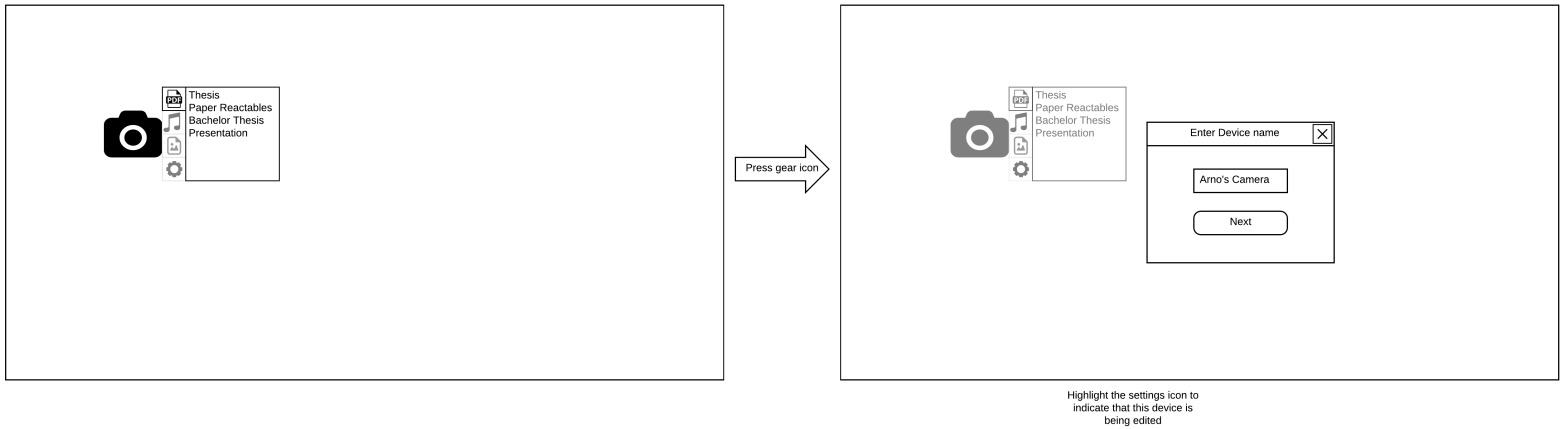

Thesis
Paper Reactables
Bachelor Thesis
Presentation
Press gear icon
Thesis
Paper Reactables
Bachelor Thesis
Presentation
Enter Device name
Arno's Camera
Next
Highlight the settings icon to indicate that this device is being edited

Figure A.3

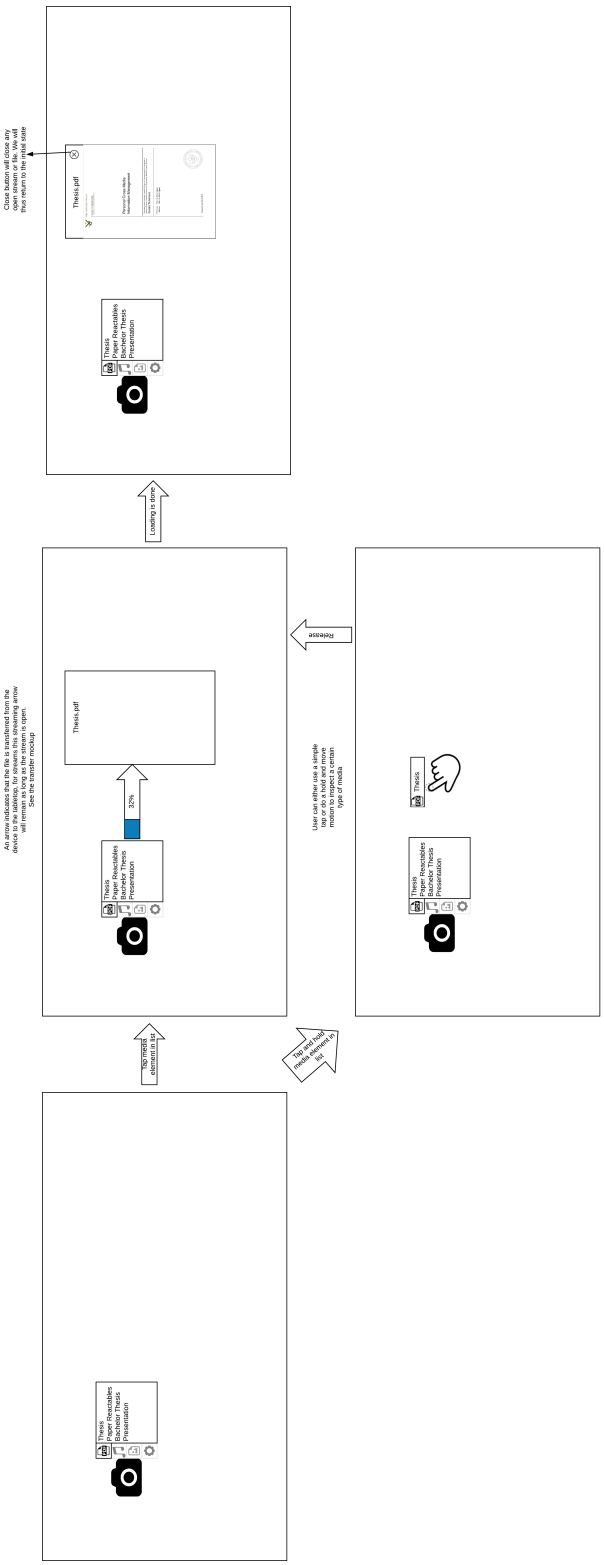

Thesis
Finger Reactables
Bachelor Thesis
Presentation
Tap and hold element to lift
Tap and release to drop
User can either use a simple tap or hold and make motion to inspect a certain type of media
Release
Thesis
Finger Reactables
Bachelor Thesis
Presentation
Thesis.pdf
32%
An arrow indicates that the file is transferred from the device to the website. Meanwhile the transfer screen will remain as long as the stream is open. See the transfer mockup
Loading is done
Thesis.pdf
Close button will close any opened media file. Website will thus return to the initial state
Thesis
Finger Reactables
Bachelor Thesis
Presentation

Figure A.4

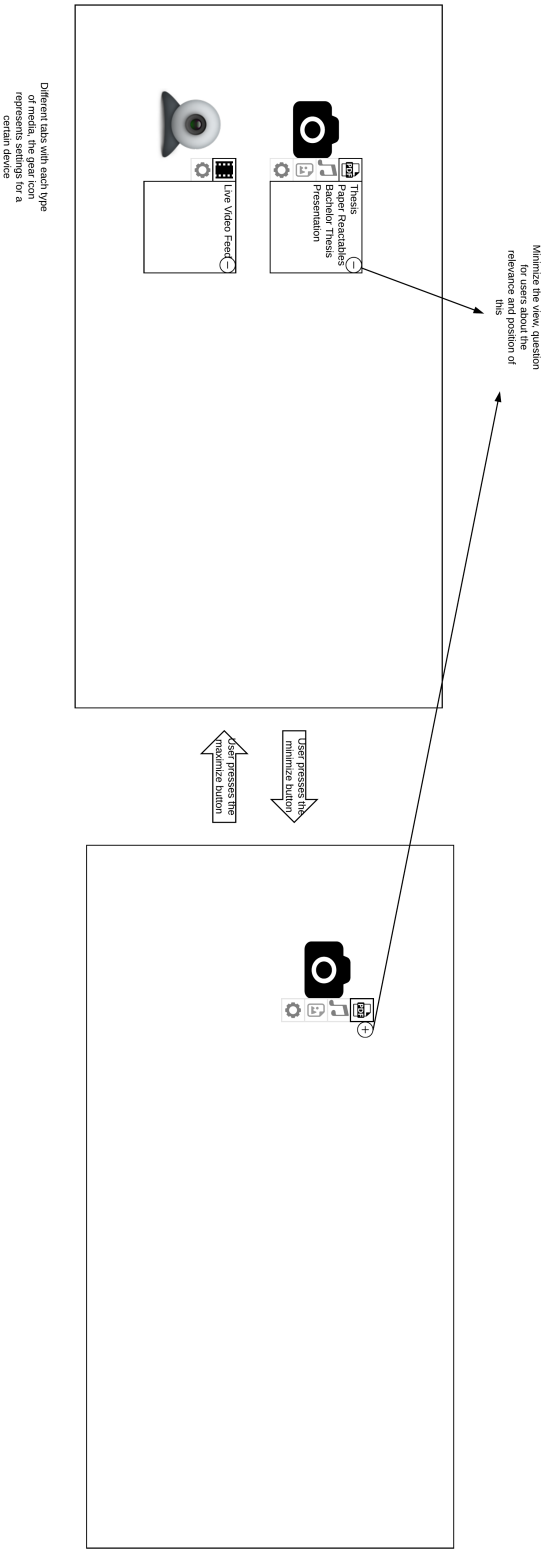

Different tabs with each type of media, the gear icon represents settings for a certain device
Live Video Feed
Thesis
Paper Reactables
Bachelor Thesis
Presentation
Minimize the view, question for users about the relevance and position of this
User presses the minimize button
User presses the maximize button

Figure A.5

To inidicate a file transfor an arrow is drawn from the source device to the target device
32%
For streams a live preview would be shown above the arrow, an animation in color through the arrow would indicate the "flow" of information

Figure A.6

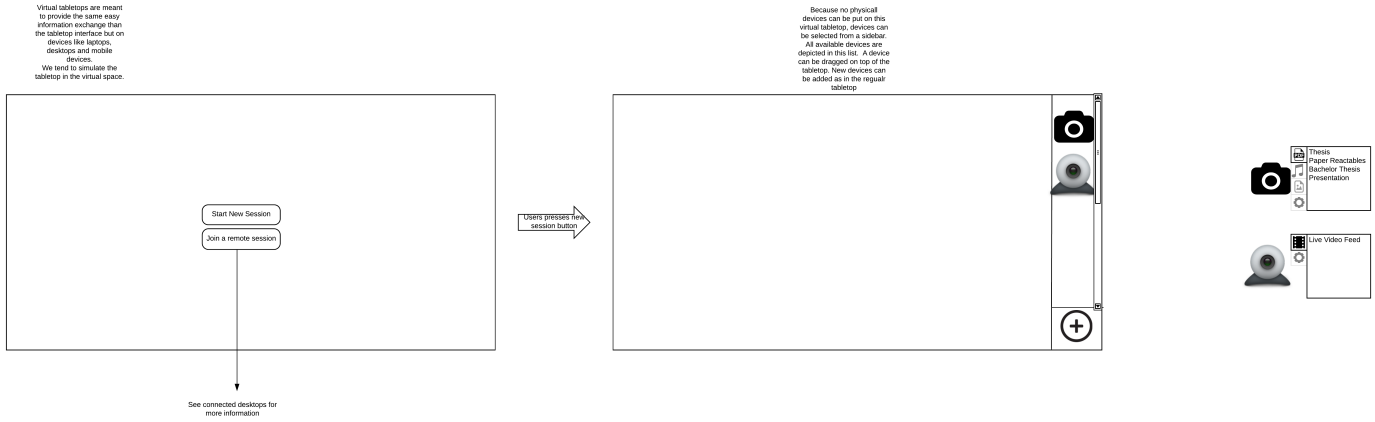

Virtual tabletops are meant to provide the same easy information exchange than the tabletop interface but on devices like laptops, desktops and mobile devices.
We tend to simulate the tabletop in the virtual space.
Start New Session
Join a remote session
See connected desktops for more information
Users presses new session button
Because no physicall devices can be put on this virtual tabletop, devices can be selected from a sidebar. All available devices are depicted in this list. A device can be dragged on top of the tabletop. New devices can be added as in the regualr tabletop
Thesis
Paper Reactables
Bachelor Thesis
Presentation
Live Video Feed

Figure A.7

B

Appendix B: Evaluation Results

Participant number 1 - 25/04/2017
Comments from observer:

- Dragging the PDF files unclear
  - Just clicking on file
- Printer representation: opening
- Printer information: feature request
- New device enter
- Fake remote 1 not clear enough
- Bug file transfer same device in remote
- Which file is unclear
- Clicking on virtual was more clear
- Handy cursor indicator

Comments from user:
Positive:
/
Negative:

- Printer did not actually print
- No information about the printer
  - Pages

  - Ink levels
  - ...

General Remarks
/
SUS score: 80

Participant number 2 - 25/04/2017
Comments from researcher

- Click to open PDF
  - Dragging is hard
- Print failed
  - Stops dragging file too early
- Identify is hard
  - Colours are confusing
  - Seem different colours depending on the view of the table
- Edges of table
- Bug position of arrow
- Devices in virtual interface not immediately clear

Comments from user
Positive

- System is easy to use and get used to

Negative

- Right column in virtual interface is unclear

General Remarks

- Colour schemes for second user could be more clear

SUS score: 77,5

Participant number 3 - 25/04/2017
Comments from researcher

- Opening PDF was difficult
- Icons were not familiar
- Bug beneath touchpoints from tangibles
- 1 touch difficult
- Remote confirm was unclear
- Virtual part still unclear
- Video unclear
- Clicking on files (see remark 1)

Comments from user
Positive

- Blue arrow was very clear

Negative

- You have to push quite hard on the table
- Difficult to open video file

General Remarks
/
SUS score: 57,5

Participant number 4 - 25/04/2017
Comments from researcher

- Clicking on files instead of dragging
- PDF inspect printing
- Intention of connecting unclear
- Red gradient not completely clear
- Confirm not completely clear
- Virtual part still unclear
- Bug virtual part has not been greyed out yet
- Video unclear
- Tabs unclear

Comments from user
Positive

- Handy system

Negative
/
General Remarks

- It would be good if you can drag (for example) an open PDF file to the printer

SUS score: 77,5

Participant number 5 - 26/04/2017
Comments from researcher

- Clicks on PDF
- Did not notice the colour (remote user)
- Bug table placement
- Wrong device
- Put back virtual devices

Comments from user
Positive

- Fun idea to connect multiple devices to each other, especially in the upcoming IoT era

Negative

- Hardware still too primitive to optimally use the program
- UI is unattractive

General Remarks
/
SUS score: 70

Participant number 6 - 26/04/2017
Comments from researcher
- Clicking on PDF
  - Not intuitive
- Bug involving tangible

Comments from user
Positive
- Very intuitive

Negative
- No information about what is being transferred from one device to another
  - Information could be in the arrow that is displayed

General Remarks
/
SUS score: 82,5

Participant number 7 - 26/04/2017
Comments from researcher
- Clicking instead of dragging
- New device invisible
  - Different button colour
- Push printer for more information

Comments from user
Positive
- Once you learn that you have to drag files instead of clicking, everything becomes very consistent and intuitive
- Easy to use
- Gestures and actions are easy to remember

Negative
- Could be more colourful
  - Attractiveness UI
- Bigger 'button' for add device

General Remarks
- Easy to use

SUS score: 87,5

Participant number 8 - 26/04/2017

Comments from researcher

- Clicks on PDF
- Zoom in inspect mode
- New device is nowhere to be found
- Pushes on tangible
- Does not notice red glow (remote user)
- Dragging on virtual interface is not natural
- Confirms virtual
- Clicks on virtual device

Comments from user

Positive

- In general: an intuitive system
- Virtual device (on laptop) resembles table very good
  - Not confusing

Negative

- Uses only dragging, almost no tap or double tap
  - I would intuitively open an item by double tap
- No possibility (yet) to enlarge/reduce size of items

General Remarks

- I do not think replacing bigger items (for example printers) for tangibles is a good idea. You will always have to make extra tangibles and keep them safe in a box.
  - It would be more logical to make them virtual (or automatic setup when connected to the network)

SUS score: 85

Participant number 9 - 27/04/2017

Comments from researcher

- Clicks on PDF
- Dragging PDF inspect
- Bug with placing tangible
- Field to show personal zone
- Virtual devices not completely clear
  - Maybe not the best names

Comments from user

Positive

- Everything works smoothly
- Smooth previews
- Very responsive

Negative

- No colors

- No sounds to accompany actions/gestures
- No animations to accompany actions/gestures
- Icons (categories) could be bigger

General Remarks

- Very strong concept, would want to have this at home
- Student in Computer Sciences

SUS score: 77,5

Participant number 10 - 27/04/2017

Comments from researcher

- Clicks on PDF
- Pointing out tangible
- Printing goes wrong

Comments from user

Positive

- Transferring files between devices is very simple
- Adding a new device is very clear

Negative

- Opening files would be more logical if you could tap/click instead of dragging
- Printing goes wrong

General Remarks

/

SUS score: 77,5

Participant number 11 - 27/04/2017

Comments from researcher

- Clicks on PDF
- Remote connection
  - Team viewer
- Personal zone unclear
- Did not notice confirm
- Virtual devices did not work right away

Comments from user

Positive

- You can see the transfer visually (arrow)

Negative

- Personal zone is not quite clear
- Difficult to see own devices
  - Edge on virtual table

General Remarks

/

SUS score: 67,5

Participant number 12 - 27/04/2017

Comments from researcher

- Clicks on PDF
- Opens PDF inspect to print
- Tab is unclear
  - Apple/Mac style would be better

Comments from user

Positive

- Logic behind the system is easy to follow
  - Once you understand how one thing works, you can apply this logic to the rest

Negative

- Printer tangible would be more clear if there was a number on the printer
  - Shape of printer was useful to find the right tangible

  NOTE: was looking at wrong printer

General Remarks

- Useful system
  - Especially since you do not have to have a table at home (you can use a virtual table)
- Icons are very clear to understand, no matter what language you speak

SUS score: 65

C

Appendix C: Consent Form

User Testing Informed Consent Form

Study administrator is: _____

Participant is: _____

Participant number: _____

This is a study about INFEX. Our goal is to make the INFEX tabletop interface appealing, intuitive and user friendly. Your participation will help us achieve this goal.

In this session you will be working with working prototype. We'll ask you to perform tasks a typical user might do, such as transfer media files acros different devices. A member of the design team will sit in the same room, quietly observing and taking notes. A facilitator will sit near you and help you if you are stuck or have questions.

All information collected in the session belongs to WISE research group at the VUB and will be used for internal purposes. We will videotape and audiotape the session. We may publish our results from this and other sessions in our reports, but all such reports will be confidential and will not include your name.

This is a test of the software. We are not testing you. We want to find out what aspects are confusing, so we can make it better. You may take breaks as needed and stop your participation in the study at any time.

Statement of Informed Consent

I have read the description of the study and of my rights as a participant. I voluntarily agree to participate in the study.

Print Name: _____

Signature: _____

Date: _____

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

D

Appendix D: INFEX Architecture

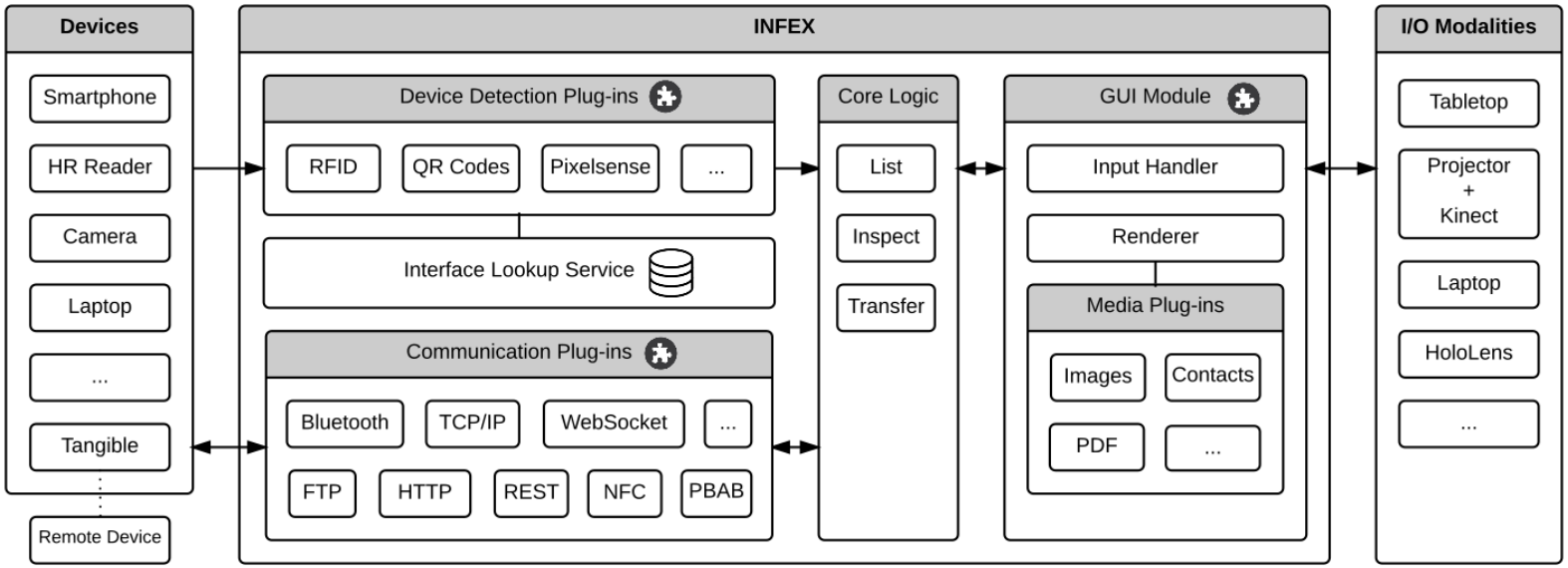


Figure D.1: INFEX architecture

Bibliography

[1] ECMAScript® 2015 Language Specification. Technical report, Ecma International, 2015. https://www.ecma-international.org/ecma-262/6.0/.

[2] Custom Elements. Technical report, W3C, oct 2016. https://www.w3.org/TR/2016/WD-custom-elements-20161013/.

[3] Web Animations. Technical report, W3C, sep 2016. https://www.w3.org/TR/2016/WD-web-animations-1-20160913/.

[4] Alissa N. Antle and Sijie Wang. Comparing Motor-cognitive Strategies for Spatial Problem Solving with Tangible and Multi-touch Interfaces. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, pages 65–72. ACM, 2013.

[5] Trent Apted, Anthony Collins, and Judy Kay. Heuristics to Support Design of New Software for Interaction at Tabletops. In *Proceedings of CHI Workshop on Multitouch and Surface Computing*, 2009.

[6] Trent Apted, Judy Kay, and Aaron Quigley. Tabletop Sharing of Digital Photographs for the Elderly. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 781–790. ACM, 2006.

[7] Aaron Bangor, Philip Kortum, and James Miller. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies*, 4(3):114–123, 2009.

[8] Jacob T. Biehl, William T. Baker, Brian P. Bailey, Desney S. Tan, Kori M. Inkpen, and Mary Czerwinski. Impromptu: A New Interaction Framework for Supporting Collaboration in Multiple Display Environments and Its Field Evaluation for Co-located Software Development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 939–948. ACM, 2008.

[9] John Brooke. SUS - A Quick and Dirty Usability Scale. *Usability Evaluation in Industry*, 189(194):4–7, 1996.

[10] Stéphanie Buisine, Guillaume Besacier, Améziane Aoussat, and Frédéric Vernier. How Do Interactive Tabletop Systems Influence Collaboration? *Computers in Human Behavior* , 28(1):49–59, 2012.

[11] Bay-Wei Chang and David Ungar. Animation: From Cartoons to the User Interface. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, pages 45–55. ACM, Sun Microsystems, Inc., 1993.

[12] François Coldefy and Stphane Louis-dit Picard. DigiTable: an Interactive Multiuser Table for Collocated and Remote Collaboration Enabling Remote Gesture Visualization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE Computer Society, 2007.

[13] Anthony Collins and Judy Kay. File System Access for Tabletop Interaction. In *Tabletops-Horizontal Interactive Displays*, pages 335–355. Springer, 2010.

[14] Raimund Dachselt and Robert Buchholz. Natural Throw and Tilt Interaction Between Mobile Phones and Distant Displays. In *Proceedings of Conference on Human Factors in Computing Systems*, pages 3253–3258. ACM, 2009.

[15] Paul Dietz and Darren Leigh. DiamondTouch: A Multi-user Touch Technology. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, pages 219–226. ACM, 2001.

[16] W. Keith Edwards, Mark W. Newman, and Erika Shehan Poole. The Infrastructure Problem in HCI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 423–432. ACM, 2010.

[17] Melissa A. Federoff. Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games. Master's thesis, Indiana University, 2002.

[18] Luca Frosini, Marco Manca, and Fabio Paternò. A Framework for the Development of Distributed Interactive Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 249–254. ACM, 2013.

[19] Hans Gellersen, Carl Fischer, Dominique Guinard, Roswitha Gostner, Gerd Kortuem, Christian Kray, Enrico Rukzio, and Sara Streng. Supporting Device Discovery and Spontaneous Interaction with Spatial References. *Personal and Ubiquitous Computing*, 13(4):255–264, 2009.

[20] Dimitri Glazkov and Hajime Morita. HTML Imports. Technical report, W3C, 2016. http://www.w3.org/TR/2016/WD-html-imports-20160225/.

[21] Cleotilde Gonzalez. Does Animation in User Interfaces Improve Decision Making? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 27–34. ACM, 1996.

[22] Mike Hazas, Christian Kray, Hans Gellersen, Henoc Agbota, Gerd Kortuem, and Albert Krohn. A Relative Positioning System for Co-located Mobile Devices. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, pages 177–190. ACM, 2005.

[23] Simo Hosio, Fahim Kawsar, Jukka Riekki, and Tatsuo Nakajima. DroPicks–A Tool for Collaborative Content Sharing Exploiting Everyday Artefacts. *Ubiquitous Computing Systems*, pages 258–265, 2007.

[24] Steven Houben and Nicolai Marquardt. Watchconnect: A toolkit for Prototyping Smartwatch-centric Cross-device Applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1247–1256. ACM, 2015.

[25] Steven Houben, Paolo Tell, and Jakob E. Bardram. ActivitySpace: Managing Device Ecologies in an Activity-Centric Configuration Space. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, pages 119–128. ACM, 2014.

[26] Scott E. Hudson and John T. Stasko. Animation Support in a User Interface Toolkit: Flexible, Robust, and Reusable Abstractions. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, pages 57–67. ACM, 1993.

[27] David Hyatt, David Baron, Chris Marrin, Dean Jackson, and Sylvain Galineau. CSS Animations. Technical report, W3C, may 2017. https://drafts.csswg.org/css-animations/.

[28] Rodolfo Inostroza, Cristian Rusu, Silvana Roncagliolo, and Virginica Rusu. Usability Heuristics for Touchscreen-based Mobile Devices. In

Proceedings of the 2013 Chilean Conference on Human-Computer Interaction, pages 24–29. Springer, 2013.

[29] Keith Instone. Usability Heuristics for the Web. *Web Review*, 1997.

[30] Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 234–241. ACM, 1997.

[31] Shahram Izadi, Harry Brignull, Tom Rodden, Yvonne Rogers, and Mia Underwood. Dynamo: A Public Interactive Surface Supporting the Cooperative Sharing and Exchange of Media. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, pages 159–168. ACM, 2003.

[32] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reacTable: Exploring the Synergy Between Live Music Performance and Tabletop Tangible Interfaces. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, pages 139–146. ACM, 2007.

[33] Martin Kaltenbrunner and Ross Bencina. reacTIVision: A Computer-vision Framework for Table-based Tangible Interaction. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, pages 69–74. ACM, 2007.

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

[35] Clemens Nylandsted Klokmose, Janus Bager Kristensen, Rolf Bagge, and Kim Halskov. BullsEye: High-Precision Fiducial Tracking for Table-based Tangible Interaction. In *Proceedings of the 9th ACM International Conference on Interactive Tabletops and Surfaces*, pages 269–278. ACM, 2014.

[36] Russell Kruger, Sheelagh Carpendale, Stacey D. Scott, and Saul Greenberg. Roles of Orientation in Tabletop Collaboration: Comprehension, Coordination and Communication. *Computer Supported Cooperative Work (CSCW)*, 13(5):501–537, 2004.

[37] Russell Kruger, Sheelagh Carpendale, Stacey D. Scott, and Anthony Tang. Fluid Integration of Rotation and Translation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 601–610. ACM, 2005.

[38] Ming Li and Leif Kobbelt. Dynamic Tiling Display: Building an Interactive Display Surface Using Multiple Mobile Devices. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, page 24. ACM, 2012.

[39] Ethan Marcotte. *Responsive Web Design*. Number 4. Editions Eyrolles, 2013.

[40] Nicolai Marquardt, Till Ballendat, Sebastian Boring, Saul Greenberg, and Ken Hinckley. Gradual Engagement: Facilitating Information Exchange Between Digital Devices As a Function of Proximity. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*, pages 31–40. ACM, 2012.

[41] Nicolai Marquardt, Ken Hinckley, and Saul Greenberg. Cross-device Interaction via Micro-mobility and F-formations. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, pages 13–22. ACM, 2012.

[42] Rolf Molich and Jakob Nielsen. Improving a Human-computer Dialogue. *Communications of the ACM*, 33(3):338–348, mar 1990.

[43] Meredith Ringel Morris, Anthony Cassanego, Andreas Paepcke, Terry Winograd, Anne Marie Piper, and Anqi Huang. Mediating Group Dynamics through Tabletop Interface Design. *IEEE Computer Graphics and Applications*, 26(5):65–73, September 2006.

[44] Mark W. Newman, Jana Z. Sedivy, Christine M. Neuwirth, W. Keith Edwards, Jason I. Hong, Shahram Izadi, Karen Marcelo, and Trevor F. Smith. Designing for Serendipity: Supporting End-user Configuration of Ubiquitous Computing Environments. In *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, pages 147–156. ACM, 2002.

[45] Jakob Nielsen. Coordinating User Interfaces for Consistency. *Academic Press*, 20(3):63–65, 1989.

[46] Jakob Nielsen. Enhancing the Explanatory Power of Usability Heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 152–158. ACM, 1994.

[47] Jakob Nielsen. *Usability Inspection Methods*. John Wiley & Sons, 1994.

[48] Jakob Nielsen and Rolf Molich. Heuristic Evaluation of User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 249–256. ACM, 1990.

[49] Don Norman. Emotion & Design: Attractive Things Work Better. *Interactions*, 9(4):36–42, jul 2002.

[50] Donald A. Norman. Affordance, Conventions, and Design. *Interactions*, 6(3):38–43, 1999.

[51] Matunda Nyanchama and Sylvia L. Osborn. Access Rights Administration in Role-Based Security Systems. In *Database Security VIII - Status and Prospects*, pages 37–56. North-Holland, 1994.

[52] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In Steve Howard, Judy Hammond, and Gitte Lindgaard, editors, *Proceedings of the International Conference on Human-Computer Interaction*, pages 362–369. Springer US, 1997.

[53] James Patten, Hiroshi Ishii, Jim Hines, and Gian Pangaro. Sensetable: A Wireless Object Tracking Platform for Tangible User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 253–260. ACM, 2001.

[54] 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*, pages 45–54. ACM, 2014.

[55] J. Rekimoto and M. Saitoh. A Spatially Continuous Workspace for Hybrid Computing Environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 377–385. ACM, 1999.

[56] Jun Rekimoto. Pick-and-drop: A Direct Manipulation Technique for Multiple Computer Environments. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pages 31–39. ACM, 1997.

[57] Peter Robinson and Philip Tuddenham. Distributed Tabletops: Supporting Remote and Mixed-Presence Tabletop Collaboration. In *Horizontal Interactive Human-Computer Systems*, pages 19–26. IEEE, 2007.

[58] Reinout Roels and Beat Signer. INFEX: A Framework for Cross-Device Information Exploration and Exchange. 2017.

[59] Iván Sánchez, Jukka Riekki, Jarkko Rousu, and Susanna Pirttikangas. Touch & Share: RFID based Ubiquitous File Containers. In *Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia*, pages 57–63. ACM, 2008.

[60] Mario Schreiner, Roman Rädle, Hans-Christian Jetter, and Harald Reiterer. Connichiwa: A Framework for Cross-Device Web Applications. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2163–2168. ACM, 2015.

[61] 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*, pages 294–303. ACM, 2004.

[62] Orit Shaer and Eva Hornecker. Tangible User Interfaces: Past, Present, and Future Directions. *Foundations and Trends in Human-Computer Interaction*, 3(1–2):1–137, 2010.

[63] Chia Shen, Katherine Everitt, and Kathleen Ryall. UbiTable: Impromptu Face-to-face Collaboration on Horizontal Interactive Surfaces. In *Proceedings of the International Conference on Ubiquitous Computing*, pages 281–288. Springer, 2003.

[64] John T. Stasko. Animation in User Interfaces: Principles and Techniques. In *User Interface Software*, pages 81–101. John Wiley & Sons, 1993.

[65] Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. *User Interface Design and Evaluation*. Morgan Kaufmann, 2005.

[66] Anthony Tang, Carman Neustaedter, and Saul Greenberg. Videoarms: Embodiments for Mixed-presence Groupware. In *Proceedings of HCI*, pages 85–102. Springer, 2007.

[67] Lucia Terrenghi, David Kirk, Hendrik Richter, Sebastian Krämer, Otmar Hilliges, and Andreas Butz. Physical Handles at the Interactive Surface: Exploring Tangibility and Its Benefits. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 138–145. ACM, 2008.

[68] Philip Tuddenham and Peter Robinson. Territorial Coordination and Workspace Awareness in Remote Tabletop Collaboration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2139–2148. ACM, 2009.

[69] Manas Tungare, Pardha S. Pyla, Pradyut Bafna, Vladimir Glina, Wenjie Zheng, Xiaoyan Yu, Umut Balli, and Steven Harrison. Embodied Data Objects: Tangible Interfaces to Information Appliances. In *Proceedings of the 44th Annual Southeast Regional Conference*, pages 359–364. ACM, 2006.

[70] Xin Wang and Frank Maurer. Tabletop AgilePlanner: A Tabletop-based Project Planning Tool for Agile Software Development Teams. In *Proceedings of 3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems*, pages 121–128. IEEE, 2008.

[71] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991.

[72] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, 1993.

[73] Pierre Wellner and Stephen Freeman. The DoubleDigitalDesk: Shared Editing of Paper Documents. *XEROX Euro PARC Technical Report EPC-93-108*, 1993.

[74] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. The Cognitive Walkthrough Method: A Practitioner's Guide. In Jakob Nielsen and Robert L. Mack, editors, *Usability Inspection Methods*, chapter Usability Inspection Methods, pages 105–140. John Wiley & Sons, Inc., 1994.

[75] Daniel Wigdor, Hao Jiang, Clifton Forlines, Michelle Borkin, and Chia Shen. WeSpace: the Design Development and Deployment of a Walk-up and Share Multi-surface Visual Collaboration System. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1237–1246. ACM, 2009.

[76] Andy Wilson and Raman Sarin. BlueTable: Connecting Wireless Mobile Devices on Interactive Surfaces Using Vision-Based Handshaking. In *Proceedings of Graphics Interface*, pages 119–125. ACM, May 2007.

[77] Uwe Zdun. Patterns of Tracing Software Structures and Dependencies. In *Proceedings of 8th European Conference on Pattern Languages of Programs*, pages 581–616, Irsee, Germany, June 2003.