Combining and Extending Concepts of Single Display Groupware and Tangible User Interfaces for Intuitive Collaboration

A dissertation submitted to the
ETH ZURICH

for the degree of
Doktor der Wissenschaften

presented by
CHRISTOPH SCHWAB-GANSER
Dipl. Arch ETH
born December 26, 1972
citizen of Basel - Switzerland

accepted on the recommendation of
Prof. Dr. K. Wegener, examiner
Prof. Dr. M. Fjeld, co-examiner
Prof. Dr. A. Kunz, co-examiner

2011
# Contents

1 Introduction .......................... 1
   1.1 Motivation ........................... 2
   1.2 Research Projects Context .......... 6
      1.2.1 togETHer ......................... 6
      1.2.2 CRION ............................ 7
   1.3 Outline of the Dissertation .......... 8
   1.4 Contribution .......................... 9

2 Related Work .......................... 11
   2.1 Interaction Concepts .................. 11
   2.2 Systems and Applications ............... 13
   2.3 Input Technology ........................ 15
   2.4 Conclusion Related Work ............... 18

3 Supporting Interactive Meetings ........ 19
   3.1 Setup of the Environment .............. 19
      3.1.1 Interacting with the Environment Elements .................. 22
   3.2 Interaction Principles .................. 23
      3.2.1 Required Basic Functionalities .................. 24
      3.2.2 Physical Interaction Constraints .................. 25
      3.2.3 Intuitive Interactions and Skill Acquisition .................. 26
      3.2.4 Multi-User and Multi-Device Interaction .................. 27
3.2.5 Direct Manipulation ........................................ 29
3.2.6 TUI ................................................... 31
3.2.7 TUI and Pen ........................................... 37
3.3 Proposed Interactions and Input Devices ....................... 39
  3.3.1 Pen ................................................... 39
  3.3.2 Eraser ................................................ 40
  3.3.3 Handler .............................................. 40
  3.3.4 Cutter ............................................... 42
  3.3.5 Ruler ............................................... 43
  3.3.6 Color Tool .......................................... 46
  3.3.7 Laser Pointer and Multi-Display Interactions ............. 47
  3.3.8 Folding Import Tool .................................. 47
3.4 Tracking Technology and Input Devices ......................... 50
  3.4.1 Pilot Studies ........................................ 50
  3.4.2 Requirements ....................................... 54
3.5 Remote Collaboration ........................................ 56
  3.5.1 Video Conferencing ................................... 56
3.6 Software .................................................. 57
  3.6.1 Input Event Handling .................................. 57
  3.6.2 Client Application: DOG .............................. 59
  3.6.3 Development Platform ................................. 61
4 Enabling Rich Multi-User Interaction .............................. 63
  4.1 Evaluation of Base Technologies ............................. 63
  4.2 Principle of Operation .................................... 65
    4.2.1 Bit Code ........................................... 66
    4.2.2 Synchronization ..................................... 68
    4.2.3 IDs and LED Topology ............................... 74
    4.2.4 Mixing Devices with Different Bit-Code Length ....... 76
4.3 Recognition Software .................................................. 77
  4.3.1 Obtaining the State of the Synchronization .................. 80
  4.3.2 Acquisition ...................................................... 80
  4.3.3 Segmentation .................................................... 80
  4.3.4 Combining Spots to a Bit Code ................................ 85
  4.3.5 Device-Specific Input Handling ............................... 92
  4.3.6 Calibration ...................................................... 93
4.4 Electronic Circuits ................................................... 93
  4.4.1 Synchronization Circuit ........................................ 94
  4.4.2 Interaction Device Circuits ................................... 96
4.5 Realized Interaction Devices ....................................... 97
  4.5.1 Stylus .......................................................... 98
  4.5.2 Laser Stylus .................................................... 99
  4.5.3 Eraser .......................................................... 99
  4.5.4 Color Can ....................................................... 99
  4.5.5 Ruler .......................................................... 100
  4.5.6 Handle .......................................................... 103
4.6 Cameras ............................................................... 104
4.7 Projection Screens .................................................... 107
4.8 Resulting Detection Rate ............................................. 109

5 Interfacing the Input Devices ........................................ 113
  5.1 Auxiliaries .......................................................... 113
    5.1.1 Event Address and Filter .................................... 114
    5.1.2 Pointer ....................................................... 115
    5.1.3 Event ........................................................ 116
  5.2 Configuration ...................................................... 117
  5.3 Drivers and Devices ............................................... 118
  5.4 EventDispatcher .................................................. 119
7.0.2 Analysis of the experiments ........................................... 175

8 Conclusion and Outlook .................................................. 181
  8.1 Outlook ........................................................................... 182
    8.1.1 Embedded System ..................................................... 183
    8.1.2 Using the Amplitude of the Tracking Signal .................... 183
    8.1.3 Integrating Passive Optical Markers ............................ 186
    8.1.4 Experiments with the Embodiment of Devices .............. 187

A Evaluation of Existing Programming Interfaces .................... 189
  A.1 WIN32 ......................................................................... 189
  A.2 Direct Input ..................................................................... 190
  A.3 OS Mouse Driver and Mouse Filter .................................. 190
  A.4 Raw Input ....................................................................... 190
  A.5 TabletPC SDK and Ink API ............................................. 191
  A.6 Wintab API ..................................................................... 192
  A.7 Avalon API ...................................................................... 192
  A.8 Conclusion of Evaluation ................................................ 193

B Curriculum Vitae ................................................................ 195
  B.1 Scientific Publications ................................................... 196
Abstract

Companies are concentrating on their core business while concurrently interacting with a larger market. This requires more collaboration between globally distributed business partners and between specialists from different fields. The computer-supported environment presented in this dissertation addresses this need. It supports real-time collaboration of collocated and distributed work groups.

Users in work groups are deeply immersed in communication, so the system should impose as little cognitive load as possible. This requirement is addressed by leveraging skills the user has already acquired. Paper-based work methods are still widely used in engineering for planning and brainstorming. Thus, the interactions and tools used in these setups act as references for developing the interactions for the new environment. Conventional graphical user interfaces (GUI) allow the interaction of a single user using a keyboard and a pointer. In contrast, paper-based setups do not have such restrictions: users can interact simultaneously and use specialized tools for different tasks. These features of paper-based setups are present in the computer-supported environment. Single Display Groupware (SDG) allows multiple users to simultaneously interact on a shared display, while Tangible User Interfaces (TUI) are used to explore interactions based on input devices other than the traditional mouse and keyboard. A selection of TUIs imitate tools known from paper-based setups, allowing interactions similar to what the user already knows. Similar to the paper-based tools, the TUI devices keep their properties when they are used on different displays in the same conference room.

To enable interactions using a pen and a TUI, a novel input technology has been developed. A bit code is sent by an infrared light emitting diode to a camera behind the back-projection display. The position, orientation, and the state of the input devices can be determined by processing the camera's image frames. The device's state consists of its keys and other input capabilities. Using a 200-Hz camera and up to eight input devices, the input technology allows detection rates of up to 66 Hz.

A client software for supporting the work groups has been developed which allows
sketching on cards and on imported documents (PDF, JPG, 3DS). Among the supported interactions are: sketching, erasing sketches, measuring drawings, changing the color of a pen, manipulating digital Post-Its, and changing the view on the drawings.

The environment was tested in a user study which compared the setup to a paper-based setup and a commercial touch-sensitive setup. The results of the study questionnaire confirmed the chosen approach. The developed system was built to imitate the interactions of the paper-based system. In the user study, the interactions of the developed system were rated as being almost on par with those of the paper-based system, which had the highest ratings for most aspects. The touch-sensitive system was mostly rated last. With regard to some aspects such as changing color, measuring, and collaboration, the developed system was rated higher than a paper-based setup.
Zusammenfassung


Das Konzept "Single Display Groupware" erlaubt mehreren Personen die Interaktion auf einem Bildschirm, wobei mit "Tangible User Interfaces" (TUI) vielfältigere Interaktionsmodelle als mit Tastatur und Maus zu Grunde liegen. Hilfsmittel, welche bei papierbasierter Zusammenarbeit verwendet werden (Stifte, Büroklammern, Post-its etc.), werden als TUI nachgebildet und erlauben eine ähnliche Interaktion im computergestützten System. Dabei behalten die Hilfsmittel ihre Eigenschaften, auch wenn sie auf mehreren Bildschirmen im Sitzungsraum zur Anwendung kommen.

Es wird eine Interaktionstechnologie entwickelt, welche Codes von infraroten Leuchtdioden in den Eingabegeräten an eine Kamera hinter dem Rückprojektions-Schirm sen-
det. So kann das System Position, Orientierung, Identität und Zustand der Eingabegeräte erkennen. Der Status eines Gerätes setzt sich aus dem Zustand seiner Tasten und anderer Eingabemöglichkeiten wie Schiebern zusammen. Wenn gleichzeitig bis zu acht Geräte und eine 200 Hz-Kamera verwendet werden, erreicht die Eingabeteknologie eine Erkennungsraten von bis zu 66 Hz.

Es wird eine Software entwickelt, die sowohl für die verteilte wie die lokale Zusammenarbeit verwendet werden kann. Sie erlaubt das Skizzieren auf dem Bildschirmhintergrund, auf digitalen Post-its und importierten Dokumenten (PDF, JPG, 3DS etc.). Die Anwender können u. a. Skizzen erstellen und löschen, Pläne vermessen, die Stiftfarbe ändern, Post-its verschieben und die Ansicht auf die Dokumente verändern. Die von den Anwendern vorgenommenen Änderungen werden kontinuierlich zwischen den an der Zusammenarbeit beteiligten Standorten abgeglichen.

Acknowledgements

This dissertation would not have been possible without the support from many people and organizations. This work was carried out within the togETHer and CRION projects. I would like to thank ETH World and KTI for funding those projects. Next, I would like to thank the following people for supervising my dissertation:

- Konrad Wegener for giving me the opportunity to conduct my research at the inspire AG.
- Morten Fjeld for pointing me to related research and for his help in setting up the user study.
- Special thanks goes to Andreas Kunz. He provided an environment which allowed me to conduct my research independently, but he always offered support when needed. His help was also invaluable in clarifying my thoughts when working out the written dissertation.

I would also like to thank my work colleagues at the time and especially those who contributed to the presented work:

- Adrian Steinemann for helping me with many aspects of the user study, the setup, and the electronics.
- Ramon Hofer for his talent in building interaction devices that actually work.
- Thomas Kennel for designing and building the housing of the digital pen and the table used as a horizontal workspace.

Many students contributed to the success of the presented work. First of all, I want to thank Christoph Hürzeler for his great support in implementing the software. Also many thanks to the following students for their contributions: Christoph Stifter for evaluating
the screens, Samira Andersson for developing interaction devices, and Sem Hediger for building the A/V conferencing solution.

Further, the ETH workshops’ employees Freddy Bürcki, Kurt Fügli, and Albert Weber were of great help in building the interaction tools.

I would also like to thank Sarah McGee for proofreading the dissertation. I really benefitted a lot from working with her. She not only made corrections, but also explained why her corrections are necessary.

Also many thanks to my family: without their support, I would not have been able to complete the dissertation after starting to work in a full-time position. Most importantly, I would like to thank my wife, Kathrine, for taking over many of my family duties. This allowed me to free up some time to work on the dissertation. I would also like to thank my mother for occasionally babysitting the kids. Further, I apologize to my daughter, Samira, and my son, Julian, for the time we could not spend together. I hope I can make up for it in the near future.
Chapter 1

Introduction

The integration of an increasing number of functions into a limited space results in the development of mechatronic products. This trend demands tight collaboration between different engineering disciplines.

At the same time, companies are concentrating on their core business. This discrepancy between the need to integrate an increasing number of core competencies into a product and the companies’ specialization in a small area of expertise can be solved through an increasing level of collaboration between the companies and between development units within them. This is part of the parallel work process also known as Simultaneous Engineering [61].

Figure 1.1: Simultaneous engineering schema with arrows to denote possible collaborations

Figure 1.1 illustrates the simultaneous engineering process. Arrows show potential
interfacing between different engineering teams and denote possible application areas for computer-mediated collaboration systems.

The different phases in the product development process require employees with different levels and areas of expertise. Examples include patent attorneys, financial experts, marketing experts, design engineers, and sales and service employees. Communication and collaboration between these teams with different backgrounds is further complicated through the different jargon used in their domains.

Large companies’ sites can be scattered all over the world. Therefore, employees have to work with colleagues worldwide. Despite the physical distance, they have to interact as members of the same virtual team or as members of collaborating teams. This distribution of work further hinders collaboration for many reasons. Soft factors such as language and cultural differences play an evident role. Different time zones and physical distance between partners are geographical obstacles. Traveling to meet partners in face-to-face meetings consumes a considerable amount of a company’s resources with regard to time and money. This has motivated the development of a multitude of products for remote collaboration.

## 1.1 Motivation

A common way to distinguish between the different ways of collaboration is by analyzing the dimensions of time and space. Distinguishing whether the dimensional conditions are the “same” or “different” creates four categories (Table 1.1).

For example, a collaboration may take place at the same place but at a different time. These categories can further be applied to the information technology used in mediated collaboration [99].

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Same Time (Synchronous)</th>
<th>Different Time (Asynchronous)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same Location</td>
<td>Face-to-Face Meeting</td>
<td>Bulletin Board</td>
</tr>
<tr>
<td></td>
<td>Electronic Whiteboard</td>
<td></td>
</tr>
<tr>
<td>Different Location</td>
<td>Telephone</td>
<td>Mail</td>
</tr>
<tr>
<td></td>
<td>Video Conferencing</td>
<td>E-mail</td>
</tr>
</tbody>
</table>
1.1. Motivation

Software to support asynchronous teamwork is deeply integrated in the routine of today’s office workers. We communicate by e-mail, use document servers to share data, and use workflow systems to support our business processes. These systems are widely used in industry and make perfect sense for coordinating tasks within a team. An example would be a designing engineer requesting a manager to approve a modification in a design. Also, synchronous remote collaboration tools such as video conferencing (VC), application sharing, and instant messaging have recently gained much more acceptance among users. However, tools to support co-located, computer-mediated groupwork such as electronic whiteboards are still rare [16].

Face-to-face meetings and slide shows are perceived as the typical way to perform teamwork. Undoubtedly, electronic presentations by domain experts adequately support decision makers in their work. But the importance of work groups has not been studied as extensively [118], [46], [123], [62]. Work groups consist of a small number of people with different levels and areas of expertise, and they constitute an important category in office and knowledge work (Figure 1.2).

Discussions with companies engaged in the automotive industry, mechatronic systems development, and in other mechanical industries have shown that paper-based, co-located work practices used in these groups go far beyond simple slide shows. Existing documents are printed out and annotated using various kinds of pens. Tracing paper is used to compare different design solutions. Creativity and brainstorming methods are used to generate new ideas for a specific development task (Figure 1.3). Small sketches representing solutions to certain problems can be clustered according to categories to get an overview of what has been done. This paper-based work practice allows the users to handle many kinds of physical tools. Paper or cardboard can have different colors and thicknesses. Design drawings can be annotated and measured with rulers. Physical artifacts can be used in a discussion and ideas can be quickly externalized by sketching them on a note attached to a related document.

Typically, a lot of information has to be available and visible in the teamwork process. Thus, co-located teams often have a dedicated project room in which all documents are arranged and the current state of the project can be easily visualized. Project rooms allow members to cluster information, to rearrange it, or to move it around (Figure 1.4). Walking around in the project room, having different viewpoints (physical and mental), or perceiving different information from any direction has a stimulating effect on the work in such a team. Thus, the paper-based work practice gives the user a rich experience which addresses many senses.

However, the paper-based work method also has its shortcomings. The results of such
teamwork are only available in an analog manner, i.e., on flowcharts, whiteboards, etc. All results have to be digitized to distribute them to the team members before and/or after the meeting. Taking pictures of the physical artifacts and mailing them to the participants is widely practiced but cumbersome. Based on these pictures, users will make modifications to designs in domain-specific applications. To keep the project room in sync with the modified digital data, plans have to be printed out and displayed.

Paper-based work methods can hardly be applied for remote collaboration. Typically, a document camera hooked up to a VC system is used to transmit sketches from one collaborating site to another. Nevertheless, since only one site can modify the document, it does not suit the interactive nature of group work.

Using a computer-supported environment instead of an analog work method could remove these drawbacks by working on a digital basis from the very beginning (“born digital”). This allows the integration into the information infrastructure of the company and eases the information flow.
1.1. Motivation

When using computer-mediated systems for collaboration, users are often overloaded by the need to operate an IT system while also being engaged in the meeting. Therefore, the user interface for supporting group work must be simple and intuitive and not impose a high cognitive load to the user.

My motivation was to create a very intuitive computer-supported environment for synchronous, co-located group work to support planning and sketching tasks in the engineering field. Examples for such tasks include brainstorming, design review, and Failure Mode and Effects Analysis (FMEA). An important aspect was to minimize the
intrusiveness of the system into the users’ collaboration and to ease interaction. Therefore, I facilitated pen-based and tangible interactions on a tabletop setup and across co-located vertical displays.

1.2 Research Projects Context

This presented work is based on research from two projects. Both projects dealt with remote collaboration and provided the groundwork for the dissertation.

1.2.1 togETHer

The togETHer project [13] was an internal Eidgenössische Technische Hochschule Zürich (ETH) project and focused on synchronous remote collaboration. Besides the
1.2. Research Projects Context

Innovation Center Virtual Reality (ICVR), Prof. M. Norrie (Department of Computer Science), Prof. M. Angélil (Department of Architecture), and the center “Network for Educational Technologies” (NET) contributed to the project.

The project did not solely focus on technological improvements, but also evaluated the needs of ETH members. Therefore, interviews and two web-based questionnaires were conducted. They indicated that members of ETH were often unaware of systems to support remote collaboration. Only an estimated four to six percent of the ETH employees uses videoconferencing systems (VC) [58]. Nevertheless, the questionnaires indicated that the usage of VC is increasing. Any reluctance can be explained by lack of need and the fear of losing too much time when beginning to use VC. Within the project, various systems for real-time conferencing were evaluated and an extensive market survey was conducted. As a result of the survey and the users’ needs, requirements for a collaborative editor (GEdit) were derived. GEdit features import/export filters in a standard file format (SVG), platform independence, and easy installation using Java Webstart. As many members of ETH had very little or no knowledge of current remote collaboration tools, interactive video tutorials were created. The tutorials provided the user with information on using and organizing video conferences. Terms commonly found in video conferencing software were explained. Finally, the interactive tutorials explained video conferences’ basic operation principles.

An “Augmented Reality” application for collaborative 3D viewing was developed for the project (Figure 1.5). The application was tested with architecture students. The results of this user study motivated the author to support group work with user interfaces that are less intrusive and that require a much shorter training period.

1.2.2 CRION

The presented work was also carried out within the CRION project [76], contributing in many areas to the success of the project.

Innovative solution finding is often associated with group work. Creativity methods can influence the quality and quantity of a group’s work outcome. Within the CRION project, an environment was developed for supporting creative group processes using information technology and work methods. By supporting joint interaction in a system and allowing group work with spatially distributed partners, CRION went beyond current work practices.
1.3 Outline of the Dissertation

In this section, an outline of the remaining chapters is provided followed by a discussion of the dissertation’s research contributions.

An extensive overview of related work follows in Chapter 2. The first section presents the approaches undertaken by researchers to support synchronous, co-located group work within various systems. The next section highlights contributions to the development of input technology.

In Chapter 3, a usage scenario of the proposed system is described. This scenario was refined by feedback from industry. Based on the scenario, the requirements guiding the development of the presented work are derived. The chapter then concludes with an overview of the concept chosen to meet the requirements.

Chapter 4 describes the developed input technology. It starts with the presentation of a pilot study conducted to gain insight into the problems encountered when supporting multi-user interaction. Next, the hardware and software for the input technology are presented. Besides the principles of operation, further optimizations and measurements are presented.

Chapter 5 describes application programming interfaces for receiving events from input devices. It presents the solution implemented in this study. In addition, Appendix A compares industry standards and proposals from research for handling input from input interfaces.
The client application providing the user interface is described in Chapter 6. Besides providing an overview of the data structures created, it also describes how the graphical user interface was realized, how the rendering subsystem was implemented, and what concept was used to distribute the data model across multiple hosts.

The completed collaborative environment was evaluated in a user study. The design and the results of a conducted user study are presented in Chapter 7.

Chapter 8 concludes the dissertation and gives an outlook on future research and improvements.

1.4 Contribution

This section concludes Chapter 1. It summarizes the key contributions of this dissertation for later reference.

This dissertation presents a novel computer-mediated environment supporting synchronous group work of co-located and distributed teams. The main contributions of this work are:

- **Evaluation of teamwork methods and support systems.** To propose a system for supporting co-located and remote group work, existing methods in the field of engineering were evaluated. Besides current work practices in industry, various systems proposed by research projects were studied. The evaluated information was used to construct a concrete usage scenario, which allowed concrete development requirements to be deduced.

- **Development of new interaction concepts for supporting group work.** Based on the scenario and the needs of the evaluated industries, a computer-mediated environment was realized. For this setup, new interaction concepts for supporting group work were designed by combining the concepts of Single Display Groupware (SDG) [120] and Tangible User Interfaces (TUI) [51]. Chapter 2 explains the concept of TUI and SDG in detail. The result of combining the two concepts allows multiple users to interact simultaneously on a shared back-projection display with multiple styli and Tangible Input Devices. This combination allows novel interactions through the notion of shared TUIs and styli personalized to the user. Furthermore, novel tangible user interfaces were developed, and existing ones were improved to support group work in the engineering field. Novel multi-computer
interactions were also developed to allow seamless interactions across multiple co-located displays.

- **Development of a novel tracking technology.** A novel tracking technology was developed to combine SDG and TUI interaction concepts. It uses infrared (IR) light to simultaneously detect multiple input devices on an interactive surface. Besides locating devices, the technology allows for device identification and the detection of the state of the devices’ input capabilities (e.g., buttons). The underlying principle is very flexible and allows a developer to adopt the technology for a specific application.

- **Development of a software for co-located and remote collaboration.** To implement the new interactions, a distributed software infrastructure (DOG: Distributed Object-oriented Groupware) was developed. It allows multiple co-located and remote computer-systems to join a collaboration session. DOG was developed to prototype various applications of groupware and to allow follow-up projects to continue the development by adapting the system to their needs. The software is based on portable libraries and has no direct dependency to a specific platform. This allows the code base to be ported from Linux to other platforms like Windows or Mac OS X with little effort.

- **Developing, conducting, and evaluating a user study.** A user study was developed to compare different setups in co-located group work. A design review task had to be solved by subjects in all setups. Qualitative feedback from the subjects was made possible by extending an existing questionnaire. A software module processing the events from the input devices allowed the spatial and temporal user activities to be recorded and analyzed. The developed user test was administered to 24 subjects in three setups: a commercial tabletop system, the developed system, and a paper-based setup. The video taping of subjects provided insight on usage patterns. As expected, users preferred paper-based work practices. From the computer-mediated environments, the presented system was preferred for most interactions. For some aspects, it was even rated above the paper-based setup.
Chapter 2

Related Work

Supporting the collaboration of a team with information technology belongs to the research field of Computer Supported Cooperative Work (CSCW). As described in Chapter 1, the intention of this work is to create a setup allowing collaboration without imposing a large cognitive load on users caused by the users handling complex interfaces. Intuitive interaction should be used to let the user concentrate on the collaboration and the given task. This is more closely related to the research field of Human Computer Interaction (HCI).

CSCW and HCI are interdisciplinary research fields. In both areas, people from various fields like computer science, work psychology, interaction design, sociology, and business administration conduct their research with a different emphasis.

The presented dissertation focuses on the problem areas related to computer science and interaction design. The sources presented here have a similar approach. Nevertheless, the issues encountered in group work go far beyond mere technical or artistic aspects (see e.g., [66]).

The related work is divided into three sections. First, human-computer interaction concepts are presented. Second, systems for co-located group work are shown. Third, input technology research and technologies are presented.

2.1 Interaction Concepts

The Graphical User Interfaces (GUI) shipped with today’s operating systems have an underlying single-user concept. Following the PC (personal computer) metaphor, the operating systems only provide single-user interaction with a two-dimensional mouse
Chapter 2. Related Work

If a team wants to interact with a conventional GUI in such a system, only one user can access the system at a time. Passive users often point with their fingers and give orders to the user with the input device. Users also compete for the input control. Furthermore, group work is hindered as people disengage and become frustrated [121]. Having a private input device is more convenient for the users as they simply do not have to share [120]. Single Display Groupware (SDG) proposes the use of one shared output channel and a private input channel for each user (see Figure 2.1). The SDG term was introduced by Stewart et al. [121], while many technical aspects were addressed by Bier et al. [9]. This approach allows every user to interact with the data model at the same time. As only one display is used, gestures and mimics can be used for communication.

![Figure 2.1: PC Single-User concept (a) and Single Display Groupware concept (b); source: Stewart.](image)

The Tangible User Interface (TUI) concept [51] arose from the criticism of mouse and keyboard interaction. Human senses are able to handle complex interaction systems such as driving a car [11], where bi-manual and bi-pedal interaction is needed. Our ability to manipulate physical objects is poorly exploited through the usage of the keyboard and mouse. TUIs emphasize many different aspects of HCI: bi-manual interaction [51], parallel input, physical handles to access digital content [133], the direct manipulation of virtual objects with physical counterparts [52], and specialized input devices shaped like icons to depict their function and to allow an intuitive interaction [131]. TUIs often support parallel input and support multiple users. Although styli and mice are literally “tangible” user interfaces, in [51] they are perceived as bound to the GUIs’ generalized pointer concept.

“Tabletop” is another category of interactive system. Whereas normal GUIs have been
developed for computers with vertical screens, Tabletop interfaces use a horizontal interaction space. Unlike vertical displays, the arrangement of the screen does not favor the users standing on a particular side of the screen. This lack of a preferred direction renders existing GUIs inappropriate. In [50], a rotating user interface is presented for single-user applications, whereas Kruger et al. [79] presented an alternative interaction and evaluated it in a collaborative setup.

The collaborative use of a computer system requires large displays. They allow all participants to see the system's output and to directly generate input on the display. Originally, GUIs were developed for small screens. This has motivated researchers to improve interaction on large screens. A big problem is the limited reach of the user [46]. A child may not be able to reach the application’s menu bar on a vertical display, whereas a tall person may find it uncomfortable to draw on the lower part.

2.2 Systems and Applications

Most commercial real-time groupware is designed to be used for remote collaboration; each user interacts with an individual PC. This model does not conflict with the single-user-per-terminal concept of current operating systems. Examples of these joint editing software packages include Corel Grafigo (Sketchbook), UGS Teamcenter Visualization (Digital Mockup), and SubEthaEdit (text editing).

The CoLab research project [118], an early work in colocated group work, emphasized the importance of simultaneous input in group work. Up to six users collaborated through their networked workstations and controlled a large, touch-sensitive display. All users shared the same view of the application through a shared GUI-system. But unlike as with application-sharing systems, the users could simultaneously interact with the shared data model. The brainstorming application “Cognoter” allowed topics to be entered, clustered, and related to each other by drawing arrows; and “Boardnoter” allowed simple sketching. For private data like e-mails, the system allowed unshared windows. The authors came up with the principle “What You See Is What I See” (WYSIWIS). Not to be confused with “What You See Is What You Get” (WYSIWYG). Besides WYSIWIS, CoLab uses joint editing applications on colocated, networked computers. A user study highlighting the difficulties encountered in these kinds of setups was published in [38].

The DigitalDesk [94] is aimed at combining the well established interaction metaphor of the digital desktop with a physical desk. The system supports seamless interactions between the physical and the virtual world, like copying numbers printed on a piece of
Chapter 2. Related Work

paper to a calculator program projected on the digital augmented desk.

Bier et al. presented Multi-Device Multi-User Multi-Editor (MMM) [9] [8], a user interface and software architecture allowing a group of users to simultaneously interact with computer mice on a shared display. Steward refined the SDG model by conducting two evaluations in primary schools using the drawing application KidPad [42].

An early contribution to the interaction on large screens is “Tivoli” [103], a whiteboard software to support small work groups. The software was an early experiment in the design of a pen-only user interface. The software was also able to network whiteboards and was an early attempt to enable simultaneous shoulder-to-shoulder collaboration (similar to SDG). Meanwhile, much of the described functionality can be found in today’s commercial products. Flatland [90] separates the users’ actions into groups. Besides moving, scaling, and merging, application-specific operations can be performed. Rekimoto proposed Pick-and-Drop [106] [107] [109] [108], a multi-device interaction technique. Users can simultaneously work on their handheld pen-based devices and can move the created content to an electronic whiteboard using a simple pen movement. Guimbretiere et al. [62] were motivated by work practices in design studios. They concentrated more on single-user interactions on large vertical displays. The presented interactions were based on a digital pen and content was placed on virtual cards. The ZoomScape interaction helps manage the screen space and a novel marking menu [63] simplifies the command selection.

Motivated by the paper-based work practices of web-designers, Klemmer et al. [78] presented a tangible system for manipulating physical Post-Its and their relationship. In [131], a TUI is presented for interacting with a geographical visualization system. The system uses various specialized interaction devices, designed as graspmable replacements for conventional GUI elements. Phicons are the graspable version of GUI icons. Physical handles to manipulate the UI and lenses are the physical counterpart of the GUI window. Caretta [124] is a system for face-to-face collaboration. The users can collaboratively plan a city on a table by using a tangible interface. Additionally, they can interact with the system using their networked PDAs. This allows users to create their own version of the city plan, while other users are engaged in discussion.

In [110], a military project is presented to support command, control, communication, and intelligence (C3I) tasks in situation rooms. It features a large, tiled back-projection display. It has a high resolution and is integrated into the room’s walls. Body gesture and voice recognition were evaluated to provide untethered interaction. The military personnel working in the setup were therefore not hindered by cable-bound interaction devices. A laser pointer pen was proposed for up-close interaction at the display, when
the user is hard to detect by gesture recognition. “I-Land”, an environment for creativity and innovation is presented in [123]. The system consists of various room elements like walls, chairs, and tables with integrated information technology called “roomware”. It presented multi-computer interactions for transporting information between the different elements. The mechanism known as a “passage” mechanism allows data to be associated with physical objects. The data can then be moved to other screen elements using the physical object. Weight is used to identify the physical objects. Another interaction allows the user to slide content from one screen to another with a sweeping gesture.

2.3 Input Technology

Commercial single-user systems for interactive surfaces can be categorized by the technology they use. The Canon LV-DP 10 and 11 [14] allow one person to interact with the system during a presentation. They feature states for hovering, left and right clicks, and detecting the position of one single input device. The detection of the states is based on a modulated infrared signal. DViT [116] consists of a frame that can be applied to any back-projection or plasma display. The frame stands out approximately 4 cm from the projection surface and contains four cameras in the corners to observe the drawing surface. In addition, the frame contains an integrated IR illumination, simplifying the detection of a finger or a pen. The system detects one state for drawing.

Older SMARTboards [116], DynaWall and InteracTable [123] use a touch-sensitive foil to locate the touch of a user. The vertically mounted DynaWall consists of three plasma screens, each superimposed with such a touch-sensitive foil allowing direct interaction by the user. InteracTable is a horizontally mounted plasma display with a touch sensitive foil on top.

Ultrasonic tracking systems use the time of flight principle to detect the position of input devices. Additionally, an infrared signal is used for synchronization and pen identification. The peak frequency is at 8 kHz with a noticeable acoustic noise. eBeam 1 [43] uses two receivers, allowing left and right-handed users to interact with the system. eBeam 3 [43] and Mimio Xi [88] have only one receiver, making them more sensitive to shadowing. Both systems have only one state for drawing. The pen is only tracked if it is pressed on the whiteboard.

WACOM Intuos II tracks a stylus and a puck simultaneously on the tablet. The discontinued “WACOM Meeting Staff” could identify which of its three styli was used.
LiveBoard [46] consists of an LCD projector and active styli with four distinct states. It uses a rear mounted photo-effect diode that detects the location of a stylus fitted with an LED. The LED intensity is modulated, allowing the system to use frequency multiplexing to identify a few different styli with each stylus having its own frequency.

Several projects have worked with laser pointers to allow interaction on large displays at a distance. The use of laser pointers requires special interaction techniques as the device does not feature any buttons or states [98]. Lingering with the laser pointer can be used to signal a left-button event. Cavens et al. [17] added buttons from a wireless mouse. Davis and Chen [37] used laser pointers to allow multiple users to interact on a high-resolution tiled display. Cheng and Pulo [18] used the commercial Smart-Nav system to allow users to interact by pointing infrared lasers on front-projection screens. In [97] and [1], a system is described that enables up to three users to simultaneously interact with laser pointers on a front projection display. In order to identify the devices, they are turned off and on again every third frame. LaserWall [100] is a low-cost laser range scanner that tracks a user’s hand in front of an interactive surface. States are available since the system recognizes hand gestures with fingers.

DiamondTouch (DT) [39] is an outstanding multi-user, multi-touch input technology for SDG. It allows users to interact with their bodies (fingers, hands, and arms) on a horizontal front projection display. The detection of the user’s finger is realized by capacitive sensing. Each antenna integrated into the interaction surface emits an electromagnetic signal at a unique frequency. After a signal has passed through the user’s body, it is picked up and analyzed by a receiver integrated into the user’s chair. DT can assign every touch to a user. In order to allow unambiguous bi-manual interaction, DT uses a cable-bound pen connected to a separate receiver.

Multi-touch became further popular through internet videos by Han [64], through the Apple iPhone [23], and through Microsoft Surface [30]. Nevertheless, the underlying idea was first mentioned in 1981 in [142]. The initial Microsoft Surface system uses a back projection display with a IR emitter and cameras behind the screen. The IR reflection of objects and fingers on the surface is used for tracking. In [72], Microsoft Research integrated a similar system into an LCD display. A grid of IR sensors and emitters are mounted behind the LCD’s back-light. The most recent Microsoft Surface system (version 2.0 [31]) features a 40-inch LCD display from Samsung that integrates IR sensors into the LCD matrix. A similar display is also available from Sharp [32]. Han’s and Microsoft’s systems can detect multiple touches, but unlike DT, they cannot identify which touch belongs to which user. Therefore, the application for collaboration is limited. For example, no color could be assigned to the touches of a particular user’s
2.3. Input Technology

finger. In [40], a camera above an interactive surface is used to correlate the touches to a hand of a particular user. Nevertheless, the system assumes that the position of the user does not change. Further, the system is susceptible to occlusion since the user is likely to bend over the horizontal display. Both Microsoft Surface systems also support tracking the position and orientation of objects tagged with IR-reflective tags. The systems do not support device states, which would allow them to support buttons and other input capabilities on the tracked objects. PlayAnywhere [141], a multi-touch system that emphasizes ad-hoc setups, is also able to register the distance of a finger above the interaction surface.

The interactive Tabletop system Sensetable [101] consists of two modified WACOM tablets. The system uses pucks as input devices. Modifiers like dials can be plugged onto the pucks. The update rate of the system is less than 1Hz. If only one device is touched, it can be tracked at a higher rate.

Six-DOF tracking systems offer additional degrees for the pitch, yaw, and elevation of multiple tracked devices. Besides having many advantages, they tend to be very expensive. As the envisioned setup is thought to be crowded with many users and objects, ultrasonic tracking is not expected to work properly because of shadowing. Electromagnetic tracking devices are cable bound and vulnerable to distortion through the presence of metallic objects. Optical tracking systems require a minimum distance from the acquisition unit to the tracked objects, limiting the possible applications in tabletop setups. Optical systems often use IR light emitted by LEDs or reflected on retroreflective spheres for tracking. As the system cannot distinguish these markers, multiple tracking systems must be mounted on tracked objects with a distinct geometry. This restricts the construction of the input devices and makes the system sensitive to occlusions. The state of the buttons must be transmitted using an alternative communication channel like RF transceivers. In [89], a High-Level Data Link Control (HDLC) frame is transmitted from a handheld device to a desktop computer using IR light. The author used no synchronization between the devices. The transmission required up to five seconds. PhaseSpace’s tracking system [70,86] uses active markers emitting a modulated signal containing an ID. Synchronization is done using an RF signal.

Commercial radio frequency identification (RF-ID) systems have been used for TUIs. Examples include the SenseBoard [73] and Caretta [124]. RF-ID systems allow the presence of a tagged device above the reader to be detected. The resolution is normally limited to a wide-meshed grid. Zowie Power is an RF tracking technology developed for a new generation of electronic toys. In the whitepaper [115], the company claimed the system was able to detect four dimensions (2D position, elevation, and yaw). Toys
called Playsets were sold around the year 1999 using Zowie Power for detecting pup-
pets on the game board, triggering a story telling software on the attached computer. In 2000, LEGO acquired the company.

Optical tracking systems for TUIs either use markers to detect the position, or they try to recognize the objects through their distinct shape features [77, 83, 117, 131]. When using markers, the system can either identify input devices by a code printed on the marker [75, 108] or by the relation between similar markers [33, 135]. Klemmer et al. [78] use a back-projection SMARTboard for their setup. Besides the single-user touch screen, it uses two additional cameras: one tracking the Post-Its from the backside of the SMART board and another acquiring the physical sketches of the Post-Its. Unlike input technologies for SDG, TUIs often do not use a high update rate.

2.4 Conclusion Related Work

Many TUI systems have featured very natural and intuitive interactions with complex computing systems. In comparison, the stylus is an old tool in human history and has been used for thousands of years for writing and sketching. Many people have learned how to use a stylus, even before primary school.

Concerning the interaction capabilities of GUI systems, SDG showed a viable way of extending existing GUI systems for collaborative use. Whereas SDG requires every user to have a pointer, the interaction design of TUIs often does not address the aspects of the collaborative use as fully as SDG. The input modality is often specialized for a specific task.

SDG and TUIs depend on the features of the underlying input technology: the number of devices tracked simultaneously, sampling rate, number of dimensions supported, identification of input devices, and number of device states. Current input systems do not provide the combination of multiple styli and TUI in a collaborative setup. Either they are built for TUIs and lack support for styli or vice-versa.
Chapter 3

Supporting Interactive Meetings

The motivation for this dissertation is the potential that computer-mediated systems offer in supporting collocated and distributed group work (see Chapter 1). The main focus is supporting groups working in collocated and net-based team sessions. This dissertation should result in a computer-mediated environment which demonstrates how work groups can be supported in a completely digital environment and better integrated into the company’s work flow. Of course, the environment should also offer improved usability compared to current solutions which are based on conventional graphical user interfaces (GUI).

In the following, the requirements of the envisioned environment are described. First, the physical environment and its use are described, followed by the desired interactions. Next, the requirements for the input technology supporting the described interactions are defined. The chapter concludes with a set of requirements for video conferencing systems and accompanying software.

3.1 Setup of the Environment

The basic elements of the environment are large displays. These elements should be usable stand-alone or in conjunction with other elements. Depending on the users’ needs, one or more of the elements can be used to facilitate group work.

There are two types of elements: a horizontal (table-like) and a vertical, similar to current electronic whiteboards (see Figure 3.1). Depending on the particular application, one of the two or a combination can be used.

The horizontal large-display element is suitable for group work in which every team
Chapter 3. Supporting Interactive Meetings

Figure 3.1: Horizontal (left) and vertical (right) large-display workspace.

member participates in the same way. Everybody has the same access to the display for interaction. The work group gathers around the shared table and works on the shared display with a common goal.

The vertical large-display element structures the surrounding space so that only user standing in front of the element can see and access the display. It might be used as a shared workspace for two persons working simultaneously, but it is better suited for presentations. If used in conjunction with a horizontal element, the vertical elements can be used as additional display area. This display area can be used to show related project data, which can also be annotated if required. Furthermore, the vertical elements can be used if a group splits up into smaller groups for individual tasks (see Figure 3.2).

The physical requirements for the horizontal display are as follows:

- The display size and position must allow an average user to access all areas of the screen. As the system is designed for multiple-screen usage, the user is expected to roam between the displays. Therefore, the length of the table may be varied to allow more users. Nevertheless, all users should be able to see all displayed content. The permitted length of the table depends on the application and the characteristics of the available display technology. The required width should be within reach of the user’s arm (approx. 100 cm).

- The display must be readable under a relatively wide angle. For example, a 170-cm-tall user sees the other edge of a 150-cm-long and 90-cm-high table under an
3.1. Setup of the Environment

Figure 3.2: Configuration of horizontal and vertical environment elements.

angle of 25 degrees ($\alpha = \arctan(\text{height}_{\text{user}} - \text{height}_{\text{table}} - \text{distance}_{\text{top-to-eye}}/\text{length}_{\text{table}})$).

- For a group of four users, we defined the size of the horizontal element to be 150 cm by 100 cm. These dimensions were tested using a normal table with a similar size.

- For video conferencing with up to four users, four small vertical displays should be mounted on the big horizontal workspace. The small screens show the videos of four remote collaborators.

- The display must be readable under normal office lighting conditions. The illumination of office spaces is around 500 lux. A good luminance value for a display is $100 \text{ cd/m}^2$ [134].

The physical requirements for the vertical element are the same as for existing electronic whiteboard products. The requirements were therefore simply taken from existing products like the SMART back projection display. The SMART board is approximately 150 cm wide, 200 cm high, and 75 cm deep.

If users want to work close to the vertical screen, the diffusion characteristics for the screen must be similar to those for the horizontal screen. If a user writes on one edge of the screen and looks at the opposite edge, he sees the screen under an angle of 14 to 18 degrees ($\alpha = \arctan(\text{distance}_{\text{eye-screen}}/\text{width}_{\text{screen}})$). A comfortable eye-to-screen distance was found to be 40 to 50 centimeters.
Chapter 3. Supporting Interactive Meetings

Both element types have approximately the same screen size, and the user's distance to the screens is also similar. Comparable electronic whiteboards feature a resolution of 1024x768 pixels (XGA). Therefore, the same resolution was required for both modules.

3.1.1 Interacting with the Environment Elements

Interacting with the environment elements has to be as simple and direct as possible.

- Starting the element should be very simple. Pressing a button on an element should also directly boot all other elements in the room and launch the collaboration application without any further interaction.

- As confidential company data can be accessed through the elements, most companies will require the users to authenticate before using the collaboration environment. The authentication should be as simple as swiping a security badge in front of a badge reader.

- The elements should connect to the company’s file servers to import and export data. Each user in the company should have a folder accessible from the elements. If the user signs in for a meeting, the files in his or her shared folder become accessible from the meeting room’s elements.

- External vendors usually do not have any access to the company’s fileserver. Therefore, these users should be able to bring in data to present and annotate during a meeting. Further, they should also be able to take any data resulting from the meeting with them. This could be done, for example, by using pluggable memory devices that participants connect to the elements during the meeting.

- Although the environment is centered around the two types of elements, it should also be possible to integrate mobile devices into the meeting. For example, a user may want to draw or sketch on her Tablet PC and then make it available to all users in the meeting. Supporting laptops and smart phones is also desirable.

- If multiple elements or mobile devices are used in a meeting, their geometrical position relative to each other should be known to the system. This would allow interactions among devices to be supported (see Subsection 3.3.7). Therefore, a system should be developed that detects the geometrical relationship between the elements and any optional mobile devices. An example of such a device topology can be described as follows: One vertical element is to the right of the horizontal
3.2 Interaction Principles

This section outlines what principles should be followed when designing interactions for the collaborative environment. These principles are guided by the target application described in Chapter 1. To summarize, the goal is to support the collocated and remote collaboration of small work groups.

Users engaged in group work have to keep track of many things at the same time. For example:

- Following the discussion.
• Communicating with the other group members.
• Keeping their personal agenda in mind. Trying to make statements at the right time.
• Determining the feelings and the morale of the other participants.

These tasks are already demanding for the user, and operating a computer-supported environment requires even more from the user. Therefore, keeping the cognitive load of the user at a low level is the most important goal for designing the interactions. The strategies in Table 3.1 were put in place to address this.

Table 3.1: Interaction Design Principles

<table>
<thead>
<tr>
<th>Interaction Design Principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The computer-supported environment should provide only basic functionalities as opposed to those for specialized tasks.</td>
</tr>
<tr>
<td>2. Domain-specific applications (for instance, CAx applications) should be used outside of the group work for any required specialized tasks.</td>
</tr>
<tr>
<td>2. The designed interaction should be physically (motorically) easy to carry out.</td>
</tr>
<tr>
<td>3. The user should easily discover the interactions and find them intuitive.</td>
</tr>
</tbody>
</table>

3.2.1 Required Basic Functionalities

From the desired application, the basic functionalities can be deduced. They represent what the user must be able to do in such a scenario as described in Chapter 1. The following list contains the most important features the computer-mediated system must provide. Many of the underlying interactions are know to the user from paper-based setups.

• Draw lines. The user needs to draw lines to sketch, write, and annotate.
• Draw lines with different visual properties (e.g., different colors).
• Delete lines.
• Point at certain parts of the drawings.
3.2. Interaction Principles

- Create cards.
- Sketch on cards.
- Cluster and rearrange cards.
- Delete cards.
- Measure parts of the drawings.
- Change the view on a drawing: zooming and panning.
- Import documents from domain-specific applications.
- Export the sketches and cards of the meeting for later reference.

This list does not contain any interactions needed for the administration of the collaborative environment, like setting up remote conferences and other configuration work. This was considered less relevant for a prototype.

3.2.2 Physical Interaction Constraints

The design principles in Table 3.1 require the interactions to be physically (motorically) non-demanding. This means no special gross and fine motor skills should be required from the user. The time efficiency of the movements can be expressed by Fitt's Law [49]. It shows the limitation of the human motor system by describing the required time to move a target into a specific area at a specific distance. Depending on the muscle group used, a different performance results (see [15] or Langolf 1973). Besides these human movement limitations, the limitations introduced by the setup itself also need to be addressed. The limited human reach on large displays is an important interaction constraint.

Interactions on the horizontal element have to consider that the module has no preferred orientation. Users can gather on any or all sides of the module. Needless to say, the horizontal surface invites users to put random artifacts on the display (see Figure 3.4). The interactions should not be made impossible by these artifacts. The evaluation of these physical restrictions was addressed by trials with subsequent design iterations. For example, pilot-studies were used to characterize the requirements for the input technology.
Figure 3.4: Paper-based setup: Possible artifacts brought in by users include pen, paper, printed matter, and laptops.

3.2.3 Intuitive Interactions and Skill Acquisition

As mentioned in Table 3.1, the proposed interactions should be intuitive and easy for the user to discover. This means that the users should be able to learn how to work with the environment with minimal effort. Ideally, they should be able to walk into the environment for the first time and immediately be able to use it without having to think about what they are doing: They are immersed in the meeting and use the environment intuitively. This expectation regarding the user’s use of the environment can be related to how skill-acquisition has been described in general. Dreyfus et al. [41] introduce a five-step model for skill acquisition (see Table 3.2). The steps are the following: novice, competent, proficient, expert, and master. The user’s decision making is either analytical or intuitive. Dreyfus’s model states that intuitive decisions can only be made by expert or master users. Novice to proficient users have to use their analytical capabilities to make decisions. Furthermore, Dreyfus states that from the novice to expert level, the user is aware of his situation. In our case, he is aware of using the collaborative environment for communication. The master user is immersed (Dreyfus: absorbed) in using the environment. In our case, he is only aware of communicating. He is interacting with the environment without being aware of it. In such a state, the cognitive load
that the system imposes on the user is expected to be minimal compared to his main

task (communicating).

Table 3.2: Skill Levels As They Relate to Human Mental Functions

<table>
<thead>
<tr>
<th>Skill Level</th>
<th>Novice</th>
<th>Competent</th>
<th>Proficient</th>
<th>Expert</th>
<th>Master</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recollection</td>
<td>Non-situational</td>
<td>Situational</td>
<td>Situational</td>
<td>Situational</td>
<td>Situational</td>
</tr>
<tr>
<td>Recognition</td>
<td>Decomposed</td>
<td>Decomposed</td>
<td>Holistic</td>
<td>Holistic</td>
<td>Holistic</td>
</tr>
<tr>
<td>Decision</td>
<td>Analytical</td>
<td>Analytical</td>
<td>Analytical</td>
<td>Intuitive</td>
<td>Intuitive</td>
</tr>
<tr>
<td>Awareness</td>
<td>Monitoring</td>
<td>Monitoring</td>
<td>Monitoring</td>
<td>Monitoring</td>
<td>Absorbed</td>
</tr>
</tbody>
</table>

Obviously, making all users instantly master users is a challenging goal. Since this is a

skill acquisition problem, the only way to make the user a master in no time is to make

use of what the user already knows. So what the user already knows when he first

enters the collaborative environment must be leveraged.

This shows the value of a user’s experience. Most people in industrialized countries are

able to work with pen and paper. From childhood on, users developed their fine motor

skills using pen and paper. They are masters in using a pen in the sense that they

are consciously drawing, but not consciously moving the pen. The use of the tools in

paper-based setups may not have been intuitive in the beginning, but years of practice

engraved their use in the user’s brain.

Therefore, it is crucial for the proposed application to leverage this existing knowledge to

make every user proficient or an expert right from the first use of the environment. The

proposed interactions will therefore resemble or imitate interactions the user already

knows from paper-based work.

3.2.4 Multi-User and Multi-Device Interaction

The importance of leveraging the user’s prior knowledge has already been discussed. This sections stresses the importance of not breaking the user’s expectations.

It might seem trivial, but users are expected to be able to collaboratively draw or erase on a whiteboard (see also [113, 120]). The user does not have to think who will be next to draw on the whiteboard. He can stay immersed in the meeting and wait until the current speaker pauses and then try to take over the dialogue. Besides this, two more reasons were given in Chapter 2 for supporting multi-user interactions: avoiding user frustration and avoiding disengagement.
Chapter 3. Supporting Interactive Meetings

In paper-based setups, the users can interact with both hands. Typically, the non-dominant hand may hold a paper while the dominant hand draws on it. Similarly, one hand might hold a ruler while the other hand draws a straight line. The environment therefore should allow bi-manual interactions. Not only to fulfill the user’s expectations, but also to allow a richer experience using the tactile sense of both hands.

Bi-manual interaction is not all that needs to be supported. The workspace must further allow multi-user interactions, hence enabling multiple users to interact with the system using multiple devices (multi-user-multi-device interactions). Additionally, the proposed workspace may consist of many elements. Therefore, each display should support simultaneous interactions by multiple users, and the user should be able to perform all interactions on all elements. The user does not only have to think about what interactions can be done on which elements. Additionally, to meet the user’s expectations: if a pen draws blue on one whiteboard, it should also draw blue on the other displays. These requirements exceed the requirements of Single Display Groupware (SDG) where all users share one display (see Figure 2.1 and Figure 3.5).

![Diagram](image)

Figure 3.5: Information flow in the extended SDG using multiple displays.
3.2. Interaction Principles

3.2.5 Direct Manipulation

Shneiderman established the three following principles for direct manipulation interfaces (see [114]):

- Continuous representation of the object of interest.
- Physical actions or labeled button presses instead of complex syntax.
- Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

At the time of dialog-based UIs like the command line, the GUI was an emerging interaction technology. But direct manipulation considerations can also be seen independently of GUls in the context of other interaction paradigms (see, for example, [130]). Hutchins et al. explored in [69] the underlying basis of direct manipulation. The authors link directness of interaction with the cognitive load: “At the root of our approach is the assumption that the feeling of directness results from the commitment of fewer cognitive resources.” They name two aspects of the feeling of directness: distance and engagement.

Distance

The term distance denotes the distance between the user’s thought and the physical requirement of the systems. “A short distance means that the translation is simple and straightforward [...].” This distance is further divided into two gulfs (wide gaps): the gulf of execution and the gulf of evaluation. The first is bridged when the interaction modalities of the interface match the thoughts and goals of the user; the second is bridged when the output and the inherent conceptual model of the system allows the user to rapidly perceive it.

The distance of both gulfs can be divided further into the semantic and the articulatory distance. The semantic distance denotes the distance between the user’s intention and the meanings of the expressions the user interface provides. The articulatory distance denotes the meanings of expressions and their physical form.

The articulatory distance relates to what was called physically (motorically) non-demanding in Table 3.1. Hutchins et al. further state that the better the gulfs are bridged and the higher the directness of the interface, the lower the cognitive effort. Figure 3.6 shows the model.
Chapter 3. Supporting Interactive Meetings

Figure 3.6: Distances in user computer interaction (Hutchins, 1985).

Engagement

Direct engagement refers to the feeling that the user is directly engaged with the control of the objects in the application domain (not with the interface itself). For example, the user feels that he is analyzing the data; he does not feel as if he is interacting with a UI of a data analysis tool. This clearly relates to the previously mentioned awareness in skill training (see Table 3.2). In [96], Laurel requires the following for the feeling of direct engagement:

- Execution and evaluation should exhibit both semantic and articulatory directness.

- Input and output languages of the interfaces should be inter-referential, allowing an input expression to incorporate or make use of a previous output expression. This is crucial for creating the illusion that one is directly manipulating the objects of concern.
3.2. Interaction Principles

- The system should be responsive, with no perceived delay between the execution and the results, except where those delays are appropriate for the knowledge domain itself.

- The interface should be unobtrusive, not interfering or intruding. If the interface itself is noticed, then it has a third-person relationship with the objects of interest, thereby detracting from the directness of the engagement.

Some of the items in the list above relate to language-based user interfaces (text-based interface, command line). Such interfaces inherently use language as an intermediary. But, for example, a GUI is able to directly represent the model of an application (domain model). The file icon and the trash bin on the desktop are an examples for such a model world that allows direct interaction. The GUI allows the user to drag the file into the trash bin, whereas the command line tool requires the user to write a command “del C:\project\file.txt”.

Frohlich points to later developments in direct manipulation in [57]. Virtual Reality (VR) systems are designed to completely immerse the user in the model world using technical equipment like head-mounted displays and rooms with stereoscopic projections on many or all surrounding walls (CAVE). VR systems often try to make direct interaction available in this simulated world. A typical example for such a VR system would be an architect immersed into the simulated reality of his development project. He could walk though the design of the future house and change the design of the house in which he is immersed. For example, he could move the walls with his hand to try out different proportions. Frohlich is skeptical about the assumption behind VR, namely that users prefer to conduct many activities in a completely artificial environment. He further points to systems like DigitalDesk [94, 139] that prototype the idea that computers should blend into our everyday activities instead of the user having to adapt to, or even enter into, a computer-generated digital environment. This idea of hidden computers that blend into our everyday life is often referred to as ubiquitous computing and was first presented by Weiser in [137]. Obviously, this fits well with the plan to leverage the experience users already have from paper-based setups.

3.2.6 TUI

The environment should provide users with a richer experience than currently provided by conventional Graphical User Interfaces (GUI). The experience should approximate the user's experience from paper-based setups, in which he can use his sense of touch
and his dexterity when manipulating objects. As noted in Chapter 2, Tangible User Interfaces (TUI) arose from the criticism that GUIs only use a generalized pointer (the mouse) and a keyboard as input devices. Therefore, the concepts of TUI were integrated into the environmental concept to provide a richer user experience.

According to Fitzmaurice [51], a design according to graspable/tangible user interfaces (TUI) is advantageous because it:

- encourages two-handed interactions;
- shifts to more specialized, context-sensitive input devices;
- allows for more parallel input specification by the user, thereby improving the expressiveness or the communication capacity with the computer;
- leverages off our well developed, everyday skills of prehensile behaviors for physical object manipulation;
- externalizes traditionally internal computer representations;
- facilitates interaction by making interface elements more direct and more easy to manipulate by using physical artifacts;
- takes advantage of our keen spatial reasoning skills;
- offers a space multiplex design with a one-to-one mapping between the control and the controller; and finally,
- affords multi-person, collaborative use.

These advantages of TUI are in line with the general reasoning of the interaction principle for the proposed collaborative environment. The broader input capabilities minimize the articulatory distance of the interface, further specialized and context-sensitive input devices allow to reduce the semantic distance. Therefore, the TUI manipulation techniques feature a high level of directness.

The following list shows how TUIs are used for interactions (see also Chapter 2):

- A TUI may serve as a handle to digital information (see for example [52] and Figure 3.7). Because multiple handles are supported, interaction is accelerated and simplified. The user’s hands can manipulate different handles simultaneously. Additionally, a handle can be continuously attached to digital elements to quickly allow future access to them.
3.2. Interaction Principles

Figure 3.7: Photomontage to illustrate the idea of using an input device (the grey device is hereafter called handle) to control the position and orientation of a digital card. The digital card is labeled with “Bodenheizung”.

- The physical TUI handle tool is coupled with a digital model in order to make a certain functionality easier available (see [131], for example). A TUI device could be created to staple a stack of digital cards, for example, in order to make grouping cards a richer experience. If the physical staple device is pressed, the cards on the display would be logically grouped together (see Figure 3.8).

- Two TUIs can be physically combined with each other to provide new functionality (for example, see [132]).

In Ullmer et al. [130], an interaction model for TUI is presented. In analogy to the widely known model-view-controller(MVC) GUI design principle (see Figure 3.9), it is called model-control-representation physical and digital (MCRpd). See Figure 3.10. In MVC, the controller software component is responsible for handling the input, the model represents the domain model the software provides to the user (word processing, CAD, etc.), and the view is the part that generates the output shown on the screen. For a TUI, the user can directly grasp the control. For TUIs, the view is replaced with representation. This representation is not only the output on the screen, but also the physical device itself. Therefore, the control and the physical representation is drawn as linked together in the diagram.

Further, by controlling the physical device, the user directly interacts with the underlying model. In the handle example in Figure 3.7, moving the handle directly updates the digital model of the card (changes its position and orientation).
Chapter 3. Supporting Interactive Meetings

As mentioned earlier, the user’s knowledge from paper-based setups should be leveraged as much as possible. TUI’s specialized representation (physically as digitally) seems ideal for making the devices resemble the tools the user already knows from paper-based setups. So a ruler can look like a ruler, an eraser resembles a true eraser, etc.

The following sections describe in more detail how the input devices relate to the original tools used in paper-based setups. A metaphor denotes an analogy between two objects or ideas. The original idea is called the “Tenor”, while the idea used to transform the first is called the “Vehicle”.

The following set of manipulations can be used to analyze a user interface design that
3.2. Interaction Principles

Figure 3.10: Interaction model of the TUI: MCRpd (Ullmer/Ishii 2001)

uses metaphors (see [5]):

- Manipulations serving an underlying application function (Set V; an example is the ability to cut and paste text).

- Manipulations having a meaning with regard to the metaphors used for the mental model of the interface (Set M; an example is the use of scissors and clipboards a metaphors).

- Manipulations provided by the user interface (Set I; two buttons with scissors and a clipboard icon to cut the currently selected text and paste previously cut text).

The successful use of metaphors for a user interface will lead to manipulations being included in all sets. In the following, two examples will be used to highlight more problematic cases. If an operation is not in Set M, the manipulation could not be mapped to the metaphors used by the model of the interface. For example, in the Mac OS, the compact disk (CD) drive is mapped to an icon of a CD. The metaphor used in the mental model of the UI is the physical medium. Double clicking on the CD will play or open the content of the disk, but dropping the CD into the waste bin will eject the CD. In the model of the metaphor, this would destroy the CD. If an operation is not in Set V, the manipulation has no underlying application function. This means the manipulation only serves the model of the metaphor. For example, users tidy their desktops up. This
manipulation makes sense for the metaphor of the desktop, but has no meaning for the underlying application functions (store files in folders with file names, manipulation dates, and so on).

Another model for the application of metaphors was presented by Fishkin in [48]; the model distinguishes different ways metaphors can be applied for TUIs. Among them is the usage of a metaphor as a noun, meaning that the object’s physical properties create an analogy to an object the user already knows. Another way is using a metaphor as a verb, which refers to an analogy created by the usage of the metaphor. An example would be shaking a TUI map to initiate the action of clearing.

Fishkin also introduces the term embodiment in the same paper. It relates to what is called distance in the direct manipulation terminology. Embodiment can be characterized by a couple of gradations: full, nearby, environmental, and distant. Full embodiment means the output of the TUI completely matches the input (same physical body). The user manipulates a TUI and gets the system’s feedback exactly where the manipulation took place. On the other end of the scale, distant means that the reaction to an input takes place at a spatially unrelated place. This is like the interaction on a remote control (input) that causes a reaction on a TV screen (output).

To create analogies to interaction techniques the user already knows from paper-based setups, the user should be able to manipulate the physical state of the input devices. The feedback regarding this state change should be temporally and physically close to the user’s input.

The following types of state changes should be expected for the interaction with the environments:

- The device is brought into the area of the display. It is detectable.
- The device is moved on the display. The position state is changed.
- The orientation of the TUI can be used to enable interactions. For example, a sound dial TUI could use the orientation for allowing the user to control the sound volume.
- The user might push a button embedded in the device. The button state is changed either by pressing a side of a TUI or by pushing the TUI on the interaction surface.
- The user might move some parts of the TUI when interacting (slider, dial).
3.2. Interaction Principles

Changes in the state of the input devices are then used to trigger some operations in the collaborative environment. For the example given above (the stapler device), pressing the device on the display could be used to trigger the operation of grouping the cards underneath the device.

TUI devices are designed to be shared among users. This corresponds to paper-based setups, in which the number of scissors, erasers, and rulers is also limited. As the use of these tools is more specialized than the use of the pen, it is not expected to hinder the user when he tries to externalize his ideas in a discussion.

Though if multiple TUI devices are needed for certain applications, the simultaneous use of these devices must also be supported. Nevertheless, their functionality is not bound to a specific user.

3.2.7 TUI and Pen

From the paper-based setups, users are trained to interact with both hands simultaneously. For example, the user might hold a card while drawing on it; he might hold a ruler while drawing a line; or he might cluster cards with both hands at the same time.

In the simplest case, the user just does two independent interactions simultaneously that he could also have done consecutively. For example, he moves two cards using two physical handles at the same time. The two interactions do not interfere. The result is the same as if they were carried out sequentially. The order in which they are carried out does not affect the end result.

Another case of bi-manual interaction arises if the interactions of each hand are related to each other. They cannot be performed one after the other in any order. Such interactions were explored using TUI and pen. Two types of interactions were identified:

- The static relation of the TUIs to each other can be used for interactions. The position of a TUI can either be expressed in the coordinate system of the display or in the coordinate system of the other TUI. Depending on the pen’s position in relation to the TUI coordinate system, a different action might be triggered. This type of interaction can make use of digital overlays assigned the TUI devices. The overlay is part of the digital representation of the TUI (MCRdp see Section 3.2.6). An example for such an interaction is a yellow rectangle attached to a TUI that the user can draw upon using his pen. It is a virtual card or Post-It™.

- Another type of interaction uses the dynamic relation of the TUIs. This can be
expressed as gestures between the interaction devices. For example, if the user draws a circle around a pencil sharpening tool, the diameter of that circle is used to change the line width assigned to the pen (see middle part of Figure 3.11).

Further, both interaction types can occur with or without physical contact among the interaction devices. Therefore, their movement is governed by their shape. An example for such an interaction would be a digital pencil sharpener featuring a hole for the pen and a crank to simulate the sharpening process (see left part of Figure 3.11). This interaction uses the dynamic relation with physical contact among the devices.

The two types of relations can also be compared to the way they use metaphors to create an analogy to something the user already knows. The static relation can use **noun** metaphors created using the digital representation (see Section 3.2.6). The dynamic relation can use **verb** metaphors.

In the following, the previously mentioned interaction criteria (see Section 3.2) are used to evaluate the interaction types listed above. Using gestures (dynamic relation) with the TUIs does not seem feasible for two reasons: the gestures are not obvious to the user and it is difficult to find appropriate gestures between the TUIs that are already familiar to the user. Interactions using a static relation of the TUIs and an overlay appear to be more intuitive for the user. The overlay can be used to indicate to the user what interactions are possible with this device. Since the digital representation of the TUI moves with the device, it seems ideally suited for the special requirements of a horizontal element. Users can gather around the display and move and rotate the representation to suit their needs. Interactions that involve physical contact between the input devices must also either use a static or a dynamic relation for interaction. But for the case of physical contact, it seems easier to find interactions that the user already knows from paper-based setups (for example, a pencil sharpener, drawing stencil, and so on). In any case, because physical contact between the devices is required for the
interaction, no digital representation of the TUI is involved, resulting in an interaction that appears to be less mediated by the system.

3.3 Proposed Interactions and Input Devices

This paragraph outlines how the basic interactions may be specified for later implementation is outlined. Important decisions such as which input devices are provided to the user and and how they can be used are also discussed.

As mentioned in the previous section, the number of interactions should be limited to not overwhelm the user during the collaboration. To further improve the usability, the user’s prior knowledge should be leveraged when designing interactions and their corresponding input devices. Devices should indicate their functionality by resembling an object known from the familiar paper-based environment. The most basic example is that an input device for sketching should look like a pen.

3.3.1 Pen

In paper-based group work, the user interacts with the pen on paper to create sketches or annotations. He also uses a pen to point at certain elements of a sketch.

To support these interactions in digitally (in software and hardware), the tip of the pen must be represented as a pointer. A digital pen, resembling a board pen the user already knows from paper-based setups, is proposed as a control for the pointer.

In paper-based setups, the user can choose from a large set of pens depending on the desired properties like color, line width, transparency (marker pen), and so on. In contrast, normal whiteboards usually have only four pens with the colors red, green, blue, and black. Therefore, the system is required to provide these four pens to mimic normal whiteboards.

Nevertheless, all users may need a red, blue, green, and black pen to annotate existing documents. A decision was therefore made to provide an environment in which each user has a pen. The user is given means to change the visual properties of the lines drawn by the pen.

With regard to the basic functionalities listed in Table 3.1, the pen in designed to be very simple. It is used for drawing lines and pointing to parts of the drawing. The latter requires that a remote participant's pen be represented on the screens.
3.3.2 Eraser

An eraser TUI is proposed, allowing the users to erase unwanted lines. Like its physical counterpart, the user should be able to change the size of the erased surface by changing the way the TUI is held against the display: using a corner only erases a small portion, holding an edge on the display erases a larger area, and holding the front face of the eraser on the surface erases the largest area of the drawing (see Figure 3.12).

Figure 3.12: Erasing with different parts of the eraser device causes different area sizes to be cleared.

3.3.3 Handler

In the previous sections, the collaborative environment was required to support the following basic functionalities using cards: create a new card, move a card, and delete a card.

Previous projects [78] used physical cards which are tracked and digitized on a white board using input from rear and front-mounted cameras. This has the advantage that it is closest to what the user already knows from paper-based work. Nevertheless, the user is required to work with two kinds of pen: one for the work with the physical cards and one inkless pen to work with the electronic whiteboard. To bridge this gap in the presented work, only non-physical cards were used. The cards are therefore never physically represented; only their digital representation is displayed.

The following explains how the handle can be used to perform the previously mentioned interactions (see also Figure 3.13). The user creates a new card by putting the unassigned handle on the display. The newly created card is then assigned to the handle and follows the movements of the handle on the display. When the user removes the TUI from the display, the association with the card should be removed. This means the card stays where it was while the handle can be placed on another spot of the display.
3.3. Proposed Interactions and Input Devices

If the user places the handle on an existing card, the handle and the card will be linked. Additionally, the user can remove the TUI while squeezing the sides of the TUI. This would leave the card assigned to the TUI. If the handle is then placed again on the display while the sides are still squeezed, the assigned card then also appears on the display. The user can drop (delete) the card by releasing the grip after taking the TUI.
with an assigned card off the display.

The following interactions are not required by the basic interactions, but they could make the interaction with the environment more complete. The user can copy a card by putting and then pressing an unassigned handle on a card that is already assigned to a handle. This will copy the card and assign it to the previously unassigned handle.

These interactions are envisioned for small Post-It™ cards; they should also work for handling drawings and other design documents. While the cards are generally smaller for brainstorming, the cards need to be bigger for design review to hold and display a drawing or other documents.

### 3.3.4 Cutter

The previous section explained how the handle device can be used to manipulate cards and that the cards can contain graphical content like sketches, drawings, and other documents.

In paper-based setups, the user can draw on the whiteboard or on a Post-It™. To imitate this, a user should not only be able draw on cards, but also on the background representing the actual whiteboard. Unlike with whiteboards and Post-Its™, a user should be able to copy sketches and drawings from a card to the background. This action should be triggered when the user pushes an assigned handle on the display (see Figure 3.14).

![Pressing a handle on the display to copy its content on the background](image1)

![Cutting content out of the background to create a new card](image2)

Figure 3.14: Card-background interactions using handle and cutter knife.

The cutter knife TUI is proposed to do the reverse: getting content from the background on a card for further manipulation (see Figure 3.14). If possible, the cutter should not
only work on the background, but also on the cards on the display. This interaction is thought to be similar to operating a real cutter knife.

Like a real cutter, the tip of the cutter TUI should be pushed on the display to start cutting and releasing will stop cutting. To guarantee that the piece of the background is entirely cut out, the line between the start and the end point is cut by the system (see Figure 3.14).

### 3.3.5 Ruler

One requirement is that users be able to measure a displayed drawing. Further, users should be able to scale and pan the displayed content. The measurement displayed on the ruler is determined by the scale of the drawing. Therefore, an attempt was made to address the functionalities with a single TUI. As the section’s title already indicates, this TUI should resemble a ruler.

Like any ruler, the TUI should contain a scale that allows the user to get an impression of a drawing’s dimensions. Similar to a slide caliper, a sliding mechanism should allow users to measure certain parts at high precision. The measured distance should be displayed to the user (see Figure 3.15).

![Figure 3.15: Measuring interactions with the ruler TUI.](image)

To move and rotate, the user should press the ruler on the display (see Figure 3.16). The pressing gesture should denote that the ruler is now linked to what should be modified. This interaction is meant to resemble paper-based interactions. If a ruler is pushed on a drawing, the drawing will follow the movements of the ruler. To simulate this to the user, the object manipulated should follow the movements of the ruler (translation and
rotation) while it is pressed on the display. Depending on what is under the ruler when the interaction is started, the interaction will either change a card or the background. Changing the background will effectively alter the view of all content.

While printed drawings can obviously not be scaled in real time to reveal a particular detail, a drawing on the display can be interactively scaled to meet the user’s need. The scaling interaction should be implemented using the sliding mechanism while the ruler is pushed on the display. Similar to the translate/rotation interaction, the illusion should be maintained that the ruler is actually connected to the displayed content. Therefore, the points under the ruler’s origin and the readout should stay connected while scaling. Scaling should work on either a card or on the background (view).

In many areas of technical drawings, it is desirable that lines be drawn only straight and
only at certain angles relative to the ruler (for example, with only 30 degrees steps). This means that no curved shapes and free hand lines are allowed for these applications. Since the ruler is used to draw straight lines in paper-based setups, intuition suggests that such an interaction would also be linked to the digital ruler. Therefore, nothing must hinder the user when drawing a straight line using the edge of the ruler (see Figure 3.17). Further, if the user holds the pen at the edge of the ruler, this will cause the two devices to be linked. If a pen is linked to a ruler, it can then only draw straight lines under certain angles relative to the ruler. To remove the linkage, the user has to hold the pen again at the edge of the ruler.
3.3.6 Color Tool

One requirement in Section 3.2 is that the user be able to draw lines with different colors. This can be done by providing dedicated pens for each color, or for certain applications, it might be necessary to make a pen's color changeable (see Section 3.3.1). In the following, the interaction to change the color of the pen is presented.

A related interaction that users know from paper-based setups was sought. Unlike colored pencils and felt pens, fountain pens offer the possibility of changing their color by filling them with a different ink. Further, brush and color palette can also be used as a metaphor from paper-based setups.

![Choose color](image1.png) ![Assign color to pen](image2.png) ![Pick color](image3.png)

**Figure 3.18: Color tool interactions.**

One proposal is that the TUI (color tool) should resemble a combination of an ink well and a color palette to support the interaction. The device should be ring shaped so that the user can dip the pen into the device to change the color (see Figure 3.18). To change the color of the virtual ink in the color tool, the user can rotate the device. In order to make this interaction more apparent, the color tool should feature a color ring with a mark indicating the selected color.

The color is selected from the displayed background by pushing the color tool on the display. Similar to the ruler device, this should denote that the device is now linked to the display. The center of the color tool now features crosshairs to pick the desired color. The colored ring around the device shows the current color. If the user is satisfied with his selection, he can release the pressure on the device and dip the pen into the color tool. Rotating the device brings the color tool back into the normal mode.
3.3.7 Laser Pointer and Multi-Display Interactions

One of the interaction requirements in Section 3.2 is that users must be able to cluster cards. Creativity methods commonly involve users writing their ideas simultaneously on cards. The cards can then be clustered according to certain criteria. Usually, the cards are written on a table and then stuck on a whiteboard. In the collaborative environment, a user can write on the digital cards on the horizontal element and cluster the cards on the vertical elements. The handle interaction of removing a card from the display and putting it on another display should therefore preserve the content of a card attached to the handle (see Figure 3.19).

![Figure 3.19: Multi-Screen interaction with the handle TUI.](image)

As an alternative interaction for the same use case, a laser pointer TUI is proposed. It is shaped similar to the pen, but allows the user to drag a card from one display to another. Therefore, the user points to the card on one display and squeezes the pen to indicate that he would like to drag the card. The card then follows the dot of the laser pointer device. The user can then move the card on the display and even point at another display and the card should appear under the laser's dot on this display (see Figure 3.20). When the user releases the pressure on the pen, the card will stop following the laser pointer’s dot.

3.3.8 Folding Import Tool

Importing data from domain-specific applications is a required interaction. The users should be able to specify the data to be automatically imported when the environment
starts up. More data may be required depending on how the discussion evolves. To support such ad-hoc imports, the folding fan tool is proposed. The folding mechanism was chosen to represent that it will open something, like opening a new book or opening a new drawing. A drawer mechanism could be used as an alternative, but is omitted since it would need more space on the display. Since the fan has a circular shape, it better fits the scenario of all users gathering around a table. So they might see the tool from any side, not necessarily from the side where the drawers open.

If the user opens the folding fan, one segment after the other would be visible at the minimum opening angle (see Figure 3.21). Each segment would show a preview of the underlying document, an icon, and the name of the document or subfolder. The user can put the handle on the desired segment in order to import the document or to dive into a subfolder. In the case of a imported document, it would then be attached to the handle for further use.

To make the document name and the preview readable, a minimum angle for the segments should be defined. If more documents are in the listing than would fit in to the opening angle of the tool, the user should still be able to navigate through all documents. Therefore, making the opening angle of the fan smaller by moving one leg of the fan should not rotate the displayed documents back, but opening the fan should rotate the documents. This enables the user to navigate through all documents by opening and closing the fan. This navigation technique can be supported by letting the opening movement give the documents a spin. Besides waiting for the simulated friction to stop the spinning of the documents, the user can also press the handles of the fan.

In order to make more documents visible in the fan, the visualization could be changed
3.3. Proposed Interactions and Input Devices

Figure 3.21: Folding fan interaction for dropping documents from a pie menu to the workspace.

from circle segments to a spiral. In this case the center of the spiral would be at the intersection of the two legs of the fan. The fan handles would only serve to spin the spiral outwards or inwards towards the center. The documents get smaller as they spiral towards the center. To limit the screen space used, the documents eventually have to disappear as they move outside. The spiral can therefore not be closed. As only few documents were expected to be in the list, a decision was made to start with the segments and try out the spiral if needed.
3.4 Tracking Technology and Input Devices

From the interactions defined in the previous section, the requirements for the input technology should be derived. Besides this synthetic approach, two pilot studies were conducted to gain some practical experience. The study findings should not only contribute to specifying the requirements of the input technology, but also provide valuable knowledge regarding building the physical workspace. The pilot studies are listed in the following subsection. The results of the studies are evaluated and integrated into the requirements for the input technology, which is discussed at the end of this section.

3.4.1 Pilot Studies

As mentioned before, two pilot studies were conducted. Both featured their own input technology. The first used a color-coded optical tracking system. The second was built on top of a commercial optical tracking system. Both studies used the same workspace built for the color-coded optical tracking system.

Color-coded Optical Tracking System for SDG

In Kunz et al. [80, 128], a first prototype of the horizontal workspace was presented. It showed the construction of the physical setup and a prototype of a multi-user input technology for SDG.

![Pen for the use with super bright Color LEDs.](image)

Figure 3.22: Pen for the use with super bright Color LEDs.

Different alignments for the projector and the camera were evaluated. Compared to normal projector applications, the users look at a screen with a flat angle. This requires projection material with a wide and homogeneous diffusion of the projected light. To comply with this requirement, many screen materials were evaluated in [119].

A pen was created which consists of a super bright LED, a battery, a micro switch, and a resistor mounted on a modified felt pen. Figure 3.22 shows an image of the pen. The
optical tracking technology operates in the visible spectrum. If the user presses the pen on the display, a micro switch turns on the LED in the pen tip. The spot created on the projection display is brighter than the projected image. A camera mounted underneath the table acquires images of the projection screen.

The spots are detected in the acquired image and the projection image is discarded. To assign the spots in the image to a pen known to the system, the spot’s color value is converted into a three-dimensional vector within the RGB color space. This vector is then compared to the reference color vectors known to the system. The spot is assigned to the closest reference value in the color space. A demo application proofed the principles of the input technology. It displayed the detected coordinates and the identification of the pen (see Figure 3.23). The input technology of this pilot study allowed the position and the identification of up to four pens to be detected. Extending this principle proved to be difficult, since the system could not reliably distinguish more than the four colors due to the limited number of color available for super bright LEDs. Further, the superimposed projection degraded the color information, making it unreliable when
assigning the correct reference color. The *hot spot* is the point on the display where the projector’s light reflects or shines through into the eye of an observer. The other areas on the display only diffuse the projector’s light and are therefore darker. The *hot spot* is usually avoided by positioning the components so that no observer is in the area where it occurs. Camera use wide angle lenses to fit into the setup. The resulting aperture angle and physical contrains specific to this setup caused the camera to acquire images with the *hot spot*. Detection of the LEDs was therefore obstructed in the areas where the *hot spot* appeared in the image.

While some of the issues could have been solved by working on the setup (the image acquisition and the image processing), other restrictions have made it advisable to seek a new input technology. Only one state and no hovering of the pen is supported. No cursor is visible to the user until the pen is pressed on the display, causing an action. Additionally, users reported being distracted by the bright light of the LED mounted on the tip of the pen.

**Commercial Optical Tracking System for SDG**

More experience with the implementation of SDG applications was gained in another pilot study. It also allowed the further refinement of the requirements through practical work on a demonstrator. In the setup [112], a commercial optical tracking system [91] was mounted on the ceiling above the interactive table to track the input devices. Due to the minimum required distance between the system and the tracked volume, it could not be mounted beneath the table (see Figure 3.24).

The stylus (Figure 3.25) consisted of a plastic tube with reflectors for the tracking system attached to it. The states of the buttons on the tube (tip and additional button) were transmitted by the printed circuit board taken from a wireless mouse.

As the system was invisible to the user and allowed hovering, it was superior to the previous system. Unfortunately, the setup turned out to not be suitable for the interactive table setup as users generated hindrances in the optical path and occluded the markers. Additionally, the optical tracking system requires three markers to be fully visible, which was not the case with the upright position of the pen. The markers on the two pens were arranged with different geometry to make each pen identifiable. The results of the two pilot studies are summarized in Table 3.3.

A sketching application was developed for this setup. It allowed the users to draw freehand lines on a back-projection display by combining the positional input from the tracking system and the mouse button events. For each connected PS/2 mouse, an
3.4. Tracking Technology and Input Devices

Figure 3.24: Setup of the pilot study using an optical tracking system for SDG.

An individual device was opened and the PS/2 protocol was interpreted to generate events for the drawing application. Further, two more users could simultaneously interact using regular mice.

Lessons learned from the pilot studies:

- The setup should avoid the occlusion of the input devices by users. Preferably, the detection should occur through the display.

- The detection technology should be imperceptible so it does not irritate users.

- The input devices should offer device states for hovering, drawing, and bringing up a context menu.

The lessons learned from the pilot studies influenced the requirements listed in the following section.
### Table 3.3: Results of the Pilot Studies of Two Alternative Tracking Systems

<table>
<thead>
<tr>
<th>Pilot Study</th>
<th>+ or − Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color-coded Tracking System</td>
<td>+ Free line of sight, no occlusion.</td>
</tr>
<tr>
<td></td>
<td>− Irritation through perceptible light.</td>
</tr>
<tr>
<td></td>
<td>− No additional states. Pen is undetected or drawing.</td>
</tr>
<tr>
<td>Commercial Tracking System</td>
<td>+ No irritation.</td>
</tr>
<tr>
<td></td>
<td>+ Additional states, supports hovering.</td>
</tr>
<tr>
<td></td>
<td>− No free line of sight, occlusion through users.</td>
</tr>
<tr>
<td></td>
<td>− Markers on input devices hinder interaction.</td>
</tr>
</tbody>
</table>

![Figure 3.25: The pilot study’s interaction devices in use](image)

### 3.4.2 Requirements

The requirements for the input technology were derived from the technology’s application in the computer-supported environment. The following requirements were setup to support the previously presented interactions using pens and TUI:

- Four people should be supported to simultaneously interact on a vertical or a horizontal back-projection display.

- The input technology should not be intrusive to the user. No cables should hinder the user’s interaction on the display. As the user wants to interact on different vertical and horizontal displays, the user should be free to move around the table as she wants.

- The input devices should be identifiable by the system. This will allow a device
to be mapped to attributes like color and line width. Additionally, the identification should be invariant across different displays, allowing the input devices’ attributes to be synchronized across the computers involved. This seems natural, as the properties of the physical pen do not change much between different drawing backgrounds. The identification also allows a pen to be mapped to a user. This way different authorizations on the system can be granted depending on the specific users or on the users’ group affiliation. An example is the case of a moderator having more rights than a normal participant.

- The interaction devices’ detection has to be robust, since users will put all kinds of objects onto the interaction surface like elbows, mobile phones, coffee cups, and laptops (see Figure 3.4).
- Experience with older systems [80] for tracking multiple styli on projection displays by using color LEDs showed that the detection system should be imperceptible to the user. Thus, the emitted light should not irritate the users.
- Smooth sketching is attainable with a system that supports a refresh rate of at least 10 Hz. An analysis of the scribble writing developed for the Palm PDA indicated a minimum refresh rate of 12 Hz. Following [39], 10 Hz is the bottom line. However, a higher refresh rate is highly advisable.
- The delay between the user’s motion and the system’s graphical response should be less than 120 milliseconds. This response time has been chosen from the delays found in existing commercial single-user systems [44].
- As defined, the input technology must support three states to support state-of-the-art interaction with the stylus. This definition is based on the interaction models described in [12]. The input device (like a pen) might be out of range, in the tracking (hovering) state or in the dragging (drawing) state.
- Another requirement is the extensibility of the system. A former tangible user interface [52] co-developed by the ICVR group was too limited for some engineering applications, since the underlying tracking technology only recognized the orientation and position of the input devices, but not their current states. Therefore, the input technology should support additional input capabilities on the devices, like buttons and sliders. These are required for enabling the interactions previously mentioned in this chapter. The ruler features a slider for scaling. To detect whether a user is squeezing, additional input capabilities like buttons must be used. The same is true for interactions like pushing the color tool on the display.
3.5 Remote Collaboration

As stated in Chapter 1, the collaborative environment should be used to allow remote collaboration. For example, two horizontal elements in two different locations of a company should allow a distributed work group to collaborate.

Therefore, a drawing shown on one display element should also appear on the other display. Further, the users’ sketches and annotations should also be visible on the other site with very little delay (< 1 second), so that discussions can rely on what the users draw.

Occlusion of the actual content is prevented by showing only the digital representations of the TUI on the local display. The digital pointers of the users’ pens allows pointing gestures among users in different locations. Thus, a pen’s pointer must also appear at the remote side.

Each site may contain a different number of display elements. For this prototype, no special strategy has been proposed for adapting to such situations. Therefore, all sites in the current study should have the same number of elements.

3.5.1 Video Conferencing

When doing remote collaboration, all team members not only want to exchange data like sharing sketches, but also have to see and hear each other over audio and video channels. Thus, the workspace has to be enhanced by additional audio and video components.

To facilitate a more interconnected atmosphere between local and remote users, monitors showing remote participants should be arranged around the table. In addition, cameras should be mounted on top of those monitors to facilitate eye contact with remote users. First tests showed that for intuitive collaboration, it is important to have correct eye contact with the remote collaboration partner. Eye contact is an important part of non-verbal communication. It supports a coordinated discussion and also helps to understand the spoken word. Additionally, loudspeakers and microphones should be integrated into the monitors for a correct spatial correlation between the video and the audio signal.
3.6 Software

In order to enable users to sketch and annotate on cards, a software application had to be developed. The software can be divided into two parts: input event handling and the client application (the groupware). First input event handling is described, then the client application is presented.

3.6.1 Input Event Handling

An input technology transforms the user’s interactions into a data stream using a specific protocol. For example, a mouse transforms the user’s interactions on the mouse into a data stream conforming to the PS/2 protocol. Normally, an operating system provides device drivers (software components) for these protocols. If the protocol is proprietary, the vendor must provide a device driver for the input device for the user’s OS. These device drivers transform the protocol’s way of communicating the interactions into the way the operating system represents them. This representation is then provided to client applications through an application programming interface (API). For the presented work, this standard approach implies some difficulties which are explained in the following section.

Components of an Operating System’s GUI

As stated in chapter 1, current OSs only support the concurrent input of a single user per graphical terminal. Needless to say, this statement is general and unspecific. The current section gives a brief overview of the components involved.

The operating system can be described as an interface above the hardware. It was conceived to help the software developer manage the complexity of the underlyng hardware. By doing so, it allows the software developer to concentrate on the domain-specific needs of his customers. For the case presented here, these needs are annotating existing documents with sketches. Further, the OS manages the access of applications to shared resources. Examples of such resources are the system’s memory, processor time, disk access, and access to other devices like printers. The operating system makes it possible to simultaneously run multiple applications on one computer.

In the past decades, the user interfaces provided by operating systems have changed considerably. Today, the graphical user interface (GUI) is state-of-the-art for desktop computers. The main elements of the GUI are: windows, icons, menus, and pointer
(WIMP). Text-based command line interfaces are still used for software development and administration tasks, but usually they are used in a window within a GUI environment.

Following the terminology of the X11 Window System [56], the minimal components involved in building a GUI are:

- The display server
- The window manager
- toolkits for building the actual GUI

The display server controls the application’s access to the screen area by taking control of the output devices (graphic cards) and managing the application’s access to this system resource. Further, it connects to input devices (traditionally keyboards and mice) and provides an event system (interprocess communication) to distribute the input data to applications.

The display server provides only functions for drawing primitive two-dimensional graphical elements like lines, rectangles, and patterns.

The window manager defines how the application’s window is drawn on the display. It also provides the user interface (UI), allowing the windows on the screen to be modified. For example, the UI provides buttons to change the state of the window (like minimize, maximize, close). The UI may also provide a title bar for dragging the window and a border allowing the user to resize the window. The detailed features may vary greatly among different window managers.

A widget toolkit provides the actual visible elements of a GUI (the widgets) - menus, sliders, buttons, and icons. For example, the application creates a menu bar, adds a menu item with the label *File Open*, and specifies instructions to be called when the user selects this menu item. The client application chooses which widget toolkit it uses. Therefore, applications using different widget toolkits can be used on the same screen.

Making the GUI of an OS support multiple user involves changing all the components listed before. The input events created by the display server must identify the device causing the event to be created (e.g., Pen 3 started to draw). The window manager must allow multiple users to change the appearance of the windows. The widget must communicate which input device was used during interaction with the widget. For example, the widget should not only notify the client application that the “OK” button was pressed, but also that the “OK” button was pressed by Pen 3.
Requirements Event Handling

In the Appendix A, an evaluation of input APIs is provided. None of the evaluated APIs provide exactly what would be needed for the setup. Often their use would provide some benefit, as they would allow some interoperability with existing client applications. But as this often means that only one stylus or some subset of the devices could be used with the existing application, it does not seem to be a big benefit. The downside is that additional development effort is required to make the input technology fit into the API. What would also restrict the flexibility of development is new requirements arising during development. Therefore a custom input-event handling system was developed to which the groupware can interface.

3.6.2 Client Application: DOG

The client application was named DOG from Distributed Object-oriented Groupware. The main purpose of DOG is to serve as a test environment for an iterative development of the input technology, the interactions, as well as the collaborative environment and its driving concept.

The most important requirement was to allow multiple users to interact with a data model using styli and TUIs. The data model consists of:

- **Free hand drawings** by the user: Defined as a number of strokes, consisting of the vertices that define the lines. The stroke should contain visual properties like color, line width, etc.

- **Documents** are external graphical content from domain-specific applications. Documents cannot be edited in the groupware, they can only be annotated. A document can either be a drawing, an image, a text document, or a 3D model. Supporting standard data formats for these data types is required.

- **Cards** are areas on the screen that can contain free hand drawings or documents. Cards are drawn on top of the background and can be attached to handles in order to move them around.

- **Digital representations** of the TUIs (like the scale of the ruler) must be drawn on top of the cards and the background. Digital representations must follow the movement of the TUIs they represent.
• **Background** contains sketches and documents. To move them around, the user first has to first put them on a card by cutting them out.

Additionally, DOG should address issues specific to collaborative environments such as the presence of multiple, large interactive displays driven by individual computers, adhoc integration of computers the users bring into the environment, or remote collaboration. Further, DOG should be used to conduct a qualitative user study evaluating the interaction concept presented. Last but not least, it has also been used as a demonstrator for industrial and academic partners.

DOG should allow the demonstration of different applications of the environment and its interaction concept. Some examples are technical design reviews or creativity meetings. Therefore, the software should be developed in a very generic way, and it should not be trimmed towards one specific application. For example, creativity methods require simple 2D sketching, but design reviews require 3D content viewing. As 2D graphics can be easily implemented on top of a 3D system, DOG was designed as a 3D software.

As mentioned above, the prototype should allow multiple collocated displays and remote collaboration. DOG should therefore communicate over networks with other instances of the DOG application. To allow many experiments with the software, not only the actual content of the data conference should be shared, but the information concerning the user interface (UI), the physical setup, and session.

• The shared **graphical content**, like sketches, cards, annotations, and the imported documents, should be visible and manipulatable for all users on all sites.

• The **shared user interface** data enables DOG to show the pointers of the remote participants, a requirement stated in a previous section. Further, it can be used for remote collaboration applications following the "What you see is what I see" (WYSIWIS) concept (see [118]). WYSIWIS interfaces offer the participating users shared windows. The content of the shared windows is identical on any participating machine. This ensures that all users have the same view on the shared data. Manipulating the view on the graphical content on one site also changes it on the remote site.

• The shared data describing the **physical setups** allows interactions and setup configurations that rely on the configuration of the devices and displays. One example is automatically setting up the views on multiple large displays. The virtual camera used on a horizontal display can be perpendicularly aligned to the
camera used to view the same virtual scene on a vertical screen. The shared data can also help set up video and audio stream for video conferencing. The video stream should be positioned on the display just below the camera. As described in Section 3.5.1, this improves eye-contact among participants. Further, the user might want to share data from his laptop. He could do so by moving a file using his touchpad from his laptop to a display of the environment. Such interactions should be made possible by using the device setup configuration and a system that tracks the geometrical relation among the devices (see Section 3.1.1).

- The shared data concerning the **session management** makes it possible for the user to see who is participating in which conference from which site. This also allows the user to choose which conference to join.

### 3.6.3 Development Platform

The environment presented here has been conceived in the sense of "Ubiquitous Computing" (see [137]). Similar to i-Land [123], the table and the whiteboard device should blend into the office like normal furniture. Therefore, they should have the ease of use of a coffee machine rather than the complexity of a full-featured operating system (OS) with a graphical user interface. Instead of navigating through a startup menu and trying to find the right menu entry, pressing a button on the display element should be enough to launch the client application.

To allow a quick startup of the system, the developer should be able to customize the operating system for a dedicated purpose and leave out unused parts of the OS. This customization makes an embedded OS favorable. Nevertheless, it must support real-time 3D graphics and connections to file and database servers. These requirements made Windows CE or Linux advisable. By its open-source nature, Linux is more customizable, has a large development community, and is easily available. Thus, it was chosen as the development platform.

As real-time 3D graphics are required, C++ appears to be the programming language of choice. As laptop clients should be supported in the future, portability should be kept in mind. If needed DOG should be portable to Windows or Mac OS. Therefore, only function libraries that are also available on these platforms should be used for the development of the software.
Chapter 4

Enabling Rich Multi-User Interaction

Current input technologies are often biased toward pen-based or tangible interaction. Since the principles of TUI and SDG were to be combined in this dissertation, a new input technology had to be developed. It was named InfrActables since its principal of operation used infra-red light and allowed interaction.

4.1 Evaluation of Base Technologies

Based on the requirements presented in the previous chapter, the following functions were chosen for implementation: the simultaneous position and orientation tracking of multiple interaction tools, the identification of these tools using an ID, and the transmission of the devices’ states to the application. Even though these tasks had to be solved wirelessly, many high-level designs still needed to be considered. As there are many standards for wireless RF communication with high bandwidth, the usage of RF technology was well suited. This would allow the transmission of almost unlimited device-state information. Also, the identification of RF streams is part of many standards for RF communication (IEEE 802.11a,b,g,n aka WiFi, IEEE 802.15.1 aka Bluetooth, IEEE 802.15.4 aka ZigBee). Nevertheless, tracking the position and especially the orientation of a device emitting standard RF communication signals is difficult. Many of these standards use spread spectrum technologies and are therefore hard to track. RF signals were expected to interfere with the bodies of the users crowding around the table and with the objects they carry with them. On the other hand, using a camera to track the position and orientation of an object is relatively simple; it is the core functionality of many optical tracking products on the market. The drawback of the optical approach is its limited suitability for data communication when transmitting states and IDs. Usually
this is done through another communication medium like a cable or an RF transceiver. Cables hinder the users too much, which conflicts with the requirements, leaving RF transceivers as the only option. To merge the input devices’ streams containing the information regarding state and position, they must still be identifiable to the system. Identifying devices in a camera-based optical system turns out to be challenging. In the pilot study, multiple optical markers arranged in a unique geometrical fashion were used. However, this hindered the construction of the input devices and was more susceptible to occlusion problems. In an other experiment, the light emitted by the stylus was used to project a unique pattern on the display. Similar as with optical markers, this pattern is then used to identify the stylus. But making such a stylus unambiguously identifiable would require a very restricted and bulky stylus construction. These considerations led to the approach of having input devices emit a light signal, which is acquired by the camera and decoded by the detection system. This solution renders the RF communication channel obsolete, as transmitting the state can also be done in the same way. The communication between the devices and the detection system is time and location multiplexed. Each device’s communication channel is defined by the energy of the emitted light at a specific 2D location at a specific time. As tracking is only required to work on the two-dimensional display, no collisions between the devices’ communication channels should be expected. To make the input technology imperceptible, every device has at least one IR diode to transmit a binary code.

The frame rate of cameras is limited compared to other base technologies like sound and RF. Depending on the interface used, up to 1000 Hz (high-speed camera with embedded processing) or 500 Hz (CameraLink) can be achieved.

Ultrasonic tracking systems usually operate at a signal modulation frequency of 8 kHz [44], whereas RF transceivers operate in an ISM band (such as 430 MHz or 2.5 GHz). As a system’s sampling rate is directly related to the transmission rate that it is capable of [126], the camera will constitute the bottleneck for the InfrActables. Frame rates in the order of 100 Hz are more than sufficient compared to the 10 Hz stated in the requirements. Since encoding the ID and status will require multiple images, the final input rate is expected to be lower. As the camera acquisition rate is critical, a design decision to implement a synchronized system was made, meaning that the parts of the system must synchronize their states. According to the Shannon-Nyquist Sampling Theorem, an unsynchronized system would need a frame rate (sampling rate) twice as high as the communication rate [126].
4.2 Principle of Operation

A bit code is transmitted from the interaction device to the detection system. This code (Figure 4.1) must contain the ID and the state of the input device. The position is detected through the location where the code appears in the camera’s image. If a device lights up its IR-LED, it denotes a logical 1 otherwise a logical 0 is shown (see Figure 4.1).

A code containing only zeros would be unusable as it would never light up the IR-LED and would thus stay invisible to the camera. Using an IR-LED mounted on the interaction device, the bit code is transmitted through the rear-projection display to the camera mounted underneath the interaction surface. An optical IR-filter that is attached to the camera blocks all other light in the visible spectrum. This simplifies the detection in the captured images since no interference with the projected image occurs. The computer now processes the captured images to generate events describing the user’s interaction (e.g., device moved, button pressed). These events are then propagated to the application's user interface. The application can now react to the user's input and draws a sketched line into the framebuffer, giving visual feedback to the user. Figure 4.9 shows the setup in principle.
4.2.1 Bit Code

For the first test of InfrActables, a bit code with a length of five bits (three ID, two state) was used. Theoretically, this allows the use of up to eight interaction devices with four states (see Table 4.1). Using a camera with a frame rate of 60 Hz gives a refresh rate of 12 Hz (frame rate/bit-code length).

Nevertheless, the number of bits used for IDs and states can be adapted to meet an application’s requirements. For some applications many physical handles but no states may be required. The bit code could be adapted to five bits for the IDs and no bits for the state. This would allow the simultaneous use of 32 input devices. Other applications may only be required to support two users and may use rotary switches and sliders on the input devices. In this case, four bits could be used to transmit 16 states and only one bit would be used for the device’s ID. For the given prototype, the bit code can be constant. However, being able to set the configuration of the bit code at launch time or even at run time seems feasible. A bit code consisting only of the bits for ID and state is not as comprehensive compared to other protocols in the physical layer of the OSI model [126]. For example, the serial data transmission standard V.24 additionally defines start, stop, and parity bits. The start-bit is needed in asynchronous communication to denote the beginning of the transmission of the next byte. Originally, it was needed to prepare an electromechanical teletypewriter for writing the next letter [92]. Nowadays it is used to prepare the receive buffer. Without a start bit, the receiver would not know when the transmission has started. The stop bit denotes the end of the data transmission and is used to trigger the storage of the received byte. Parity bits help to detect whether errors happened during transmission. The start bit in V.24 is required as the technology is asynchronous. Synchronous technologies, like the proposed one, usually have a clock signal that allows the receiver to synchronize with the sender. This renders the start and the stop bits unnecessary for our setup. The challenge for the input technology is that the user can, at any time, remove the device from or bring the

Table 4.1: Bit Code for ID and Device State

<table>
<thead>
<tr>
<th>ID Code</th>
<th>Device</th>
<th>State Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Stylus 1</td>
<td>11</td>
<td>Draw</td>
</tr>
<tr>
<td>110</td>
<td>Stylus 2</td>
<td>10</td>
<td>Hover</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>001</td>
<td>TUI Device 3</td>
<td>00</td>
<td>Bring Up Context Menu</td>
</tr>
</tbody>
</table>
4.2. Principle of Operation

device into the tracked area (e.g., the system never knows when the user “plugs in” or “unplugs” the devices from their communication channels, but the receiver should still be able to produce an output that makes sense). As a result, the devices also do not know when the data they send is actually received by the detection system. This issue was solved by using something similar to start and stop bits. Since the frame rate of the camera is the bottle neck, the bit code must be kept as short as possible. Unlike a device on a serial line, the input devices need to constantly send their bit code so that their position can be tracked. Therefore, a device’s communication channel is never idle. The start bit always follows the stop bit.

To deal with unpredictable user interactions, the following was suggested: if a new blinking IR-LED is detected, the system waits for a second period (stop bit) before the ID and the state is determined. This allows the detection system to be certain that the full bit code was received regardless of how the user brought the device into the detection area. Two scenarios illustrate potential problems when a stop bit is not used. In the first scenario, when the user removes the device from the detection area, the bit code is forced to be zero. The state “10” is then be detected as “00”, causing the application to perform an unwanted action. In the second scenario, a corrupted “011” ID is detected as “010” causing the cursor of the pointer with ID “010” to jump back and forth to the place where the device with the ID “011” was removed. Obviously, the stop bit is necessary for reliable operation of the tracking technology. Figure 4.2 shows an example with two bits for the ID: one bit for the state and one for the stop bit. The user moves the device into the detection area, presses it on the surface to draw a line, ends the line by releasing the pressure, and then removes the device from the detection area. As mentioned earlier in this section, the parity bit helps to detect any
corruption that may have occurred during transmission of the data. There are several ways to calculate a parity bit. Odd and even parity methods calculate only a single bit, while cyclic redundancy checks (CRC) allow the computation of more parity bits. The even parity bit is set to “0” or “1” so that the total number of “1’s” in the bit code is even. For example, if the ID and state would be “0110”, then the parity bit would be “0”, and a code of “0010” would result in a parity bit of “1”. If a bit code with an invalid parity bit is received, it can simply be discarded. The functionality for parity bits has been implemented in most parts of the system (PC and microcontroller). As the system ran well enough without a parity bit, they were never used. The parity bit would make the bit code longer and, as a consequence, would slow the detection of the input devices.

![Figure 4.3: Version of the bit code used in the prototype.](image)

Figure 4.3 shows the version of the bit code used experimentally in the prototype. The code contains six bits: three ID bits, two state bits, and one stop bit.

### 4.2.2 Synchronization

To ensure the synchronization between all components (see Figure 4.4), a custom-made electronic device (SyncBox) was built. Its main function is communicating with the processing unit as to which bit should be acquired from the input device. The SyncBox then signals the current bit position to the input devices and triggers the camera’s shutter.
Synchronization Lines

The communication between the SyncBox and the PC was realized using an RS-232 interface, which is easy to implement and is widely supported. Additionally, a serial to USB converter like the FTDI FT232RL allows the connection of the SyncBox to a PC without any RS-232 interface.

The synchronization of the input devices can be accomplished using many base technologies. IR transmission was used because the components are inexpensive and easy to integrate. The bandwidth of IR transmission used in commercial remote controls is sufficient for the prototype. As the synchronization signal emitted by the SyncBox and the data signal of the interaction devices use IR-LEDs, interference between the two signals had to be avoided. The components for the synchronization use a 36 kHz carrier frequency for the signal. Interference is additionally hindered by using one wavelength for the synchronization and another wavelength for the communication lines. IR light at 950 nm is used for the synchronization interrogator and 850 nm for the input devices’ response. As mentioned in Subsection 4.2.1, the input devices use no carrier frequency.
SyncBox Driven Synchronization

The first implementation of the synchronization procedure allowed the SyncBox to control the state of the whole system. This is advantageous as it facilitates the synchronization of the SyncBox with multiple interactive displays in a room. This setup eliminates interference between synchronization signals from the displays. The interaction devices were synchronized by a modulated IR flash that signals the first bit. The devices can use their timers to signal the following bit positions at the correct time. The first setup worked at a frame rate of 60 Hz and required the CPU to process one frame every 16.667 milliseconds. This subjected the system to real-time constraints. The actual task needed 5% of the total CPU power and waiting for a new image to be captured took 20%.

A 2.8 GHz Intel Pentium 4 system with a Linux operating system [47] did not work when the default scheduler settings were used. Up to Kernel version 2.6.23, which was released in September 2007, the minimum time slice an application may run without being preempted by another task is 5 ms, the default is 100 ms and the maximum goes up to 800 ms [21]. Obviously, if another task on the system preempts the detection application and blocks the processor for 100 to 800 ms, the synchronization breaks. Five concurrent tasks, each only using the shortest time slice, will cause the execution of each task to be suspended for 20 ms due to the 5 ms CPU time required for each task. Nevertheless, installing a low-latency kernel from CCRMA [68] and using First In, First Out (FIFO) kernel scheduling [2] solved the issue. A task scheduled in a FIFO manner cannot be preempted by any normal task.

Processing Unit Driven Synchronization

As higher frame rates are not feasible and synchronization was too delicate, another solution was required. One solution to this problem is to use a real-time operating system (RTOS). A “hard RTOS” guarantees precise timings for the execution of a task under any circumstance. A “soft RTOS” tries to provide similar scheduling but does not eliminate the possibility of failure (best effort). Developing the system on a “hard RTOS” would have made software development more difficult. Letting the processing unit control the clock of the whole system, seem to be the better solution. If the OS were to suddenly lag for several milliseconds, the whole tracking system would wait. This principle has also proven to be useful for debugging the tracking software, allowing a step-by-step operation of the system.
Timing and Modulation of Synchronization Signals

In the following, the synchronization scheme is explained in more detail. Figure 4.4 shows the communication direction of the synchronization signal. Figure 4.5 shows the timing and the actual waveform of the signals used. The numbers at the left in the figure correspond to the numbers in the following list:

1. The processing unit triggers the SyncBox through the serial port and communicates which bit is expected in the acquired frame.

2. The SyncBox generates the 36 kHz IR signal to inform the input devices of the current synchronization state.

3. The output of the IR receiver on the input device goes to a low level when receiving the 36kHz IR signal. The signal’s length is used to encode the precise type of the synchronization signal.

4. As the receiver needs some time to tune the 36 kHz carrier signal, the SyncBox must delay execution before generating the trigger signal for the camera.

5. The input device then turns on its LED to signal the transmission of a high level for the current bit. To save valuable battery energy, the LED is only turned on while the camera is integrating the light.

Although the hard real-time constraint regarding the processing unit was resolved, the timing between the components is still required to be strict (see the red rectangle in Figure 4.5). However, this timing can be guaranteed as the devices were either directly or indirectly controlled by a custom programmed micro-processor through synchronized signals. The InfrActable’s principle mode of operation can be described as a synchronous multi-channel simplex communication between the interaction devices and the application’s code. The processing unit acts similar to a reading master in the Inter-Integrated Circuit bus standard (I2C) and generates the clock signal.

Synchronization Signals

Synchronization in the I2C standard uses a start signal, a signal to synchronize the transmission of the next bit, and a stop signal. After the stop signal, the communication medium is idle until the next start signal is sent. Unlike with other communication technologies, the interaction devices will continuously send their bit code. This redundancy is needed to constantly track the position of the input devices. Additionally, the
user might press a button in an area hidden to the camera and move it on the display while continually pressing the button. Therefore, just sending state changes is not enough. For this special case, start, stop, and next bit were not sufficient. A set of synchronization signals was developed that would work with different bit code configurations. Therefore, using a dedicated signal to denote which bit has to be transferred next was inappropriate. Like the synchronization signals used in the I2C standard, the synchronization signals here denote operations to be applied on a “pointer”. The pointer indicates which bit is currently being transmitted by the input device’s IR LED.

Five different signals are needed to signal all possible bit codes (see Table 4.2). Four signals set the pointer to a specific position in the bit code: ID (I), state (F), parity (P), and stop bit (S). A shift operation (N) moves the pointer to the next bit of the transmitted code. ID, state, and parity bit signals allow the transmission of the bit code to be started at multiple positions instead of having a simple start bit. An ID bit signal (I) may directly correlate to the start bit as it denotes the start of the transmission of a full bit code.
4.2. Principle of Operation

Table 4.2: Synchronization Signals Used to Query All Bits of Code

<table>
<thead>
<tr>
<th>SyncSignal</th>
<th>ID</th>
<th>State</th>
<th>Even Parity Bit</th>
<th>Stop Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goto ID (I)</td>
<td>110</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goto Next (N)</td>
<td>110</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goto Next (N)</td>
<td>110</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goto State (F)</td>
<td>110</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goto Next (N)</td>
<td>110</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goto Parity (P)</td>
<td>110</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Goto Stop (S)</td>
<td>110</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The state bit signal (F) is used for an optimized query scheme and will be explained later. A parity bit signal (P) is available if the system requires its use for a particular application. The parity bit was planned as an optional part of the bit code. A stop bit signal (S) allows the tracking of devices with ID and state values of zero. Without a stop bit (S), they would never light up and stay imperceptible to the camera. The stop bit is also used to determine if a device has been removed from the detection area. The next bit signal (N) denotes the same as in other communication standards and synchronizes the transmission of the next bit. The synchronization signals for communicating the full bit code (as described in Subsection 4.2.1) are as follows: I,N,N,F,N,S. This is the complete interrogation of the three ID bits and the two state bits, which are followed by the stop bit.

Optimized Interrogation Scheme

As stated earlier in this chapter, constantly transmitting the same bit code over and over again is wasteful. The ID of a device should, by definition, never change. As long as the device stays within the range of the detection system, it should be possible to simply track the movement of the input device’s LEDs in the acquired images. This tracking is necessary to construct the bit code from the LED’s spots in multiple images. Therefore, knowing the ID of the device is only necessary when a new device appears in the detected area. As a consequence, the normal sequence of synchronization signals is made up of the state bits, parity bits, and the stop bit (F,N,S for the example in Figure 4.3). The full sequence is only needed when new devices need to be identified. The set of synchronization signals presented so far allows the interrogation of any bit code consisting of ID, state, parity, and stop bits. If the bandwidth of the IR-synchronization
is not big enough to denote all signals at the desired rate, the number of signals can be further reduced (see Table 4.3). If parity bits are not used in the bit code (for example, as in Figure 4.3), the stop bit can also be addressed using the next bit signal (N).

A normal sequence would be F,N,N and a full (device identification) sequence would be I,N,N,F,N,N. The downside is that the components lose their flexibility regarding the configuration of the bit code.

**Synchronization Signal Sequence**

As mentioned earlier in this chapter, the processing unit controls the overall synchronization clock of the system. Strict timings are hard to accomplish on a non-real-time OS. As the signals are therefore always slightly delayed compared to the desired time interval (e.g., with a frequency of 60 Hz), the camera’s maximum speed cannot be achieved. As only two sequences (full or state only interrogation) of synchronization signals are needed, the processing unit only needs to communicate the required sequence to the SyncBox. The SyncBox then independently creates signals to capture all frames of the sequence at the rate supported by the camera. If the detection application on the processing unit is not scheduled for a period of time, the acquired frames are saved in the receive buffer of the camera driver. A full buffer will cause the queued images to first be processed before a new sequence is signaled.

### 4.2.3 IDs and LED Topology

The bit code can be adjusted to support different applications using different numbers of devices with different numbers of states. Compared to multi-touch input technologies (see Chapter 2), each IR LED of an interaction device is identified. This allows the use of the geometrical configuration of the IR LEDs with the same IDs to encode more input device states. A trivial case is a device with only one LED and one ID. It can be tracked in only two dimensions. Adding an additional LED using the same ID allows the orientation to be detected. However, the initially detected orientation can be off by 180 degrees (see Figure 4.6). A third LED is used to clarify the orientation. As mentioned above, the state of an input device can consist of more than the states of its buttons.

<table>
<thead>
<tr>
<th>Table 4.3: Reduced Set of Synchronization Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goto first ID bit (I)</td>
</tr>
</tbody>
</table>
If multiple LEDs are used and their configuration can be influenced by the user, these LED positions are also part of the configuration. For example, a device which allows the positions of LEDs to change relative to each other gives the user an additional high-resolution state variable. Using LEDs with different IDs makes this concept even more powerful, as the assignment of the LEDs is unambiguous. As an example, using a device with two LEDs with different IDs would reveal the position, orientation, and add a high-resolution state variable described by the distance between the LEDs.

Multiple LEDs with the same ID can be used to communicate a high-resolution state variable. For this usage of InfrActables, the LEDs must be identifiable through the geometric relations to each other. Figure 4.7 shows a device consisting of eight LEDs arranged in a fixed pattern. The “missing” LED in the lower right corner is used to make the orientation of the device unambiguously detectable. All LEDs use the same ID, but they send individual state information. When the device signals the first state bit to the camera, the state of the LEDs denote the first eight bits of a 16-bit data word (see number in Figure 4.7). When the devices signal the second state bit to the camera, the state of the LEDs communicate the high eight bits. This concept is useful for supporting devices with analog input capabilities like a rotary switch.
4.2.4 Mixing Devices with Different Bit-Code Length

The partitioning of the bit code into state and ID bits is the same for all devices so far. But, for certain applications, it may be favorable to support different numbers of state bits for different input devices:

- A minimal pen may only need one state bit (hover and draw).
- A mouse with an additional button and a scroll wheel needs three state bits to encode the seven states: hover, left button, middle button, right button, scroll down, and scroll up.
- A minimal “brick” TUI [52] does not even need state information at all.
- A TUI sending the state of a rotary potentiometer as its state information may need five state bits to map the rotation in a value range of 0 to 31.

IP protocols’ network classes are a well known concept that could be used for the bit code (see e.g., citeCoulouris:2002). The intention is to split the 32 bit long IP addresses into differently sized networks. IP protocol defines five classes named A to E. The first bits of the IP address are used for assigning the appropriate network class to an address. An IP address starting with a “0” bit belongs to the address Class A, for example. Class A uses seven bits for the network ID and 24 bits for the host ID. IP addresses starting with “110” belong to the Class C network. Class C networks use 21 bits of the IP address for

---

Figure 4.7: Using multiple LEDs for high-resolution state variables.
the network and only 8 bits for the host. Therefore, 128 Class A networks with 16 billion hosts and 2 million Class C networks with 256 hosts are supported. For InfrActables, this would mean that for each individual device class, different lengths for the ID and state can be used. This allows the simultaneous usage of input devices featuring no, little, and many states in the same application. Table 4.4 shows an example with four classes with a bit code of eight bits. This concept affects the optimized update scheme presented in Subsection 4.2.2. Normal interrogation sequences must query all bits of the device with the longest state variable and the stop bit. Full interrogation sequences must query the class code, the ID, the state, and the stop bit.

4.3 Recognition Software

The recognition software uses three abstraction layers (see Figure 4.8) to detect the input device’s position, orientation, state, and identification.

- The first layer (LEDCenter) detects the spots in the image.
- The second layer (LEDPoint) assigns the LEDCenters detected in subsequent images to data objects representing the physical LEDs. LEDPoint further decodes the bit code transmitted by the LED.
- The third layer (LEDDriver) contains the logic for interpreting the state, positions, and orientation of a device from LEDPoints. Based on the ID, each LEDPoint is assigned to a driver which generates device’s specific events. Finally, the events are passed on to the application’s user interface.

Table 4.4: Eight-Bit Code with Different Lengths for ID and State

<table>
<thead>
<tr>
<th></th>
<th>Bit Code</th>
<th>Class Code</th>
<th>Device ID</th>
<th>Device State</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>01101101</td>
<td>0</td>
<td>1</td>
<td>10110</td>
<td>1</td>
</tr>
<tr>
<td>2 devices, 32 states</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class B</td>
<td>10011011</td>
<td>10</td>
<td>01</td>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>4 devices, 8 states</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class C</td>
<td>11011011</td>
<td>110</td>
<td>110</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8 devices, 2 states</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class D</td>
<td>1110001_1</td>
<td>111</td>
<td>0001</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>16 devices, no states</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The recognition software is run using two threads (concurrently executing sub-processes). One thread is responsible for maintaining the targeted recognition rate and communicates with the SyncBox over the serial port (Syncer). It also communicates with the other thread through a queue (SyncQueue). The queue contains SyncState objects containing information on which bit position has been signaled to the SyncBox. The other thread acquires images and processes them in order to generate the events reflecting the users’ interactions.
4.3. Recognition Software

Figure 4.9: Principle setup of the system.
Figure 4.9 gives an overview of how the acquired images are processed. Please note that the steps of the process are numbered. The numbers are in the left margin of the image and will be referred to in the following section.

### 4.3.1 Obtaining the State of the Synchronization

First, the system state corresponding to the image that will be acquired next must be obtained (Step 1 in Figure 4.9). The synchronization thread writes the states of the pending images into a shared queue (SyncQueue). The acquisition and processing thread waits for the next SyncState to be available before acting. The SyncState is used throughout the following steps of the detection process.

### 4.3.2 Acquisition

An IR light at 880-nm wavelength was chosen for the communication from the input devices to the camera. An IR filter is placed in front of the camera’s objective, removing all light in the visible spectrum. Light at longer wavelengths is not registered because the camera sensors are insensitive to light above 1000 nm. Monochrome images are best suited for representing the IR light, which is ideally captured at a single wavelength. The acquired image can be described as a two-dimensional matrix. Each matrix element (representing a pixel) is a number ranging from 0 to 255 (at 8-bit resolution). The values denote the intensity of the light incident on the pixel’s location on the camera chip.

The two-dimensional matrix obtained from the image acquisition contains the raw data of the camera (Step 2 in Figure 4.9). The camera driver which interfaces the actual camera hardware stores the captured matrix in a linear block of memory. The memory block starts at the left top corner of the image and consists of one row after the other.

### 4.3.3 Segmentation

The process of obtaining the compact representation of the interesting image data is known as segmentation [53]. The image is mostly black with some small round spots on it (see Figure 4.10). The interesting data is therefore a list of the detected spots (LED-Centers). Besides the location of the spots, the brightest pixel value and the number of pixels that make up the spots constitute the result of the segmentation (Step 3 in Figure
4.9). The number of pixels a spot is made of allows the removal of false positives (spots not being LEDs). The usage of the brightest pixel is explained later in this Chapter.

Figure 4.10: How the LED appears in the camera’s frame. From left to right: normal view, non-zero pixels colored white with a gain of 0, a gain of 260, and a gain of 680.

Segmentation is performed by processing the image row by row from the top left corner to the lower right corner. A simple threshold value was used to segment the image based on the pixels' brightness. If a pixel has a brightness higher than the threshold, it is assumed to be part of a spot. If the brightness is below the threshold, it is classified as part of the uninteresting background. The data model used for the segmentation models horizontal line segments and regions connecting the line segments. Figure 4.11 shows the segmentation of an image containing three spots with different shapes. The yellow line marks the row being currently processed. Four regions have been detected so far (highlighted in the colors red, green, magenta, and cyan). The white pixels are still unprocessed.

The line segments contain data regarding the start pixel, the end pixel, and the brightest pixel of the segment. The region contains information regarding the number of segments belonging to the region, the total number of pixels in the region, the brightest pixel, and the center of the region.

Starting at the beginning of the memory block, one pixel after the other is processed for the segmentation. If a pixel is above the threshold, a new line segment is created. Subsequent pixels above the threshold extend the line segment until the first background pixel is encountered. When the segment is created and closed, the segments in the previous line are inspected. If a connecting segment is found, the current segment is added to the same region. If no connecting line segment was found in the previous line, a new region is created.

To handle concave shapes, the algorithm must support the merging of regions. For example, when a U-shaped spot is segmented (see Fig. 4.11), the left and the right parts are first detected as individual regions. At the scan line with a connecting seg-
ment, both regions must be merged into a single region. During the segmentation, the center of the spots is calculated using a center of gravity calculation. At the end of the segmentation process, small spots with pixels below a certain pixel count will be filtered out.

![Segmentation of convex and concave shapes.](image)

**Figure 4.11**: Segmentation of convex and concave shapes.

### Segmentation Optimizations

Due to the IR filter in front of the camera, light in the visible spectrum is not acquired. The segmentation algorithm was therefore optimized under the premise that the captured images will contain mostly black pixels and the shapes to be segmented are the spots of IR LEDs. Under normal conditions, the spots can be described as filled circles or ellipses. Using a histogram, the camera settings (aperture, gain, shutter, and brightness) can be calibrated in such a way that the dark pixels have a zero value. If most pixels have a brightness of zero, an optimization can be used. Instead of comparing each monochrome pixel (8-bit wide) to the threshold value, four pixels are compared to zero in one step using a pointer to a 32-bit variable consisting of the four pixels (quadlet). For simplicity of the detection algorithm, the 32 bits should be on the same line. This requires the width of the image to be divisible by four without a remainder. Where \( w \) is the width of the image, the condition \( w \ mod \ 4 = 0 \) must be true. All modes of the camera used fulfilled these conditions. This optimization accelerates detection as fewer comparison operations are needed in the black areas of the frame. The processing time is reduced even further since current processors are optimized for 32-bit operations. Figure 4.10 shows how the AVS Marlin F-033b camera’s gain value affects the number of non-zero pixels in an acquired image (non-zero pixels are drawn in white). The optimization results in a sixfold increase in performance. On a 2 GHz Pentium 4 computer, 0.27 milliseconds were measured instead of 1.62 milliseconds.
When the brightness of the spots created by the IR LED is evaluated, not all pixels in the background can always be calibrated to a zero brightness value (see Section 4.3.4). Therefore, the detection can be further accelerated by ignoring quadlets containing only pixels darker than the threshold. If the threshold value is a measured brightness of 8, a quadlet with the brightness values 1, 0, 2, and 5 can be ignored by the segmentation process. Quadlets with a measured brightness of 1, 5, 9, and 4 should be inspected pixel by pixel as the third pixel is above the threshold. No processor or programming language has an operation to compare the values in the quadlet in a single step. Iterating over the pixels and comparing them individually to the threshold value seems necessary to get a reliable result. Nevertheless, using a bit mask and a bitwise AND operation, some of the quadlets containing only pixel values below the threshold can be discarded.

Table 4.5 shows some thresholds and the corresponding masks needed to test a single pixel value. To test a quadlet, four identical 8-bit masks must be concatenated to a 32-bit mask (see Table 4.6). The bit mask is used to test if the pixel values have a bit set indicating that a value is in the area at or above threshold. To create the mask, two cases of thresholds can be distinguished:

- For normal thresholds, the mask must set the bits from the most significant bit of a threshold to the most significant bit in the pixel (see examples with values 11 or 32 in Table 4.5).

### Table 4.5: Using the AND Bitwise Operation for Discarding Pixels Below a Threshold Value

<table>
<thead>
<tr>
<th>Bit number in pixel ( (n) )</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of bit ( (2^n) )</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Bits of Threshold Decimal 11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Corresponding Mask</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bits of Threshold Decimal 30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Corresponding Mask</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bits of Threshold Decimal 31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Corresponding Mask</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bits of Threshold Decimal 32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Corresponding Mask</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• Thresholds with a binary representation consisting of only “1” digits with preceding “0” digits must be treated differently. In this case, thresholds \((t)\) fulfill the following condition: \(t \text{ AND } (t + 1) = 0\). The binary representation of a value just above the threshold consists of a value with only a single bit set. The bit mask must only set the bits above the most significant bits of the threshold (Example 31 in Table 4.5).

To test if a quadlet can be discarded, the result of the binary \(\text{AND}\) operation is calculated using the bit mask \((m)\) and the quadlet \((q)\). A non-zero result indicates that every pixel value of the quadlet must be inspected, whereas zero indicates that the whole quadlet can be ignored \((q \text{ AND } m = 0)\).

As shown in Table 4.6, this optimization ignores some, but not all, quadlets with pixel values below the threshold. The bit mask for the Threshold 11 tests if the pixel value is greater than 7. Therefore, the algorithm creates false positives for pixels with the values 8, 9, 10, and 11, while 0 to 7 is correctly ignored (see Table 4.6). The algorithm discards 8 of the 12 values which are less or equal than the threshold (66%). The effectiveness mostly depends on the chosen threshold value and the pixel values encountered in the image:

• A threshold with a binary representation consisting only of “1” digits with prepended “0” digits \(2^n - 1\) with \(n \geq 0\) will discard 100% of the quadlets with pixel values below or equal to the threshold. In Table 4.7, Decimal 7 (binary 000111) is given as an example for such a threshold.

• The thresholds just below the values mentioned in the previous item \(2^n - 2\) with \(n > 0\), see Value 31 in Table 4.5) will discard slightly more than 50% of the pixels

<table>
<thead>
<tr>
<th>Mask for Threshold 0000 1011 (Decimal 11)</th>
<th>Pixel n 1111 1000</th>
<th>Pixel n + 1 1111 1000</th>
<th>Pixel n + 2 1111 1000</th>
<th>Pixel n + 3 1111 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Pixel Values 0000 0001 (1)</td>
<td>0000 0000</td>
<td>0000 0101 (5)</td>
<td>0000 0111 (7)</td>
<td>0000 0011 (3)</td>
</tr>
<tr>
<td>Mask (\text{AND}) Pixel Values</td>
<td>0000 0000</td>
<td>0000 0000</td>
<td>0000 0000</td>
<td>0000 0000</td>
</tr>
<tr>
<td>Example Pixel Values 0000 1011 (11)</td>
<td>0000 1000</td>
<td>0000 1000 (8)</td>
<td>0110 0101 (101)</td>
<td>0000 1010 (10)</td>
</tr>
<tr>
<td>Mask (\text{AND}) Pixel Values</td>
<td>0000 1000</td>
<td>0000 1000</td>
<td>0110 0000</td>
<td>0000 1000</td>
</tr>
</tbody>
</table>

Table 4.6: Quadlet Bitmask for Threshold Decimal 11
that can be discarded \((2^{n-1} \text{ pixels})\). Of the inspected pixel values, \(2^{n-1} - 1\) are false positives. This is slightly less than half of the values below the threshold. They are then individually compared to the threshold, although their values are not greater than the threshold.

Table 4.7 shows some examples of the best and the worst thresholds with respect to the discarded quadlets. From the given examples, at least 50% of the quadlets with pixels values below the threshold will be discarded.

<table>
<thead>
<tr>
<th>Threshold Values</th>
<th>Masks</th>
<th>False Positives</th>
<th>Discarded Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 = 00 0111b</td>
<td>11 1000b</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>8 = 00 1000b</td>
<td>11 1000b</td>
<td>1: 1000b</td>
<td>8</td>
</tr>
<tr>
<td>11 = 00 1011b</td>
<td>11 1000b</td>
<td>4: 1000b 1001b 1010b 1011b</td>
<td>8</td>
</tr>
<tr>
<td>14 = 00 1110b</td>
<td>11 1000b</td>
<td>7: 1000b 1001b 1010b 1011b 1100b 1101b 1110b</td>
<td>8</td>
</tr>
</tbody>
</table>

Through the availability of inexpensive 64-bit processors in consumer computers, the two algorithms can double their effectiveness since eight 8-bit pixels can be processed in a single operation using a 64-bit variable.

### 4.3.4 Combining Spots to a Bit Code

As discussed in Section 4.3, the LEDCenters represent the appearance of a physical LED in an acquired frame. The LEDPoint represents the physical LED itself, with regard to position, state of the LED (on or off), ID, and device state (e.g., state of the buttons). The state of the LEDPoint objects must be derived from the observed LEDCenters. Each LEDCenter object represents a detected LED and it must be matched to a LEDPoint object, allowing the LEDPoint to update its state.

This matching is done as follows: first, a motion estimation is used to calculate the expected position of a corresponding LEDCenter. The implemented motion estimation models time, velocity, and acceleration. The matching of LEDCenters to LEDPoints is implemented by iterating over each LEDCenter and comparing it to each LEDPoint using two nested loops (see pseudo code in Table 4.8).

The distance (delta) from the observed point to the estimated point is first computed. This delta is compared to the delta of the previous LEDCenter and LEDPoint assign-
Chapter 4. Enabling Rich Multi-User Interaction

Table 4.8: Pseudo Code for Assigning LEDCenters to LEDPoints

```python
# Comment: Initialize matching
for ledPoint in ledPointList:
    ledPoint.estimatePosition(frameTime)
    ledPoint.resetAssignment()
    ledPoint.assignmentDelta = MAX_ERROR

# Comment: Do matching
for ledCenter in ledCenterList:
    ledCenter.assignmentDelta = MAX_ERROR
    for ledPoint in ledPointList:
        delta = (ledCenter.pos - ledPoint.pos).length()
        if ledPoint.assignmentDelta > delta and ledPoint.assignmentDelta > delta:
            ledPoint.cancelAssignment()  # Cancels bi-directional assignment
            ledCenter.cancelAssignment()  # Cancels bi-directional assignment
            ledPoint.assign(ledCenter)  # Assigns bi-directional
```

ments. If the previous deltas are greater than the current ones, the assignments of the LEDCenter and the LEDPoint will be canceled and a new assignment between the LEDCenter and the LEDPoint will be created (see pseudo code in Table 4.8). Otherwise, no assignment will be changed. This results in a run-time complexity of $n_{LEDCenter} \times n_{LEDPoints}$, which is $O(n^2)$. Since the number of the LEDs used was less than 20, and there was no need to further optimize this relatively inefficient matching algorithm. When all matching is done, the following is evaluated:

- LEDCenter objects that could not be matched to any LEDPoint cause new LEDPoint objects to be created.

- When a LEDPoint object is matched by a LEDCenter object, a bit is set in the bit code of the object.

- When the stop bit is received, the system has reached the end of a detection period. If the stop bit was not set, the LEDPoint is deleted.
4.3. Recognition Software

Detection Thresholds

If the brightness of an LED is approximately equal to the threshold used for the segmentation of the image, sometimes above and sometimes below the threshold, it would constantly appear and disappear, making the detection unreliable. Meaning, in some cases, the brightness of the stop bit would be sufficient for the LEDPoint not to be removed. However, in other cases, the LEDPoint would be removed. The ID and state bits would be randomly set.

This issue was addressed by introducing an additional threshold (see Figure 4.12). The original threshold is implemented in the segmentation process of the image. It decides whether a detected spot should cause a LEDCenter to be created, or whether it should be discarded as noise. The new threshold is used to suppress creation of new LEDPoints in cases of an unmatched LEDCenter and to evaluate the intensity of stop bit. Before a new LEDPoint is created for an unmatched LEDCenter, its brightness must be greater than the new threshold. This prevents new LEDPoints from disappearing before their ID and state can be fully detected. Further, by comparing the stop bit against a higher threshold, existing LEDPoints are discarded before the detection of their state (ID and state) is critically close to the lower threshold.

A similar approach would have been to detect LEDCenters as “zero”, until their brightness is higher than the upper threshold. Subsequent matching LEDCenters would be detected as “one” until the brightness fell below a lower threshold. For the example given in Figure 4.12, the result would be almost the same. Before the LEDPoint is deleted, the state bit would not have been set to “one”. Eventually, the previously mentioned approach was chosen as it is stricter.

To experiment with the thresholds mentioned previously, a test application was implemented (see figure 4.13). Besides testing the effects of various camera settings, it also allows the thresholds to be changed in real time using the keyboard. A histogram shows the logarithmic distribution of the brightness values in the captured frame. Lines show the current thresholds for segmentation. The horizontal lines show the minimum size a spot must have, whereas the vertical lines show the minimum brightness. The LEDPoints known to the system are shown in the top part of the screen. The ID, state (function keys), age (in frames), brightness, size, position, last measured position, velocity, and acceleration are displayed for each LEDPoint. A manual operation mode allows the system to be checked frame by frame.


Figure 4.12: Thresholds used for manipulating LEDPoints. The dotted horizontal lines represent the thresholds. The y-axis shows the maximum pixel value of the LEDCenter.

**Sending an Active Logical Zero**

The following section explains how the brightness of detected LEDs can be used to further improve the robustness of the system. From the observation made in one frame, the next appearance of the LED is calculated using motion prediction. These estimated positions are used to match the observed spots in the next frame. When this matching does not succeed, the detection system will produce erroneous events. In order to improve the robustness, a better motion estimation algorithm could be developed. Nevertheless, every algorithm depends on the measurements to update its prediction model. Therefore, the more observations, the better the result of any prediction will be. As the LEDs on the input devices are turned off to denote a bit value of “zero”, the quality of the motion prediction depends on the actual bit code sent. For example, a device sending a bit code “11” is less likely to get corrupted compared to a device sending a bit code of “00”. While the prediction for the position of the first device can be updated in every frame, the prediction for the later device has to wait for the third
4.3. Recognition Software

Figure 4.13: Application to determine camera settings and thresholds. The camera frame is displayed in the background.

frame (as the stop bit is always “1”). Therefore, the possibility of tracking an LED in every frame would make the motion prediction more robust. The approach chosen was to use different levels of brightness to signal either a “1” bit or a “0” bit.

To implement this on the interaction devices, the devices also turn on their LEDs to signal a “0”, but just for a shorter time compared to the “1” signal. This makes the LED appear darker in the camera’s frame (see system timings in Figure 4.5). For the implementation in the detection software, yet another threshold was required.

- The first threshold in LEDCenter decides whether a spot can be reliably identified as an LED.
- The next threshold in the LEDPoint decides whether a LEDCenter is a logical 0 or a 1.
• The last threshold is used for comparing the intensity with the intensity of the stop bit and deciding whether to create a new LEDPoint instance from unmatched LEDCenters.

**Compensating Intensity Falloff**

Depending on the LED’s position on the projection screen, the LED will appear with a different brightness in the camera’s frame (see figure 4.14). This is caused by:

• the diffusion characteristics of the projection screen;
• the radiation pattern of the LED (angle-dependent intensity);
• the orientation of the pen;
• the varying angle under which the camera “sees” the spot created by the LED on the projection screen;
• and the distance from the LED to the sensor.

Although these factors may seem complex, they can be easily addressed by using a lookup table. Similar as with manual calibration (see Section 4.3.6), the user moves an IR LED all over the screen, while a software program analyzes the brightness of the detected LEDCenters (spots in frames). This data is used to build a two-dimensional lookup table. During normal operation, its values are applied to every detected LEDCenter and the recorded brightness of the LEDCenters detected at the outer parts of the screen is increased to compensate for position. This technique provides good results when the LEDs are mounted perpendicular to the screen. The LEDs on the TUIs are mounted this way. Nevertheless, problems arose through the varying quality of LEDs of the same model. The higher quality LEDs emitted four times more light than the lower quality LEDs (see Section 4.5). Sorting out LEDs with poor quality may be laborious and costly, but it allowed the detected brightness values to be compared to finer grained thresholds. An alternative would have been measuring the brightness of each LED and saving a calibration value on the device that the LED will be mounted on. This value would be a factor in determining how long the LED would be turned on to either signal a logical 1 or 0 bit. In order to adjust the brightness of good LEDs, they would have to be turned on for a shorter period than lower quality LEDs. The first approach of sorting out LEDs with lower quality is good enough for a prototype.
Another challenge of using the brightness of LED spots arises from the user holding the pen at a varying angle. This angle influences how the position of the LED affects the brightness measured by the camera. If the pen is tilted so that it points into the camera, the measured brightness is increased. If the pen is tilted so that it points away from the camera, the camera detects only a darker spot or no spot at all (see Figure 4.15). This makes it impossible to use aggressive calibration lookup tables for pen devices. Otherwise, the brightness of a pen placed at the side of the display pointing in the camera sensor would be amplified too much. This could cause a logical 0 signal to be detected as a logical 1. To prevent this, the pen lookup table must be created by always pointing the pen in the direction of the camera while, at the same time, moving
the pen all over the table.

With regard to the values shown in Figure 4.14, the brightness of a TUI-LED detected on the short side of the screen can be amplified using the simple rule of proportion shown in Equation 4.1. In this case, the factor $\text{brightness}_{\text{normalized}}$ would be 90 and $\text{brightness}_{\text{position}}$ would be 30. The equation should scale all brightness values to the greatest measured brightness ($\text{brightness}_{\text{normalized}}$).

$$\text{brightness}_{\text{corrected}} = \frac{\text{brightness}_{\text{measured}} \times \text{brightness}_{\text{normalized}}}{\text{brightness}_{\text{position}}} \quad (4.1)$$

For the pen at the same position, the $\text{brightness}_{\text{position}}$ factor would be 45, causing a lower amplification (see right part of Figure 4.15).

![Figure 4.15: Relative brightness of an LED tilted toward the positive y-axis (left) or the positive x-axis (right).](image)

**4.3.5 Device-Specific Input Handling**

Based on their IDs, the LEDPoints are assigned to a specific LEDDriver instance. This assignment is part of the configuration of the system. An ID can be assigned to a specific driver depending on what input devices are used. LEDDriver subclasses have been implemented for devices with one, two, or three LEDs (see Section 4.2.3). The LEDDrivers manipulate the class representing the physical input devices (“Pointer”).
This class creates the events like “device moved” or “device state changed” for the application.

### 4.3.6 Calibration

The visual feedback of the application must be calibrated to the input system. Otherwise, the application’s reaction on the display will not appear at the location where the user’s interaction took place. Initially, an automatic calibration was chosen for determining the camera’s and projector’s intrinsic and extrinsic parameters. After removing the IR filter from the camera and adjusting the camera settings, the projected image can be acquired. By projecting a white spot on a black background (Figure 4.16), the existing software (LEDCenter) can be used to generate a lookup table (LUT) for mapping camera coordinates to projector coordinates using a bilinear interpolation. Unfortunately, the IR-Filter considerably affected the optical properties of the acquisition system. Calibration without the filter and later running the system with the filter did not result in satisfactory results. Therefore, a manual calibration had to be used. As the filter now stays mounted during calibration, an IR LED has to be placed on top of the projected white spots (Figure 4.16). The calibration results in a mapping of point in the camera coordinate system to points in the display coordinate system (the projected points). Whereas the projected points are aligned in a regularly spaced grid, the alignment of the detected points in the camera coordinate system show the distortion. A simple bilinear interpolation between the data points did not calibrate the whole area of the interaction table; the calibration grew inaccurate towards the borders of the table. An approach giving better results was taken by defining a stack of transformations and then fitting the parameters of those transformations to the manually measured data points. The algorithm fit each transformation in the stack individually. The following transformations were used: scaling, translation, rotation, and skew.

### 4.4 Electronic Circuits

After initial prototypes on breadboards, five electronic circuit boards were developed. They were used in the SyncBox and in five interaction devices: a stylus, a handle, a ruler, a color can, and an eraser. The boards for the interaction devices are more or less identical. Depending on the input device, the number of IR LEDs and the number of buttons may vary. Some boards can be used for multiple interaction devices, for
example, the board of the stylus is also used for a laser pointer device. The electronic circuit boards and the mechanical design of many input devices evolved over several iterations. Each iteration improved the mechanical and electronic design, making the devices more reliable and user friendly (see Figure 4.17). In order to minimize their size, all devices were built using surface mounted devices (SMD) and printed circuit boards (PCB).

4.4.1 Synchronization Circuit

The SyncBox (Figure 4.18) is driven by an ATmega16 microcontroller (MCU). It generates the 36 kHz synchronization signal for the input devices. This signal is sent to an array of infrared diodes. Eight independent controllable channels are provided; each of them is connected to three diodes in series. Initially, the SyncBox was connected to the PC using a parallel port cable. All lines of the port were galvanically isolated using
optocouplers. Later, it was connected using the serial port. An interface chip (MAX232) allowed the SyncBox to be connected to the PC using a serial cable (RS 232). The third version of the SyncBox then permitted a connection using USB (FTDI FT232 interface chip). The electronic circuit board supported a simple user interface to start up the components of the environment. A small keypad and a standard LCD display (HD 44780 compatible) can be connected and allow navigation in a text-based menu to control the environment components. For this purpose, four relays were built in for switching the individual power lines of the computer, the projector, and additional devices.

![Figure 4.18: SyncBox schema.](image-url)
4.4.2 Interaction Device Circuits

All interaction devices were built on the same hardware platform based on an ATTiny26 MCU (Figure 4.19). An infrared receiver demodulator (SFH-5110-36) is connected to the external interrupt, pulling down the voltage level on reception of the 36-kHz signal. As the receiver’s output is connected to an external interrupt pin of the MCU, the 36-kHz signal causes the start of an interrupt service routine (ISR). Since the ISR is configured to be called at any change of the voltage level, this mechanism can be used to measure the length of the received signal using a timer/counter (TC) of the MCU. In addition, the MCU evaluates the state of the connected keys and changes its bit code accordingly. To save valuable battery energy, the IR LED is turned on only when the camera is integrating light (see Figure 4.5). To realize this well-defined light pulse using the MCU, a TC with pulse width modulation (PWM) was used.

Figure 4.19: Typical device schema.
Because normal CCD cameras are more sensitive at shorter wavelengths, an IR LED was chosen with a wavelength of 850 nm (Stanley DN 304). The radiated intensity varies between different samples of the same type. Stanley sorts individual LEDs into five ranks. Whereas rank “A” only radiates with 15 to 30 mW/sr, rank “E” radiates four times more. The intensity of the radiation directly affects how bright an LED appears in the image. This has an impact on the detection results. Therefore, measuring the diodes before using them is advisable. Only the IR LED with the highest brightness was used to guarantee that it would be clearly visible to the camera under all conditions. Battery life was additionally preserved by connecting the IR diodes of devices with multiple LEDs in series. (see Section 4.2.3).

The status of the interaction devices is communicated to the user using colored LEDs as indicators. One indicator denoted the reception of the synchronization signal (see Section 4.2.2), while the other indicated the battery’s status. A step-up DC-DC converter (MAX1674) allows the use of batteries ranging from 1.5 Volt (V) to 5 V and is also responsible for communicating a low-battery status to the MCU. The range of supported input voltages allowed batteries from a wide range to be used, allowing a trade-off between battery life and physical size. The power dissipation of an input device varies between approximately 0.05 Watt (W) and 0.5 W during operations. The IR LED consumes 0.4 W, a status LED around 0.05 W, whereas the MCU at 1 MHz consumes only 0.006 W. The efficiency of the DC-DC converter is 90% in combination with a battery pack providing 2.4 V. In order to cover this power dissipation, different kinds of batteries can be used. The smallest group is the 3 V lithium batteries used for digital cameras. For ecological and economical reasons, only rechargeable batteries were used. TUIs use NiMH AAA rechargeable batteries providing 900 mAh while the stylus uses UM5 (Lady, N) NiCd batteries with only 150 mAh. Whereas the TUIs last for several hours, the pens’ batteries have to be replaced after approximately two hours.

### 4.5 Realized Interaction Devices

The interaction devices built are presented in the following sections. The description concentrates on the hardware (physical and electrical aspects). The implemented interactions will be explained later.
4.5.1 Stylus

The normal stylus is the most important interaction device in the computer-supported collaborative environment, as users interact most of the time with it. In order to achieve a comfortable size and weight, a high integration was required and therefore PCB with SMDs were a must. Three micro switches were integrated. An IR LED is mounted in the tip to track the movements of the stylus. The IR LED sits on top of a small plastic cylinder with two notches for the LED’s legs. The cylinder is movable in the axial direction of the pen and its bottom side touches a microswitch. When the pen is pushed on the display to draw a line, the IR LED pushes on the cylinder, and the cylinder forwards the pressure to the micro switch. A rocker switch can be alternately used to activate the two other micro switches. It can be operated by the user’s forefinger or thumb.

The pen directly glides with the IR LED on the display surface. The LED does not scratch the hardened glass of the projection surface. Mounting the projection foil on the back of the glass avoids any damage from scratching pens. Because the glass is in between the projection foil and the pen, the user’s interaction is separated some millimeters from the display. This might irritate the user when observing the pen during sketching. Depending on the angle of observation, the lines drawn by the pen appear like they are next to the pen (parallax error).

Continuous synchronization of the pen in any position is ensured by three receivers. They were mounted on the tip with 120 degrees between their axes.

![Stylus and the PCBs of two styli.](image)

Figure 4.20: Stylus and the PCBs of two styli.
4.5.2 Laser Stylus

In addition, a laser stylus was developed to allow interactions from a distance (middle of Figure 4.17). Instead of an IR LED, it contains an infrared laser diode in the tip. It only has one receiver looking facing the display at which the pen is pointed. As the IR laser is imperceptible, the pupillary reflex is absent. Therefore, users must wear safety goggles to prevent harm from a beam hitting their eyes. Usability is further hindered as total reflection on the hardened glass of the projection display occurs under a relatively wide angle. This causes the pen to be undetectable when used under an obtuse angle. Originally, the laser stylus was developed to move virtual content between screens in a room (see Section 3.3.7). For example, the user could point to a virtual paper lying on the horizontal screen, press a button, and drag it to a vertical display to cluster it with papers containing similar ideas. Although the interaction seemed compelling, due to the restricted usability, the device and the interaction were abandoned.

4.5.3 Eraser

In order make the intuitive interaction complete, an eraser was realized. The head of the eraser is loosely connected to the main body and can be bent in different directions. Four micro switches are in the gap between the head and the main body. Each micro switch activates a corresponding IR-LED which is integrated in the front corner of the eraser’s head. Thus, one, two, or four LEDs can be activated depending on how the eraser is pressed onto the interaction space’s surface. The geometrical pattern that is generated by the LEDs can be detected by the camera and from this, the size of the virtual eraser is determined. If only one LED is visible, the erasing area is very small (for erasing single lines or letters). The erasing size is maximal if all four IR-LEDs are visible to the camera. Figure 4.21 shows some prototypes of the eraser.

4.5.4 Color Can

A bottomless color can was built to change the color of the styli. The device has a circular shape. This hindered the layout of the printed circuit board (PCB). Four micro switches at the bottom side of the ring allow the user to interact by pushing the TUI down. Two buttons at the side offer additional interaction capabilities. The color can uses three IR diodes: two on the middle axis of the device and one in the ring slightly off the middle (see Figure 4.22. This allows the tracking software to unambiguously
detect the device’s orientation. Therefore, the vectors between the spots of the three diodes are sorted by length. The longest vector must be on the middle axis and it allows the calculation of the center and the orientation angle of the color can.

4.5.5 Ruler

For measuring, a ruler device was developed. As shown in Figure 4.23, it consists of a transparent Plexiglas ruler, a slider, and a box. The slider allows the user to manually
measure virtual objects on the display. The box contains the electronics and features buttons to switch the interaction mode of the device. The buttons were mounted on the top, front, and outer side of the ruler. Like the buttons on the other devices, the user can easily spot them as they consist of large aluminum parts (top button in Figure 4.23).

![Figure 4.23: Final version of the ruler.](image)

Three LEDs are used to unambiguously track the position and orientation of the ruler, plus the relative position of the slider (see assigned letters in Figure 4.25). LED A is mounted on the slider and is connected to the electronic circuit board through a sliding contact and a ball bearing rolling on a metal rod (see Figure 4.24). LED B is in the box, aligned with LED A on the slider so that the resulting vector is parallel to the Plexiglas ruler. LED C is used to unambiguously specify the orientation. The slider movement is restricted so that the conditions in Equation 4.2 are fulfilled.

\[
|\vec{A} - \vec{B}| > |\vec{A} - \vec{C}| > |\vec{B} - \vec{C}|
\]  

(4.2)

Since the electrical resistance of the sliding contacts is not completely constant, the ruler device uses two power channels in parallel to drive the IR LEDs. One drives LED A, while the other drives LED B and LED C in series. Therefore, the power consumption is higher than for the other devices. Since the box offered enough space, four AAA
batteries were used as the power source. Lengths of up to 50 cm could be measured with the first version of the ruler. The device seemed to be too long, as its interaction was hindered by other devices on the projection surface. Therefore, the second version of the ruler was only 30-cm long.

Figure 4.25: Schematic showing constraint movement of the slider.
4.5.6 Handle

Like the pen, the handle (Figure 4.26) serves as a personal device. It was conceived for grasping and moving around digital content on the display. Similar to the color can, three LEDs are used to unambiguously detect the position and orientation. It also features three large buttons made out of aluminum: one button is on top of the device, and the other two buttons are on the sides. Pressing either of the side buttons will cause the same state change. A rotary encoder (see Figure 4.19) can be optionally mounted on the PCB. It allows the ID of the device to be changed without reprogramming the firmware. This gives more flexibility when choosing which set of tangibles will be used in an application. The handle’s size and weight changed considerably over the three development iterations. The housing of the initial prototype did not high enough from the projection display to provide a comfortable grasp. The next version was higher (see Figure 4.26 on the left). But this time, compared to the width of the user’s palm, it was not long enough. Moreover, the housing constructed of bent sheet metal rattled and did not feel solid. The final version of the handle gives the user the feeling of holding a solid and sturdy device. Additionally, the device’s buttons were designed to provide a well-defined pressure point. This proved to be especially important as the user can lift the device off the projection display without accidentally pressing the side buttons.

As noted in the sections before, all interaction devices were constantly improved over time. Most changes improved ergonomics and reliability of operation. Some changes added more functionality to the devices or facilitated new interaction. As mentioned at the beginning of this Section, only the hardware has been presented so far. The actual interactions realized with these devices are described in Chapter 6.
4.6 Cameras

Various cameras were evaluated for application in the prototype. The requirements for a camera are a frame rate higher than 60 Hz, small latency, providing an input capability for getting synchronized, capturing images with a Video Graphics Array (VGA: 640x480) resolution or greater, and a high IR sensitivity.

Active pixel (APS or CMOS) and charge-coupled device (CCD) sensor types are commonly used in digital cameras. Compared to CCD, APS technology uses a thinner photosensitive layer. Therefore, APSs are only a little sensitive to light with a wavelength greater than 650 nm [60]. APSs are therefore less suitable for the setup, as IR LEDs operate in the near infrared spectrum (750 to 1400 nm). As an APS integrates more signal processing features into the circuits, only 30 percent of the surface of an APS is actually photosensitive. Therefore, APSs are in general less photosensitive than a CCD sensor with the same size. CCD cameras with a peak sensitivity in the near infrared spectrum (Hitachi KP-F2A) seem promising, but unfortunately they only offer average frame rates (30Hz).

![Figure 4.27: Relative sensitivity of color sensor (AVT Marlin F-033C Camera).](image)

Source: AVT.

The usage of a monochrome camera is highly advisable as most color cameras use
4.6. Cameras

only one sensor to detect the color components. A color filter mosaic pattern (Bayer filter) applied to the sensor distributes red (25%), green (50%), and blue (25%) pixels over the sensor. Figure 4.27 shows the color sensor’s relative sensitivity to light of different wavelengths. As green and blue pixels are almost insensitive to IR light, only red pixels are usable in the IR spectrum. This quarters the resolution of a color camera in the system. Cameras using separate sensors with an individual filter for each color component (three-chip camera) provide full resolution. As only the chip with the red filter receives IR light and the other two sensors are idle, their usage seems inappropriate. Figure 4.28 shows the sensor’s sensitivity with regard to the wavelength of the detected light. As noted in Subsection 4.4.2, the IR LEDs used emit light at 850 nm. The sensor of the AVT Marlin F-033C camera is five times less sensitive at this wavelength compared to light at 500 nm (cyan color).

![Figure 4.28: Relative sensitivity of monochrome sensor (AVT Marlin F-033B Camera). Source: AVT.](image)

The VGA resolution is sufficient for tracking the position of the input devices on the interactive surface. The projector features a resolution in the same order of magnitude (1024 by 768 pixels). As only white spots have to be segmented in the otherwise black frame, various algorithms can be used to achieve subpixel accuracy when locating the spots’ positions. The actual algorithm implemented in the presented system (see Subsection 4.3.3) emphasizes speed over accuracy and uses a center of gravity calculation. It pro-
vided good results for the setup. Commercial optical tracking systems like the Qualisys ProReflex claim to achieve a subpixel accuracy of 64 units per pixel. An image sensor resolution of 658x500 pixels results in an effective tracking resolution of 20000x15000 subpixels [3]. This indicates that a VGA camera is suitable for combining with a full HDTV projector featuring an HD 1080p (1920x1080 pixels) resolution.

Typical consumer cameras usually have a frame rate of 30 Hz. Their lenses cannot be changed to another focal length and no synchronization is available. Cameras for industrial computer vision (CV) have frame rates between 73 Hz and 500 Hz; high-quality objectives can be used, and a trigger input is available. It allows the synchronization of the shutter of the camera with other components of the tracking system. CV cameras are available with different interface systems. IEEE-1394 (Firewire™) and USB™2.0 interfaces cost around 2000 CHF. As standard computers feature these connectors, no additional interface card is required. Cameras with the CameraLink™interface offer higher frame rates and image resolutions, but they are more expensive and need an additional interface card. The main advantage of IEEE-1394 is that the camera complies to an industry standard (IIDC, sometimes also named DCAM [71]). This standard allows changing between camera models from different vendors supporting the IIDC specification without any need to install a new driver or to change the software using the camera. IIDC is supported by many generic and specialized application programming interfaces (API) on Windows, Mac OS X, and Linux. Unlike generic APIs for consumer cameras like DirectShow, specialized IIDC APIs like [35] or [6] clearly specify the access to all camera features encountered in CV cameras.

In many applications, light in the IR spectrum is not desired. A cut-off filter is often mounted between the objective and the sensor and eliminates all light above 650 nm. For use in the setup, such a filter has to be removed.

For the reasons discussed in this subsection, monochrome IEEE-1394 cameras for computer vision were used in the setup. The Marlin F-033C features 73 Hz and provides a resolution slightly above VGA (656x492 pixel). The Point Grey Marlin Dragonfly Express supports of the newer IEEE-1394b specification, it features a higher bandwidth (800 instead of 400 megabits per second). This allows the Dragonfly Express to acquire images with VGA resolution at 200 Hz.
4.7 Projection Screens

An ideal projection screen diffuses the light projected on it homogeneously in the spectators' direction. This guarantees a constant brightness of the projection under any viewing angle. Such a homogeneous diffusion would also be ideal for the IR light emitted by the pen. Regardless of the pen's tilt, the spot would always have the same brightness. Of course such an ideal projection screen does not exist. Further, in the case of the presented system, IR light emitted from the diode traverses the projection screen in the direction of the projector. The direction of traversal and the wavelength of the light differ from the screen's normal use. It is unlikely to find a product designed for this case. Therefore, the projection screen's diffusion characteristics for infrared light were measured. The same IR LEDs were used as with the interaction devices (Stanley DN 304, 850 nm). For quantitative measurements, a calibrated luminance meter could be used. Since the product at hand (Minolta LS-100) measures light in the visible spectrum (450 to 650 nm), it was not suitable for measuring IR light. Qualitative assessments were sufficient for evaluating the influence of different projection screens, therefore the test series were conducted using an AVT Marlin F033b CCD camera. For comparable results, all camera settings were manually controlled (e.g., shutter, gain).

Two kinds of measurements were performed. First, the relation of the LED's position on the projection screen to the measured brightness was recorded for one projection material. During these measurements the LEDs were kept perpendicular to the screen. This test series used the application which was developed to create the lookup table for brightness falloff compensation. The results of these measurements were presented in Subsection 4.3.4. In the second test series, only pen tilting effects were measured. The setup consisted of mounting the projection screen to be measured at a distance of 100 cm in front of the camera. The camera was mounted so that the lens's axis was perpendicular to the screen. The LED touched the screen and appeared in the center of the camera's frame. This minimized any lens and sensor effects that might occur if light were registered off the lens's axis. The tilt of the LED was varied in steps of 10 degrees, starting from 20 degrees and going up to 90 degrees (perpendicular to the screen). Some products were shipped as projection foils, while others were projection screens ready to be used. This required choosing a carrier to test the foils. A test was carried out with laminated sheet glass (LSG), coated on both sides to reduce reflections. As expected, the LSG showed no diffusion properties. LSG's results were identical to the results with normally coated glass (non-LSG 3 mm) and to the results with no projection material at all. Therefore, LSG was a good carrier for the projection foils. As LSG showed no diffusion characteristics, the measurements labeled with
LSG in the following figures shows the radiation characteristics of the LED. Figure 4.29 shows the results of the measurements. The frames were acquired using 8 bits per pixel, allowing brightness values in the range from 0 to 255. The camera settings were adjusted so that all materials but the LSG had their measurements in the range. In LSG’s measurements between 70 and 90 degrees, the IR light oversaturated the sensor. As LSD’s measurements are only provided as reference, the goal was to compare the actual projection materials and these truncated values were taken into account.

Some projection screens (like the 798-1/2) let the LED appear very bright at the 90 degree position, but quickly appear darker at smaller angles. All projection materials let the LED appear darker than 50 brightness values at an angle of 50 degrees and smaller. The main difference is that some projection material has a steeper or flatter slope between 50 and 90 degrees.

![Figure 4.29: Diffusion characteristics of projection screens at 850 nm (infrared).](image)

The measurements of the ideal projection material mentioned in the previous paragraphs would appear as a horizontal line in the diagram. Under any angle, the LED would appear with the same brightness in the camera’s sensor. The higher this bright-
ness value, the better the projection screen would be. Obviously, no such material was found in the evaluation. Figure 4.30 shows the normalized data set of each screen. The Blackscreen RPF diffuses the IR Light of the IR-LED the best. The intensity measured at 20 degrees is the highest compared to the intensity at 90 degrees. Contrast Foil and Light Foil were rated second. As shown on Figure 4.29, they are overall brighter than the Blackscreen RPF, but the brightness curve is less linear.

Figure 4.30: Infrared diffusion characteristics of projection screens (each curve is normalized).

The test series showed that the optical characteristics of different projection screens vary considerably for infrared light. Therefore, the Blackscreen RPF is recommended.

4.8 Resulting Detection Rate

In the following, the performance of the presented system with the cameras selected in Section 4.6 is discussed. The performance indicators are compared to what was
required in Chapter 3. In Subsection 4.2.2, the full set of synchronization signals was presented as well as the sequence used to interrogate the three ID bits and the two state bits. This yields a detection rate of 12.17 Hz for a 73-Hz camera and 33.3 Hz for a 200-Hz camera (see Equation 4.3).

\[
rate_{\text{normal}} = \frac{fps}{n_{ID} + n_{state} + n_{parity} + n_{stop}} \tag{4.3}
\]

\[
fps = \text{frames per second} \left(60 \frac{1}{\text{second}} \mid 200 \frac{1}{\text{second}}\right)
\]

\[
n_{ID} = \text{number of ID bits (3)}
\]

\[
n_{state} = \text{number of state bits (2)}
\]

\[
n_{parity} = \text{number of parity bits (0)}
\]

\[
n_{stop} = \text{number of stop bits (1)}
\]

The update rate of the 73-Hz camera is relatively slow for sketching and writing. It barely meets the 10 Hz specified as the bottom line. The optimized interrogation scheme described in Subsection 4.2.2 increased the update rate to 24.3 Hz and 66.67 Hz, respectively (see Equation 4.4). Therefore, the achieved update rate of the system exceeds the requirement.

\[
rate_{\text{optimized}} = \frac{fps}{n_{state} + n_{parity} + n_{stop}} \tag{4.4}
\]

Chapter 3 specified a maximal latency of 120 ms. The latency describes the time it takes the system to react after a user’s action. For example, the length of time it takes for the system to start to draw a line after the user presses the pen on the display. For button events, this latency is generally defined by the time it takes to acquire \(n_{state} + n_{parity} + n_{stop}\) frames plus the time for buffering and processing the frames (see Equation 4.5). To determine this additional delay, the round-trip time of a frame was measured. The measurement was started with the processing unit communicating to the SyncBox and ended when processing of the received frame was completed. Around 20 ms were measured for the round-trip time. The actual perceived additional delay is shorter since under normal circumstances, only the time from when the synchronization signal is received on the interaction device to the time when processing ends matters to the user. Nevertheless, 20 ms indicates the order of magnitude.

\[
latency_{\text{normal}} = \frac{n_f + n_p + n_s}{fps} + latency_{\text{additional}} \tag{4.5}
\]
If all ID bits are queried because a new device has been brought into the detection area (see Subsection 4.2.2), the latency will increase as now also the $n_{ID}$ bits are interrogated. The user will experience the worst latency of the system. The user’s interaction will be delayed for the six frames plus the additional latency (see Equation 4.6). To summarize the latencies for the slower camera (73 Hz):

- During normal operations, the latency is below 61 ms.
- In the worst case, latencies of 102 ms can occur.

Therefore, the latency stays below the required 120 ms (see Chapter 3).

Unlike state or button events, events indicating that an input device has moved can be generated whenever a device was observed in a frame. By sending an active logical 0 (see Section 4.3.4), the position of the input devices can be updated in every frame. Hence, in the best case, the update rate of move events is generated at the full rate of the camera (73 Hz for the slower camera). As discussed in Section 4.3.4, the active logical 0 does not always work for the pen. Therefore, in the worst case, the rate of move events is one-third of the camera’s frame rate. As a result, the state with number three (binary 11) was used to denote when the stylus is pressed on the display surface, allowing the user to sketch under any condition at the full frame rate of the camera. The latency of move events is far below the required 120 ms. Using the 73 Hz camera results in a latency of 34 ms for state number three (binary 11) and 61 ms for state number zero (binary 0).
Chapter 5

Interfacing the Input Devices

This chapter presents the software for interfacing input devices (see Chapter 3) is presented. As the software was not only used for the client application presented in Chapter 6, it is presented as an independent component.

The application programming interface (API) was named “Tracking” since it allows the user’s interactions to be tracked. It enables a client of the API to receive events from different kinds of input devices. The “Tracking” system is loaded when the application starts. The configuration is done by an XML file [22]. From this point on, the client code can query the API for the available tracking devices. The client code can then subscribe to certain events from certain input devices.

The whole system can be divided into three parts (see Figure 5.1). Part one generates the actual events and consists of drivers for different kind of input devices. Part Two is the EventDispatcher, which forwards the events to the objects that have subscribed to the event type and forms the interface between the drivers and the client code. Part Three is a set of classes and interfaces a software developer can use to receive events.

5.1 Auxiliaries

First the auxiliaries are presented. These objects are used in many other parts of the system.
5.1.1 Event Address and Filter

The address class is used to describe the precise type of an event. It is attached to every event and is also used by the client code to specify for which events it should be registered. The address class consists of the type of the driver, the device’s number, the number of the pointer on the specified device, and the type of the event’s action. See Table 5.1 for examples of values. Additionally, wildcards can be specified for every field. They allow the client code to register for multiple types of events.

The filter’s fields (driver, device, pointer, and action) were chosen to support a wide...
range of different kinds of input technologies. The driver field allows the EventDispatcher to distinguish which events were caused by which kind of system. Obviously, every input technology hardware is connected to an input port of the computer running the Tracking software. On Linux, device files can be used to denote which device of the same type is meant. PS/2 mice appear as input device (/dev/input/mouse0), the Polaris Tracking system is connected to a serial port (/dev/ttyS0), and tracking systems using IEEE-1394 cameras (InfrActables, ARToolkit) use a video device like /dev/video1394/0. Therefore, the device number specifies the last character of the device file used. Since the regular mouse and keyboard of the OS are available through the OpenGL Utility Toolkit driver, only one pointer is available. The only configuration possible is to enable or disable them.

Some input technologies, like VR equipment or the technologies developed in this dissertation (see Chapter 4) support multiple pointers. The pointer field in the event allows the application developer to check which input device caused the event. Additionally, it allows the client to register for events from a single input device only (for example, from a single stylus).

The event's action field contains what triggered the event. Beside the usual actions like button up/down, key down, passive and active move (dragging), actions are also provided for the event when a device appears or leaves the range of the tracking systems. The event address is implemented as a bit mask. The actual bits for each field are declared as enumerations. Wildcards were defined addressing multiple bits. For example, \texttt{BUTTON\_ANY} (0x03 = 0x02 0x01) denotes button up (0x02) and button down (0x01). After registering for events from \texttt{POINTER\_ANY}, events from all pointers will be received. Wildcards are particularly useful when client codes register to receive particular types of input events. Section 5.4 explains this in more detail.

5.1.2 Pointer

The individual input devices are modeled with the Pointer class. Instances of this class are assigned to a tracking device instance. A driver generates instances of this class and changes the state accordingly to the result of the tracking method. Examples are setting the position, button or keys. When the driver is done with changing the state of the pointer, he calls a submit function. This will cause the proper event to be generated and submitted to the EventDispatcher.

The Pointer class addresses certain calibration issues. For each pointer, a four by four matrix can be specified, which is used to transform raw coordinate data. Additionally,
a three-dimensional vector can be defined to model the offset from the origin of the
device’s coordinate system to the point representing the actual pointer (see Figure 5.2).
This feature was required for the second pilot study using the Polaris tracking system.
The Pointer class also provides a field indicating its type, which can be a 2D pointer, 3D
pointer, 2D TUI, or a 3D camera for the ARToolkit (ART) [75]. Furthermore, a name field
allows each pointer to be named individually. The name can be set in the configuration
file (see Section 5.2). With the name field, the client code can distinguish between
pointers of the same type.

![Figure 5.2: Calibration possibility of the Pointer class](image)

### 5.1.3 Event

The event class contains the actual information describing the input device’s change of
state. The class must be flexible enough to support a wide range of applications and
tracking technologies. As mentioned in Section 5.1.1, the event class contains an ad-
dress object pointing to its source. The normal combination of a keyboard and a mouse
requires mouse button, mouse move (2D), and keyboard events. Six-dimensional track-
ing systems need three dimensions to describe the position and three more dimensions
to describe the orientation of the device. The change in orientation also causes a mo-
tion event. The integration of the ART system introduced another representation of the
event’s data. In ART, a set of markers can be used to define the application’s coordinate
system. ART calculates the position of the camera from the markers’ appearance in the
camera’s frames. ART describes the result of this process as a four by four matrix. It
transforms the coordinates from the world coordinate system to the camera coordinate
system. The augmented reality application can load this matrix on the transformation stack of the graphic subsystem before rendering the scene.

Due to the constant development of new input devices with the InfrActable system, the data transferred by events was subject to change. A system was therefore needed which allowed various kinds of tracking data to be communicated to the application.

Instances of the event class are created several times per second. Allocating new memory for small objects using standard facilities (new operator) is inefficient. Custom allocation strategies and the reuse of objects deleted by the application is favorable. A single event class simplifies this and therefore allows the system's performance to be improved. See [87] and [4] for examples on how to optimize small object allocation.

To satisfy the need for 2D to 6D interaction devices and events from ART, a four by four matrix is used to describe position and orientation. As with a normal transformation matrix, the cells can be used to describe linear transformations (e.g., translation, rotation, scaling, and perspective distortion). Nevertheless, for experimentation with new devices, the cells can be used in any format as long as the client code is aware of this irregularity. This solution favors flexibility and speed of implementation over type safety. Once the set of events generated by the input devices is consolidated, the event class can be adopted.

## 5.2 Configuration

A configuration file can be used to define what input devices should be used. Additionally, configuration parameters can be set for driver classes. An application using the Tracking library can be started with different input devices' configuration files. One configuration file is used for software development, whereas a different configuration file is used for each collaborative setup. For software development, a PS/2 mouse plus the operating system's mouse and keyboard are often sufficient.

XML [22] was chosen as the format for the configuration file. The encoded representation of XML (markup) is reasonably human-legible. There are many libraries available to parse XML data. The “Xerces” XML parser [54] developed by the Apache Foundation was used in the setup. It is a mature product and is available for many platforms (Linux, Windows, and others).

Besides the root tag “Tracking”, each device is described with its own tag. Subtags and attributes allow many details of the driver's settings to be configured. For the input technology developed in this dissertation (InfrActables), the thresholds, camera config-
uration file, calibration files, and many other settings must be defined. The function bool TRK::setupFromXML(string configfile) allows the application programmer to load them from a configuration file.

5.3 Drivers and Devices

The drivers and devices mentioned in the Event Address Section were implemented as classes and instances of these classes. For example, an instance of the class DrvPS2Mouse listens to the serial port (/dev/ttyS2) and represents the connected mouse.

Drivers manage the source of the events. In the case of the Polaris and PS2Mouse driver, the source of the events is an external device. The class manages the serial connection and converts the device’s protocol into changes of the associated pointer class (see section 5.1.2). The InfrActables and the ART drivers implement the actual tracking functionality into the class. The external devices just provide the raw data to be analyzed.

OpenGL Utilities Toolkit (GLUT) is a portable library taking care of low-level issues for graphical applications using OpenGL. Besides other functionalities, it connects to the display server and receives the operating system’s mouse and keyboard events. The GLUT driver makes these events available within the Tracking library. In the following, the concept of this wrapper is explained. GLUT allows the client code to register callback functions for various events like mouse moved or mouse entered the application’s window. The GLUT device provides these callback functions and registers them with the GLUT system. When a callback happens, the GLUT device’s function translates the call into state changes of the associated pointer class. The pointer class will emit an event if needed.

All driver classes share a common interface, which allows the operation of the driver to be started and stopped. Further, the associated pointers can be queried. Additionally, a four by four transformation matrix can be specified that will be applied to the events generated by the devices’ pointers.

Many drivers run in their own thread. A base class for threaded drivers provides the facilities used by all threaded driver classes. It handles the starting up, shutting down, and time outs of the driver’s thread. An abstract protected function provides a “hook” to implement the actual tracking code. The function is then called by the base class to maintain a constant tracking rate.
5.4 EventDispatcher

The EventDispatcher defines the interface between the client code and the drivers doing the actual tracking work. The client code can query for pointers of a certain type or with a certain name. A list with matching pointer instances is returned to the client code and simplifies a subsequent registration call.

When a driver detects that the tracked input devices have changed their states, it modifies the associated pointer instance and calls its function to create the corresponding events. These events are sent to the EventDispatcher and stored in a first-in-first-out (FIFO) queue. The EventDispatcher dequeues the events and uses the events’ addresses and its own registration data structure to send the events to the proper client.

Due to its purpose as a central distributor of input events, only one instance of the EventDispatcher may exist in an application. Therefore it was implemented following the Singleton design pattern [136].

Normally, the EventDispatcher has a dedicated thread to dispatch incoming events. When the EventDispatcher interacts with a system that uses thread specific storage (memory), additional mechanisms are inevitable. To easily circumvent these problems, the EventDispatcher can be used with or without a dedicated thread. In the unthreaded case, a client code must repeatedly call a function on the EventDispatcher to dispatch the waiting events.

5.5 Listeners

As noted before, a client code interested in state change events from the input devices registers with the EventDispatcher. This principle is related to the Observer Design Pattern described in [136]. The implementation favors the push model of the observer pattern, as the event contains all data regarding the input device’s state. Depending on the event type, a different method of the Listener is called. These functions are defined in the abstract Listener class. A client object must implement all functions of the Listener class. Similar to Java’s MouseAdapter, the ListenerAdapter class provides default implementations for all functions. Therefore, a subclass must only implement the functions needed.

Listeners that intend to do extensive calculation in the callback function block the EventDispatcher. Meanwhile, no other event can be dispatched and the EventDispatcher’s queue continues to fill. The Queue class was designed for such clients; it simply queues
incoming events in the client code up. The client code can then check its queue at its own pace.

5.6 Interfaces to Other Systems

So far, the classes in the previous section can only be used if they run in the same process as the Tracking system. A further restriction is that the client code must be developed in C++. To resolve these contraints, additional interfaces were developed. They are presented in the following.

5.6.1 Interface to Macromedia Flash

Tracking was initially developed within the CRION project [76] for a Single Display Groupware setup with remote collaboration capabilities. The actual SDG application (Figure 5.3) was developed by Nils Birkeland from NTB using Macromedia's Flash Communication Server (FSC) [84]. In order to pass the events from the Tracking system to the Flash application, an ActionScript XMLSocket was used. On Tracking's side, a TCP socket server waits for incoming connections from Flash Applets and converts the events into XML tags. The TCP sockets’ “Type of Service” flags can be tuned in order to reduce the latency of the communication. But a small additional delay (some milliseconds) is inevitable. However, for the use cases in the CRION project, it prooved to be fast enough.

Figure 5.3: SDG application created with Macromedia Flash
5.6. Interfaces to Other Systems

5.6.2 Interface to X11

In order to apply the InfrActable input technology to Desktop and GUI applications, an interface to the X11 server was developed to send the events from the Tracking library to the display server (Unix systems). The application allows the styli to be used with any X11 application. Nevertheless, if more than one stylus are present, the cursor of X11 will jump to the stylus that sent the last event. This is makes it unusable for multiple users, but nicely demonstrates the limitations of current operation system’s GUIs. The X11 extension “XTest” [55] was used to implement this functionality in a reasonable time. XTest was developed for testing X11 servers using synthetic input only. The XTestFakeButtonEvent and the XTestFakeMotionEvent were used to inject the events from the Tracking library. As XTest is an optional extension, it is not installed by default on Unix systems. The XTest performed well, and the experiment showed that it is also compelling to interact with regular GUI application on a large display. Nevertheless, for a real product a XInput driver needs to be developed as the events from XTest are marked as fake and some applications may reject them.
Chapter 6

DOG: Distributed Object-Oriented Groupware

In the following, the client application that was developed for the presented environment is described. The basic requirements for the client application were presented in Section 3.6. As the client application should support the simultaneous interaction of multiple user, the collaboration of multiple remote sites, and the use of existing 2D and 3D documents, no ready-made software package or programming framework was available. Therefore, a decision was made to base the client application on multiple programming libraries that provide functionalities at a lower level (graphics rendering, sound playback, etc.).

First, the software components are introduced and some principles for layering functionality are presented. The main data structure is presented next, followed by the synchronization of this data structure over the network. Next, the implementation of the interactions is explained. The chapter concludes by presenting the video conferencing system adopted to enable the communication between different sites.

6.1 Parts of the System

As mentioned in Chapter 5, the software for interfacing with the input devices was split into a separate software package. This allowed its reuse in different projects like the Flash application presented in Chapter 5. Besides this rather coarse division, a more detailed structure of the software can be derived from the requirements listed in Chapter 3. The requirements for the software can be summarized as support for freehand drawings, external documents (2D, 3D), cards, the digital representation of the input devices,
and a background. The interactions carried out with the TUIs and the pens should operate on these data entities. Such operations can be summarized as follows: draw lines, erase lines, manipulate cards, create card from background, measure, scale, change pen property, and import document on card.

### 6.1.1 Data Flow

One way of designing the software is to reason about the data flow. Below, an overview on the expected data flow is given (see Figure 6.1):

- **The input technology** creates events describing the interaction of a user, for example, pen moved to position x/y while a button was pressed.

- **The event handler** evaluates the event and decides how to react to it, for example, by adding a vertex point to a line.

- The operations are then sent to the **operation dispatcher**. It first queues them up and applies them later on the data structure. Additionally, the dispatcher also passes the operations to the network adapter. The dispatcher also receives operations from the network adapter and executes them.

- **The network adapter** remains in contact with the remote sites. It receives operations from remote sites and also sends operations to remote sites. As noted before, it exchanges operations with the operation dispatcher.
• The **data structure** models all the groupware data, storing all data entities listed at the beginning of this section (like drawing, cards, documents, etc.).

• The **graphic renderer** displays the data structure on the screen.

• The **audio renderer** is responsible for playback sounds in the scene.

### 6.1.2 Packages and Dependencies

The data flow already depicts many aspects of the system and provides a first overview of how the parts of the system interact with each other. Nevertheless, this does not show all aspects of the system, such as how to import and export data. Also, the data structure part in the diagram lacks detail. In the following, the key packages and their functionalities are listed. Figure 6.2 shows these packages and their interdependencies.

The **data structure** of the groupware contains its state, which consists of the data that can be changed through the users’ interactions. Examples for this are the sites that take part on the collaboration, the configuration of the UI, the lines the users sketched, and the position and orientation of the imported documents. As mentioned earlier, a unified data structure was sought for all states of the application. This means not having completely different ways for accessing the list of the participants, for changing the view on the sketches and drawings, or for setting the color of a line. The data structure does not depend on any of the other packages, but parts of the data structure do depend on other parts of the data structure. For example, the user interface must show the users’ drawings and therefore depends on a component that contains them. Below, the different aspects of the application’s state are listed.

The **session management** contains the information about the participants in the current collaboration session, including information about the different sites (New York, Zurich, etc.). The workspaces contain data about the collaborating users, the display elements (computers), and the available input devices.

The **OS adoption** part of the data structure contains the state related to the OS of the client application. An example for such a state is the computer’s screen size or the size and the position of the application’s windows. Further, the gathered information may indicate that the computer may contain additional collaboration software that should be started. For example, an external video
conferencing software may need to be started on a specific screen to support the collaboration.

The digital representation/UI package models the state of the user interface. Examples of this state information are how the view on the sketches and documents is currently set up or what the current digital representation of the ruler is.

The scene contains the application state consisting of the graphical content the user is interacting with: the users’ sketches and the imported documents.

The operations package contains all functionalities used to handle the modifications the user wants to perform on the data structure. In order to express the operation, the operations packages need to know the data structure. When drawing a line
from Point A to Point B, for example, the operation needs to know the scene where the line should be added. While many operations must be provided to support all modifications on the data structure, the operations themselves provide only minimal functionalities. The most important one is the “execute()” function that executes the modification on the data structure. The terms operation and command are used interchangeably in this text (see the command design pattern in [136]).

The operation dispatcher sequentially executes the operations from simultaneous users. It synchronizes the commands submitted by local and remote users. Further, it keeps a history of all operations applied to the data structure. In order to receive operations, it provides a “submit(Operation op)” function to other packages. The dispatcher must know the network package, so that it can send operations to remote sites.

The network package contains the functionality of communicating with remote sites. This communication is effectively an exchange of operations containing the users’ modifications on the data structure. The package accepts local commands for later transmission through the “submit(Operation op)” function. Further, it reads incoming operations from the network connection and sends them to the operation dispatcher. When being sent, the operations must be converted to the format used for transmission over the network (serialization). Upon reception, the raw byte stream must be deserialized into the proper operation instances.

The import/export package provides means to read and write the digital content of a scene. For example, the user might want to import a machine drawing, take scene shots, or he might want to save the current state of the scene. In order to import data, the package needs to create the operations that reflect the desired modification to the data structure. To export data, the package needs to read the data structure and translate it into the file format used for storing. Taking screen shots requires access to the graphics renderer and saving the rendered image to a file in a specified format.

The graphics renderer starts at the level of the previously mentioned OS adoption package and converts the data structure below (UI, scene) into an 2D pixel image. This image is then shown to the user on the display elements.

The audio renderer is similar to the graphics renderer since it generates output to the user. It traverses the data structure and outputs sounds that are attached to
elements of the data structure. If a pen is pressed on the display, for example, a gentle click gives the user an audible feedback of his interaction.

The **event handler** listens for incoming events from input devices (see previous chapter) and translates them into operations. The event handle needs to know the operation package to create new operations, the data structure to specify what the command modifies, as well as the operation dispatcher to submit the newly created events.

### 6.1.3 Layers and Third-Party Software

A layer diagram can be used to illustrate the dependency on third-party software (see Figure 6.3). The upper layer elements, which are more application specific than the lower (e.g., see [82]), depend on the functionalities of the lower elements. For example, the networking components of DOG use the ACE third-party library, and ACE uses the system libraries provided by the operating system.

![Software layers diagram](Figure 6.3: Software layers of the system: the blue components are part of the DOG client application; orange components are the libraries that were used to implement DOG. Among the libraries, “Tracking” and “CGUT” were developed within the presented work.)

Existing programming libraries were integrated into DOG in order to provide the required functionalities. Figure 6.3 shows these third-party software packages. As an
example, the diagram shows that DOG integrated the third-party libraries ImageMagick and lib3DS for importing and exporting data. If the user takes a screen shot, the import/export package of DOG asks for the current image displayed. It then creates a data structure of ImageMagick, fills the image into the data structure, and calls the save function of ImageMagick with the given filename. All ImageMagick functionality DOG uses is hidden behind an application-specific interface (current screen to file). This way of wrapping the functionalities makes it easy to replace a library with another library. Only a small portion of the code would need to be changed if, for example, the ImageMagic library needed to be replaced with another image processing library. Therefore, maintainability is improved. The upper layer of DOG interacts only with the lower layer of DOG, but not with the layer below the DOG sub-system (ImageMagic). This way of layering is called strict layering. Whenever possible, strict layering was used to integrate the functionalities of third-party libraries.

There is a more recent notion of strict layering which tries to change the way dependency among software modules is normally perceived. The Dependency-Inversion Principle is summarized by [85] as follows:

- “High-level modules should not depend on low-level modules. Both should depend on abstractions.”
- “Abstractions should not depend on details. Details should depend on abstractions.”

So the DOG import/export submodule defines how DOG depends on an image library for saving screen shots. How the low-level module (ImageMagick) implements this functionality need not be considered when developing applications within DOG.

For some libraries, the required functionalities were not expected to change during development. Therefore, they were used as a basis for DOG itself. This way of layering is called loose layering. The ACE and the CGUT library were used in such a way. ACE provides a layer above the actual operating system, making the software portable to other platforms supported by ACE (Windows, Mac Os X). ACE was used in particular for multi-threading, networking, and serialization. CGUT was developed within the presented dissertation. It provides math classes for calculating vectors, matrices, and quaternions. Besides being used for DOG, it also was used for the TRK library described in Chapter 5 and for the Augmented Reality application mentioned in Chapter 1. CGUT and ACE were directly used within the data structure of DOG. Loose layering has only been used for basic functionalities that are very unlikely to change as, for example, transforming a vector with a matrix.
6.2 Data Structure

An important requirement for the development of DOG is the ability to handle 2D and 3D data. A scene graph is a common data structure to represent this kind of data. As its name suggests, it is a “graph” that represents a scene of graphical objects.

6.2.1 Graph Data Structure

A “graph” is a data structure that consists of vertices and edges (see for example, [138] or [127]). An edge connects two vertices. In Figure 6.4 A, the vertices are labeled with letters and the edges with numbers. If the vertices of an edge are interchangeable, the graph is undirected (Figure 6.4 A). If they are distinct, the graph is directed (often just called a digraph). Figure 6.4 B shows a digraph. Another criterion for distinguishing graphs is whether cycles are allowed or not. A cycle is a path through the graph’s set of vertices and edges that ends at its starting point (Figure 6.4 C). Therefore, a graph is either cyclic or acyclic. Developing algorithms for cyclic graphs imposes an additional difficulty as they could potentially loop forever.

In a digraph, vertices can be analyzed by the number of incoming (indegree) and outgoing (outdegree) edges. Figure 6.4 D shows the indegree (i) and outdegree (o) for each vertex. A digraph is called a directed forest if the indegree of a normal vertex is at least one. Vertices with an indegree of zero are called roots. Figure 6.4 D shows a directed forest with two root vertices. A directed forest with a single root is called a directed tree (Figure 6.4 E).
To access the graph information, a program iterates through the data structure in a process called traversing. Drawing a scene graph or determining what 3D object a user wants to select requires a traversal. Different ways of traversals can be distinguished. In the first way, the connected nodes are processed before the actual node is processed (postorder). Traversing Figure 6.4 E would process the nodes in the following order: C,C,C,T,S,M,R. Whereas the opposite strategy processes the node before it processes the connected nodes (preorder). This would process the nodes in the following order: R,C,M,S,C,T,C. Depending on the application, preorder or postorder may be more appropriate. For example, postorder traversal must be used to determine the number of nodes on the outgoing side for each node, whereas preorder traversal is appropriate for determining the distance of any node from the root node. Preorder and postorder can be combined so that each node is processed before and after the connected nodes have been processed.

As mentioned before, a scene graph is a graph specialized for graphical applications and is related to the directed tree mentioned previously. The edges are used to model a part-whole relationship between the vertices. Vertices contain the actual scene data (like geometry or attributes). This principle is exemplified in the following. Figure 6.4 F shows a very simple scene that consists of three circles: two circles are large and have an outline, but no fill, and one smaller circle is filled with black color. Figure 6.4 E shows a graph that corresponds to the scene shown in Figure 6.4 F. A scene can often be represented through different graphs, but a graph can only express one scene. The geometry of the circle is contained in vertex C. It is connected through three paths to the scene’s root vertex R. The circle has an initial size of one unit and a material description that fills the circle with black color. The other vertices modify the attributes of the vertices on their outgoing edges: vertex M changes the material property to transparent with a black outline; vertex S scales by a factor of two; and vertex T translates vertices by 2.5 units into a given directory.

The assembly tree known from mechanical engineering is a sibling of the scene graph. It structures products and shows the dependencies of its components. Therefore, products in an assembly tree are also divided into their subparts. These subparts are named assemblies and are again divided into their subparts. The subdivision stops at the component level if the parts cannot be divided anymore. These atomic elements are called leaves. For the assembly tree, a screw, a washer, or a nut represent a leaf. This data model can be represented using the scene graph. The parts are mapped to the nodes, whereas the part-whole relation is mapped to the edges. Figure 6.5 shows how a complex scene and an extremely simplified scene are represented in a graph.
6.2.2 Design Variations of Graphs

The previous subsection gave a brief introduction into the basic principle of the data structure. In this subsection, design variants of scene graphs are discussed. As the data structure is a core component of the client application, multiple designs were evaluated. References to scene graphs using certain design variants will be given. The variants are demonstrated using a small portion of how the real data structure would need to be designed.

In the following, we will use the term node as a synonym for vertex in a scene graph. Leaf will be used for a vertex/node without subvertices. The design variants are named after the number references to the parent and child vertices they support.

For example, "0PnC" does not support references to parent vertices, yet it supports any number of references to child vertices.
6.2. Data Structure

Figure 6.6: Variant “0PnC” Class Diagram:

A) Variant “0PnC” Class Diagram:

B) Variant “0PnC” Example Graph (G = Group Object, O = Object Object, F = Font Object)

Figure 6.6: Variant 0PnC: nodes have only multiple children nodes.

The simplest implementation (0PnC variant) of a scene graph is shown in Figure 6.6. The left side shows an incomplete class diagram, and the right side shows an example graph. See [82], for example, for more information on class diagrams according to the Universal Modeling Language standard (UML). The class GroupChild specifies the interface of all node classes in the tree. The Group class can store multiple GroupChild nodes or nodes that are subclasses of GroupChild. Font and Object nodes constitute the leaves of this simple tree. The example variables "foo", "bar" and the corresponding getter and setter functions are given to show how object attributes are implemented. The Font node represents a formatted text in the scene, the Object node represents a data structure containing any 3D object. This can be a stroke in a sketch or a part of a loaded CAD model. This design variant is similar to the composite design pattern [136].
It is easy to implement and it allows nodes to be used as children of many parent nodes. For example, in a car model, a single Object node with the geometry of one tire can be used for all four tires. This is possible as no information about the parent node is stored. The drawback of this design choice is that it is no longer easily possible to refer unambiguously to a node. A reference to the left front tire is also a reference to the other three tires. This makes referencing and navigation among nodes more complicated. Therefore, the whole path from the root to the reference node must be used to identify a node. This path consists of a list of references to the nodes in the path. Maintaining the paths and passing the paths to functions imposes an overhead on this design. A similar design is used by the Inventor scene graph [122].

The next design (1PnC) adds a reference on the parent node to the GroupChild as shown in Figure 6.7. It allows the parent node of any given node to be accessed. The parent variable of the root node is set to a special value to indicate that it has no parents (ground symbol). In this design version, the car model would have four nodes containing identical data.

Figure 6.7: Variant “1PnC” Class Diagram: GroupChild Class has Parent Attribute

B) Variant “1PnC” Example Graph (G = Group Object, … )

Figure 6.7: Variant 1PnC: nodes have multiple children nodes and a parent node.
The design variant shown in Figure 6.8 (nPnC) defines a list of parents for each node. This brings back the possibility of using a single tire Object for modeling the four wheels of a car. But a reference to an object is ambiguous if the node or any parent node has more than one parent (for example O3 in Figure 6.8). The reference does not indicate to which parent the child node is referred. Nevertheless, this data structure allows all parents to be updated by simple reference. However, querying the siblings of a node returns the siblings relative to all of its parents. For example, in the hierarchy shown in Figure 6.8, the B node O3 is a child of G1, G2, and G3. If a program queries the siblings of O3, it will get all nodes except the root node G1. This makes it necessary for certain situations to reference a node by using the full path from the root node of the graph. A variant of this design is used in Performer [111].

To resolve the ambiguities mentioned in the previous paragraph, a generic Node class can be introduced that contains a reference to a specialized Leaf class (Core in OpenSG). This design (1PnC1C) is shown in Figure 6.9. The generic Node class allows arbitrary trees to be built. Like the 1PnC design, it references only one parent class. Yet as the
specialized functionality is extracted to the Leaf class, multiple nodes (N4, N8, N9) can reference the same Leaf instance (O3). Unlike 0PnC, a function taking a node as an argument has access to the tree’s hierarchy. The geometry of the car’s tire must only be stored once. Nevertheless, it permits scenegraphs to be built that do not make sense. As every instance of a node can have subnodes, even elements which logically should end the tree can have further subnodes. Since the Node class references the base class Leaf, a function working with the node has to determine the type of leaf reference. Subsequently, two references must be used in the program: one for the hierarchy and one with the actual reference to the leaf (for example, a FontLeaf). OpenSG uses a design related to the approach shown.

The following section presents the 1PnCL design. It is different from the previous design by hiding the Leaf from the programmer. The Node class provides the only public interface and the Leaf class is reduced to a simple data container (see Figure 6.10). Unlike in the previous design, one reference is enough to access the hierarchy and the specialized object in the tree.
6.2. Data Structure

Figure 6.10: Class diagram of variant 1PnC1L: specialized nodes share specialized node container classes.

Figure 6.11: Example graph of variant 1PnC1L. Nodes ON 2, ON 4, and ON 5 all share the same leaf OL 2

A simple interface is provided to allow the programmer to control how the leaf is shared:

- When a node is created, a corresponding leaf is created. In Figure 6.12 part A, ON1 refers to OL1.

- When a node is created as a copy of another node (by calling the source objects cloneNode() method), it shares the leaf with its source node. In Figure 6.12 part B, ON1 and ON2 refer to OL1.

- Calling a node’s cloneLeaf() method will make sure that it is the sole node referring to the leaf. If necessary, the method creates a copy of the current leaf. Figure 6.12 part C shows that ON2 has its own Leaf OL2 after ON2.cloneLeaf() was called.
This mechanism also works with inheritance hierarchies. As shown in Figure 6.10, the ObjectNode extends the GroupChildNode class. Therefore, the ObjectNode class inherits those properties. It effectively contains references to two leaves: the GroupChildLeaf and the ObjectLeaf. The methods to control how the leaves are shared must address this issue.

Attributes can now be stored in either the node or the leaf. If the attribute is related to the object role in the hierarchy, it should go into the node. For example, the transformation into world coordinates, the definition of the bounding volume, or a simple text label must be stored in the node. Attributes that are shared among nodes must go into the leaf. Examples are material definitions, geometry definitions, and so on.

Whether an attribute should be shared or not might not always be obvious. Figure 6.10 part A shows that the GroupNode class contains the list of subnodes. Therefore, if cloneLeaf() on a GroupNode is called, the resulting GroupNode has its own list. As the nodes in the source list only reference one parent, the list must be created by recursively cloning all child nodes in the source list. Depending on the programmer’s expectations, he might be puzzled when he adds a node to the cloned GroupNode and it does not show up in the source GroupNode. This ambiguity also exists for design 1PnCC and has been noted by its inventor [105].

The design (1PnCS) in Figure 6.13 ensures that a GroupNode returned by calling cloneNode() stays synchronized to its source GroupNode. The basis for this synchronization is the sharing of the "nodes" list in GroupLeaf with the other synchronized GroupNodes. All child nodes in the list and their subnodes must share the leaves with the corresponding nodes in the other lists (see Figure 6.14). In Figure 6.14, GN2 and GN3 share the same GL2. Therefore, their child nodes (GN4, FN1 and GN6, FN2) also refer
6.2. Data Structure

Figure 6.13: Variant 1PnCS: synchronized GroupNodes are realized through the list of referring Nodes in the GroupLeaf object.

to the same leaf objects (GL5 and FL2). Unlike the design 1PnCL, adding a new node to one GroupNode will also add a cloned node to the other GroupNode to maintain the synchronization. The basis for this mechanism is that the GroupLeaf references back to the GroupNodes, so that any modification made to one group can also be applied to the others. For example, Figure 6.15 shows what happens if a new ObjectNode is added to a cloned GroupNode. Each GroupNode (GN2, GN3) has its own new ObjectNode (ON1, ON2) referring to the same ObjectLeaf (OL1). The previous design was focused on reusing the content stored in nodes (the car’s tire). This design allows the structural elements of the scene graph to be reused.

Figure 6.14: Variant 1PnCS: example of two synchronized Groups with a FontNode and a GroupNode as child nodes.
Chapter 6. DOG: Distributed Object-Oriented Groupware

Figure 6.15: Variant 1PnCS: to maintain consistency, calling cloneLeaf() on any node will also clone the leaf nodes of parent nodes.

If a child node of a synchronized GroupNode calls cloneLeaf(), it destroys the synchronization. One example is calling ON1.cloneLeaf() in Figure 6.15, while GN2 and GN3 still both reference GL2. As now ON1 and ON2 do not reference the same leaf, GN2 and GN3 are also not identical anymore. The groups GN2 and GN3 must be split. Therefore, if cloneLeaf is called on a node, the node must call cloneLeaf on its parent until an unshared GroupNode is encountered. Figure 6.16 shows a scene with three synchronized bike handle bars consisting of the bend tube, two handles, a lamp, and a bell. If the user decides to change the knob of the bell on all handles, no additional operations must be done except the desired changes to the geometry or material. But if the user would like to change only one knob, he first has to call the knob’s cloneLeaf() method and then change the knob’s geometry. This changed knob makes the bell different from the other two bells and therefore, it needs its own leaf. Subsequently, the changed bell makes one handle bar different from the other two handle bars. As a result, the handle bar with the modified bell also need its own leaf as the synchronization is broken.

The presented system uses the design explained last. Besides allowing the geometry to be reused, the design also allows parts of the graph to be reused. Further, a single reference unambiguously identifies a node in the graph. Unlike the core/leaf model, the leaf is hidden behind specialized node classes. The chosen approach should simplify programming, since the programmer does not need to check the type of the leaf before accessing it. Further, the approach is more restricted and does not permit the client to build nonsensical scene graphs.
6.2. Data Structure

Figure 6.16: Variant 1PnCS: shows the effect of calling cloneLeaf() to change the bell’s knob model in a scene containing three identical bike handle bars. Red relations are reverted, whereas green relations and objects are newly created.

6.2.3 The Objects of the Graph

After presenting the principles of the data structure, an overview of the element types actually making up the data structure is given.

As explained in Chapter 3, the data structure must contain more than just geometry, and this information should be synchronized over the network. The data falls into the following categories: graphical content, user interface, physical setup, and session management.

Figure 6.17 shows a reduced set of the node objects used in DOG. The circles represent a Node object and its corresponding Leaf object (see previous section). A continuous line denotes a one-to-many association. The higher node has zero or many nodes of the lower nodes. For example, Session references objects of the type Workspace. A dashed line refers to an association of one to one. The ViewPort, for example, has only one Camera, or the Object has only one Material.
Figure 6.17: Data structure with nodes and associations

Overlapping circles denote a hierarchy of classes that were derived from a base class through inheritance. The class in the highest position is the base class of the hierarchy. In the case of LayoutComponent and GroupChild, a composite pattern (see [136]) is
6.2. Data Structure

present to build a part-whole relation with the classes of the hierarchy. It was denoted with the association line (e.g., between Group to GroupChild). The Group refers, for example, to objects of the type GroupChild, but Group also inherits from GroupChild. Therefore, Group can be associated with GroupChild or its subclasses. Referring back to the assembling tree example in Figure 6.5, the composite pattern allows the Group “car” to refer to a Group “chassis” which in turn refers to four “tires” with the type Object.

It is important to keep in mind that Figure 6.17 shows only one vertical cut through an actual data model. Every continuous line in the figure would unfold to many instances of the lower type. For example, below an instance of Session, there are many instances with the type Workspace; below an instance of Workspace, there are many instances with the types Input Device, Host, or User.

Figure 6.17 is color coded to denote different aspects of the collaborative application. Classes that relate to aspects of session handling are shown in the top part of the figure. Each instance of the DOG application can be part of a collaboration session with other computers. A Workspace represents a geographical location like a conference room. So it features data like longitude, latitude, time zone and so forth. The Workspace refers to Input Devices, Users, and Hosts that are located within it. This data structure reflects the goal of allowing multiple users to interact with multiple input devices on multiple computers. The state of the input device is not bound to a computer and its connected display. For instance, when the color can device (see Chapter 3) is in pipette mode, it should stay in this mode even when being moved from one display to another. Similarly, the objects the user has selected on one display should still be selected when he moves to another display.

The Host and the Window are classes DOG uses to interface with the operating system running the application. Each host in the session has an instance of the Host class in the proper Workspace instance. Besides simply grouping the windows that are on a host, it is the proper point to make OS related data accessible, or to provide OS related features (file transfers, printing, remote shutdown, etc.). The Window class represents a window in the host’s operating system. When the user manipulates the window size, DOG must adjust the UI on all hosts showing the window. As shown in Figure 6.17, the Window refers to two classes of the UI. One class takes control over the windows’ background areas, while the other manages the overlays. These are visuals that have to be displayed on top of the background. Some examples are the cursor of the styli and the digital representation of the tangible input devices.

As normal GUI toolkits like Windows Forms or Java Swing did not support multiple users, a GUI system was developed for DOG. In many points, the design follows existing
systems. However, it supports multiple input devices and arbitrary orientation of its visuals. Since users will gather around the interactive table, the GUI must be able to present its visuals in any orientation.

The base class LayoutComponents may be associated with a LayoutManager. The manager assigns each component to a part of the window for rendering. A vertical manager arranges its components above each other and the horizontal manager puts them side by side. The absolute manager allows each component to be placed with absolute screen coordinates. A menu bar, a context menu, a view port, a separator and the components representing the cursors, and TUI visuals were implemented.

6.3 Data Conferencing

Synchronous groupware applications can be implemented using various architectures. Patterson presents a taxonomy for groupware applications in [102]. Patterson states that the primary challenge of groupware applications is to keep the state consistent across machine boundaries. Further, an application may be divided into four levels of state: display, view, model, and file (see Figure 6.18). The display state refers to the information that drives the user’s display. It is the state of the frame buffer. The view state is the information that determines how the model’s information is transformed before being shown on the display. The model state defines the state of the actual data model. In the case of DOG, the model consists of the geometry of the drawings and source documents. The file state is the persistent representation of the groupware. Patterson states that an application can maintain consistency by state sharing, state synchronization, or by a mixture of both.

In the case of state sharing, collaborating instances of an application share the state at a certain level. All the depending state levels (shown at a higher position in Figure 6.18) are also shared as a consequence. A well known example is Application Sharing (AS). Figure 6.18 part B shows how state sharing is used in the case of AS. All peers share the same view, model, and file states, while each has its individual display state. The display state solely depends on the view and the model states, consequently, the system stays consistent.

AS systems are usually implemented by intercepting the function calls for rendering the application’s view into the display. The calls must then be compressed and transferred over the network to the remote peers. Finally, the calls must be applied on all hosts to show the same view. To not only show the graphical output, but to also enable
interaction, the peers must capture their input devices and transmit the actions to the
machine hosting the shared states. The host finally injects the input events into the
GUI. These events may then change a state of a shared application. The display is
refreshed to reflect these changes and then sent to the peers again.

AS systems have the advantage of working with any existing application, but as ex-
plained earlier, the OS only allows a single active user. This approach is therefore
unsuitable for the proposed system. Nevertheless, the principles of the architecture
can also be implemented at application level. An application developed using such an
architecture would directly send its graphical output to its clients. The client applica-
tion could capture the input events from the TUIs and pens and send them back to the
server. This approach would allow multiple users to interact at the same time. The
main argument against this architecture is its limited scalability. A server would need
to render and transmit the full display for each of its clients. A display of 1024x768
pixels (32 bits per pixel) refreshing 30 times per second produces a network load of 90
megabytes per second. Even using image compression, this approach would be limited
in its scalability. Further, the network’s latency affects the responsiveness of the appli-
cation. If a user interacts in the client application, the events must first be sent over the
network and processed before the rendered result is sent back again over the network.
Chapter 6. DOG: Distributed Object-Oriented Groupware

Transcontinental network latencies are approximately 100 milliseconds. Such a delay is noticeable for the user and becomes a usability issue when precise movements are required, as in sketching.

For this reason, synchronization was used as the mechanism for maintaining the state of the groupware. Unlike state sharing, where a state has to be accessed over the network, state synchronization allows each peer to have its own copy of the state. Using such an architecture, each peer may render the scene as fast as it can (30 frames per second or even more) without congesting the network. Only the synchronization signals have to be exchanged among the peers.

6.3.1 Synchronization

Synchronization tries to assert that the state of processes \( p_i \) in a groupware stay consistent. For the presented work, each display element (see Section 3.1) runs its own process of the groupware. The states of these processes must not be identical at any time, but rather the states of all processes must converge. This means that the data model of the processes must converge to the same state if no change happens for a certain period (quiescence).

The state of a process can be described by the events \( e_i \) that changed its state (see, for example, [59]). In the following sections, event, operation, and command will be used interchangeably. An event represents a change to the process memory and for some algorithms, sending data over a network also represents an event. For the presented application, an event means a change to the data model of the groupware. For example, the user adds a new point to the line he is sketching. Each event within a process can be placed in an order. \( e \rightarrow e' \) denotes that \( e \) happened before \( e' \). The history of all events occurring in all processes captures all states of a distributed system like the presented groupware.

\[
\text{history}(p_i) = h_i = \langle e_i^0, e_i^1, e_i^2, e_i^3, \ldots \rangle
\]

Synchronizing the collaborating processes means synchronizing their history of events. To achieve this, processes can exchange and apply their events mutually. For the presented application, such an event could be the deletion of a line in a drawing. The event is sent over the network to all other display elements (running the groupware process) and applied to the receiving processes. The line will therefore be deleted on all display elements.
The approach would work as long as the events are sent by the peers one after the other. If the events overlap, inconsistencies can arise. Overlapping means two events were simultaneously sent by two processes. Each event was sent before the other event was received. Figures 6.19, 6.20 and 6.21 show three examples of overlapping events and their effect on a graph. All these figures follow the same layout and consist of four parts. The left parts of the figures show the data structure on both processes before the conflicting events were sent. The parts on the left side show the data model of Process (computer) 1 after the events have been processed and the right parts show the same for Process 2. The parts on the right side of the figures illustrate the exchange of the operations. The type of the operations is written on the intersecting arrows. Each row shows a different conflicting situation.

Figure 6.19: Overlapping events for adding a new node to an existing node (1.B).

The figures illustrate that not all overlapping events will lead to an inconsistent state. Figure 6.19 shows, for example, that in the case of both processes adding a child node to the same parent node (1.B), the system is still consistent, since the order of the child nodes does not matter to this application.

Figure 6.20: Inconsistency through overlapping events changing the same property of the same node.

Figure 6.20 shows overlapping events that make the states of the two processes inconsistent. One process sets the color of node 1.C to blue, the other process sets it to green. After the events have been processed, the synchronization is lost as the graph on computer 1 has a blue node, while the other computer's node is green.
Chapter 6. DOG: Distributed Object-Oriented Groupware

Figure 6.21: Overlapping delete and change events are addressed by retaining deleted objects for some additional time.

Figure 6.21 shows another pair of overlapping events that does not break the synchronization. One process deletes node 1.C, while the other computer sets the node’s color to green. On computer 1, the node’s color is first changed and then deleted, while on computer 2, the node is first deleted and then its color is changed. This requires that the deleted nodes still be accessible in case further events for changing the node should arrive (the trash bin in Figure 6.21) or another way of discarding events affecting a deleted object must be provided.

Besides overlapping events sent from different processes, events from the same process can also overlap on their way to other processes. This is shown in Figure 6.22. The second event may have taken a different route through the network than the first event and therefore arrives first. These issues were addressed by the choice of communication protocol. For TCP/IP networks, internet protocols (IP) like the user datagram protocol (UDP) only provide a means of sending individual data packets. The IP packets must not necessarily arrive in the correct order. The transmission control protocol (TCP) guarantees that the data arrives to a process in the same order as it was sent.

Figure 6.22: Inconsistency through events arriving out of order.

Events affecting the properties of different nodes in the graph do not cause the synchronization to break. An example for such events would be if Computer 1 changes the color of node 1.C to green, while the other process changes the color of node 1.B to
blue. The given examples demonstrate that if the result of applying two events does not depend on the order, then they cannot break the synchronization (see Equation 6.1, [45] and [125]).

\[ e_1 \circ e_2 \equiv e_2 \circ e_1 \]  \hspace{1cm} (6.1)

In the given equation, \( \circ \) stands for composing two events (applying \( e_1 \) followed by applying \( e_2 \)). The symbol \( \equiv \) means in this context that the process state is equivalent: it produces the same output to the user. Most combinations of events do not affect the synchronization of the processes. One way of dealing with the few conflicting situations is to avoid them by allowing only one site (process) to do changes (also known as floor control). Before the users on a display element can make changes, they have to request control of the floor. As stated earlier, the simultaneous interaction of multiple concurrent users was important for this work (see Section 3.2.4). This should also remain true for remote interaction and so floor control is not an option.

Instead of locking the whole application, locking could also be done at a finer granularity, for example, at the level of each node. If a client wants to change a node, he first must obtain the lock. The client then changes the node and releases the lock again. A mechanism (distributed locks) makes sure that only one process is able to obtain the lock for any node at any time. This would prevent multiple clients from emitting conflicting events for the same node. Nevertheless, even the simplest distributed locking mechanism requires a network round trip to a remote server and back. As this imposes a latency of hundreds of milliseconds, it would be noticeable to the user and obstruct his or her interactions.

Considering the ratio of conflicting versus nonconflicting events, an optimistic concurrency approach seems more appropriate. Each process assumes all events are allowed without asking a locking mechanism for permission. This requires that conflicts can be detected and that a mechanism for resolving those conflicts is available. Distributed Operational Transformation algorithms (dOPT) provide such a mechanism. Ellis et al. [45] presented the algorithm for the GROVE groupware. GROVE uses IP multicast and therefore needed to address issues such as out of order events and the lack of a central server. Sun et al. [125] give an overview of different operational transformation algorithms. The algorithm presented by Nichols et al. [95] was developed for the Jupiter collaboration system. As Jupiter clients connect through TCP to a central server, this algorithm is simpler.

For the presented work, Nichols’s dOPT variation was adopted to fit the application. Figure 6.23 shows the basic principles of the algorithm. In the following, an overview of
the algorithm is given. Clients maintain a TCP connection to the server while the clients and server hold a copy of the data structure. This allows the client to make changes to the data structure and immediately show the result to the user. The server’s copy is used to process the events sent by its clients. Even though \( n \) clients may be taking part in the collaboration, as each client is only connected to the server, the synchronization can be realized on a per-connection basis.

As mentioned before, the algorithm must detect and resolve conflicting messages. Since clients are only connected to the server (unlike in [45]), conflicts from each connection between client and server can be resolved. If two events overlap on one client-server connection, both sides transform the received event and apply it to their copy of the data structure. The transformation must be defined so that the state of both copies is again consistent. A function \( xform \) is defined for transforming overlapping events on the server and the client side (see Equation 6.2).

\[
xform(c, s) = \{c', s'\}
\]  

(6.2)

The event sent by the client \( c \) and the event by the server \( s \) are transformed into events \( c' \) and \( s' \). When the server receives \( c \), it transforms it using \( xform \), applies the transformed version of the client event \( (c') \), and sends it to the other clients (making it \( s \) on these connections). As the server has already applied event \( s \), the result on the server side is \( s \circ c' \). The client on the other side transforms the server event \( s \) and ends up with \( c \circ s' \).
The $xform$ function is defined so that client and server end up with an equivalent state (see Equation 6.3).

$$c \circ s \not\equiv s \circ c \quad xform(c, s) = \{c', s'\} \quad c \circ s' \equiv s \circ c' \quad (6.3)$$

The client’s and server’s movements through the two-dimensional state space can be drawn (see Figure 6.24) to visualize the synchronization. For each event that the client sends, the first vector component is incremented. The second vector component is incremented when the server sends an event. This positional vector is a two-dimensional vector clock (see [81]) as used in original dOPT algorithm by [45].

Figure 6.24 shows that the client sends an event ($c_1$) first and the server sends the next event ($s_1$). At position 1/1, client and server simultaneously send events ($c_2$ and $s_2$). When they receive their partner’s event, they each transform and apply the event received. This makes the client and server converge again at position 2/2. The following events ($s_3$, $c_3$, and $s_4$) do not overlap and therefore the client and server take the same path. Each event contains the state the sender was in when it sent the event. This mechanism allows overlapping events to be detected. For example, if the client has sent five events and it receives an event indicating that the server so far has only received four events, an overlap was detected.

If the client and server diverge by only one event, each transforming and applying the events make them converge again (as shown in Figure 6.24). However, if the client and server diverge more, the problem is more complex. Figure 6.25 shows such a situation. The client and server diverged at position 1/1 (events $c_2$ and $s_2$ were sent). The client now receives event $s_2$, transforms it using $c_2$, and applies it ($s'_2$). Next, the client receives event $s_3$. Whereas for $s_2$, the event $c_2$ could be used as an argument for the transform function $xform$ for transforming $s_3$, no event connects positions 1/2 and 2/2. Figure 6.26 shows how this is resolved. In the previous transformation $xform(c_2, s_2) = \{c'_2, s'_2\}$, $s'_2$ would have allowed the server to move to position 2/2 in the state space. This transformed event connects 1/2 and 2/2 and can be used by the client to transform $xform(c'_2, s_3) = \{c''_2, s'_3\}$. The event $s'_3$ can then be applied by the client in order to reach position 2/3 in the state space. If the client now received another event indicating that the server had still not processed $c_2$, $c''_2$ could be used to transform this event ($s_4$).

Each event sent to the other party must also be kept in an outgoing buffer. In the previous example at the diverging point, this was $c_2$ for the client. Further, the transformed events that are not applied locally ($c'_2$, $c''_2$) must also be stored in this buffer to transform incoming events. Whenever an event is received, the sender’s state is checked and the events in the outgoing buffer that have already been received by the server are deleted.
As mentioned before, each event contains the current state of the sender. If a client sends no events for a long period while the server keeps sending events, the server's outgoing buffer grows large. Therefore, no-operations events containing just the state are periodically sent to the other party to shrink its buffer.

### 6.3.2 Implemented Events

In the following, the events implemented for the system are presented. Table 6.1 shows their basic type in bold and gives some examples. The examples are explained in the
6.3. Data Conferencing

Figure 6.25: The client and server diverge by more than one event (when the server sent $s_3$, $c_2$ had not yet been processed).

Figure 6.26: The result of transforming $c_2$ ($c'_2$) is used as input for transforming $s_3$ to $s'_3$.

text after the # sign. The implemented events only cover what was used by the groupware to fulfill the application’s requirements (loading objects, annotating, and sketching). About 180 events were defined.

Normally, the events were defined as follows: the desired functionality would be implemented as for an only-local application. For example, the member function of the GroupChildN class to hide the object on the display setHidden(boolean hidden) is implemented. This functionality is then locally tested using normal member function calls groupChild setHidden(true). Now a global function is defined for the same purpose
Creating objects as children of existing nodes:
GroupN::Ptr_t group = addGroup(scene);
   # add a group node to the scene
TextN::Ptr_t text = addText(group);
   # add a text node to the group

Remove objects from the data structure:
deleteObject(object);

Modify object properties by calling “setter” functions:
setText(text, “Hello World”);
   # sets the display text of the text node to “Hello World”
setHidden(object, true);
   # flags the object as hidden
setAmbientColor(material, ColorRGB.RED);
   # sets the ambient color of the material to red.

Copy objects within the data structure
GroupN::Ptr_t g = copyGroup(font, group);
   # copy the font object and add it to the group node

 setHidden(groupChild, true). Calling this function should then create and submit the event for distributed execution. Normally, the function is implemented by instantiating a class template (see [36]). For each type of event shown in Table 6.1, a class template was created (see Table 6.2). For modifying objects using calls to member functions (setter functions), class templates for zero to six arguments are provided. The implementation of the setHidden function is shown below. The class template for member functions with one argument is used, therefore, the template’s name ends in “A1”.

Table 6.1: Types of events for manipulating the data structure. GroupN::Ptr_t is the type of the pointer used to point to a GroupN object.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating objects as children of existing nodes:</td>
<td>GroupN::Ptr_t group = addGroup(scene);</td>
</tr>
<tr>
<td>Remove objects from the data structure:</td>
<td>deleteObject(object);</td>
</tr>
<tr>
<td>Modify object properties by calling “setter” functions:</td>
<td>setText(text, “Hello World”);</td>
</tr>
<tr>
<td></td>
<td>setHidden(object, true);</td>
</tr>
<tr>
<td></td>
<td>setAmbientColor(material, ColorRGB.RED);</td>
</tr>
<tr>
<td>Copy objects within the data structure</td>
<td>GroupN::Ptr_t g = copyGroup(font, group);</td>
</tr>
</tbody>
</table>

...
Table 6.2: Implementing the hide command using a class template for member functions accepting one argument.

```c
void setHidden( GroupChildN* gc, bool isHidden ) {
    return CMDNetObjFuncA1<
        GroupChildN,  // object type having the member function
        bool,        // type of the argument
        NMGCSetHidden,  // number used to identify the event
        &GroupChildN::setHidden  // pointer to member function to call
    >::create(gc,hi);  // creates and submits the event
}
```

### 6.3.3 Resolving Conflicting Events

The requirements for the client application did not include sophisticated editing of content. This should be done in domain-specific applications. The events therefore cover creating the data structure of the imported documents, adding sketches to the background, adding sketches to cards, copying sketches, and deleting objects. The fine grained manipulation of sketched lines or 3D objects was not needed.

As shown in the previous sections, simultaneously adding two child nodes does not affect the synchronization. Further, since objects are kept for a while in a buffer before they are deleted, overlapping deletes does not make the state of the processes diverge. The only case that needed to be addressed was overlapping modify events (for example, two change-color events). This was handled by giving the server’s event priority over the event from the client. As the server’s event also originates from a client, this mechanism means that the client reaching the server first wins.

\[
\text{form}(e_{\text{change}}, s_{\text{change}}) = \{ s_{\text{change}}, \text{no-operation} \}
\]

The more editing functionality added to the groupware, the more events would need to be transformed. For example, any combination of the previously introduced `cloneLeaf` function (see Section 6.2.2) and a modify event would need transformation. The `cloneLeaf` event would make nodes the use different leaves for sharing their properties, while the change event would modify the properties in the leaves. If they were not transformed, one party would change the properties in both leaves \( e_{\text{change}} \circ e_{\text{cloneLeaf}} \), while the other party only change one leaf \( e_{\text{cloneLeaf}} \circ e_{\text{change}} \).
Chapter 6. DOG: Distributed Object-Oriented Groupware

6.4 Implemented Interactions

In the following section, the implemented interactions are presented. First, the software design used to implement the interactions is presented followed by an explanation of the individual interactions.

6.4.1 Implementation

This subsection presents the software structure used to implement the interactions. Figure 6.27 shows the components taking part in an interaction. The associations among the components are drawn with black lines. Figure 6.17 shows the location of the components in the full data structure.

![Diagram of classes used to implement interactions]

- The **EventHandler** belongs to the input interface described in Chapter 5. The EventHandler distributes the input events (like a stylus moved to position X/Y) from the input devices to the subscribed client code.

- The **Window** represents a window on the computer’s screen. It contains data about the window’s size and position on the screen. Using this data, an orthogonal
projection onto the digital representations of the interactions devices is set up (see Figure 6.17).

- The **AbsLayoutManager** is a special LayoutManager. A LayoutManager determines the layout of the components in the UI. One LayoutManager may, for example, arrange the components in a row, while another arranges them in a grid. The speciality of the AbsLayoutManager is that it allows one LayoutComponent to occupy the background, and multiple other LayoutComponents are positioned as transparent overlays over the background component. In Figure 6.27, multiple stacked LayoutComponents (DeviceComponents) are shown to illustrate this one-to-many relation (see also Figure 6.28 and Figure 6.29).

- The **Viewport** is the LayoutComponent that shows a view on the scene. It therefore needs a camera and a scene.

- The **Camera** defines the view of a scene. The view is defined by a 4x4 matrix, allowing orthogonal views and perspective projections.

- The **Scene** contains the imported documents and the sketches. To describe this data, the scene contains many other objects (see Figure 6.17).

- A **DeviceComponent** is a LayoutComponent that represents an input device. In Chapter 3, it was also referred to as a digital representation of the input device.

- A **Group** is associated with DeviceComponent to contain the actual geometry of the digital representations. The DeviceComponent can hide, show, and alter the objects in the group to adapt the representation to the user’s interactions.

- A **Device** is associated with the DeviceComponent to contain the device’s data independent of its visualization. The Device is rooted high up in the data structure under the Workspace (see Figure 6.17). All windows in a workspace reference the same Device class, and all modifications to this class are propagated to all hosts in the same workspace. This is essential for assigning an input device’s virtual properties, which are consistent on all display elements in the workspace. The pen’s color, for example, must stay the same on all displays.

- The **DeviceState** reflects the different interaction states the input devices are in. To illustrate, the pen has three states: hidden (undetected), hovering, and drawing.
The data structure is used in two ways to realize the interactions. First, input events are propagated through the graph’s nodes, allowing the nodes to update their states. Second, a renderer traverses the data structure to update the screen content. Both ways are presented below.

**Interaction Event Flow**

Here the flow of the input events is described. The window is subscribed for the input device events. The EventHandler then sends new events to the window. The window might need to discard events that occur outside of its area. The events from inside the window are then passed to the AbsLayoutManager. The layout manager first sends them to the DeviceComponents that lay over the viewport in the background. Any DeviceComponent may intercept the event from an input device. For example, the color-can component may absorb the events from the color-can device and update its visualization accordingly. Yet it might also absorb events from the pen when the user wants to change color. Within the DeviceComponents, the events are first passed to the TUI devices (color can, ruler, etc.), and the pens are the last device components to receive the events. This allows the TUIs to intercept the pen’s event before the pen starts to draw. If a LayoutComponent (for example, the DeviceComponent) signals that it consumed the event, it is not propagated further. If no LayoutComponent consumed the event, it is passed to the LayoutComponent in the background of the AbsLayoutManager and finally ends in the viewport.

For development purposes, the viewport implements some keyboard interactions in the viewport. Shortcuts and simple text commands were implemented to create and manipulate the objects. DOG also supports reading the same text commands from a text file (see Table 6.3). Such simple scripts were then used to initialize the environment for different tasks/demos.

A traditional menu was developed for experimenting in the environment with pure Single Display Groupware (SDG) applications. The menu also proved to be useful for realizing less frequently used functionalities (screenshot, hide object, etc.) during development. The menu is a LayoutComponent that is either used in the pen’s DeviceComponent as a context menu or in a LayoutManager in the background as an application menu bar (for example, on top of the viewport). As a menu bar, it allows the simultaneous interaction of multiple users. The context menu was given a special behavior for the horizontal workspace. In these settings, it is always oriented so that the text is parallel with respect to the nearest border of the window. Therefore, the menu items should
6.4. Implemented Interactions

Figure 6.28: Schematic of the classes that react to input events.

Table 6.3: Script to create a rectangular surface (each line’s command is commented after the # sign).

<table>
<thead>
<tr>
<th>Command with arguments</th>
<th># Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>csobj</td>
<td># creates an object</td>
</tr>
<tr>
<td>sobjpos 0 0 0</td>
<td># positions the object in the origin</td>
</tr>
<tr>
<td>sobjsca 320 640 1</td>
<td># scales the object</td>
</tr>
<tr>
<td>sobjrot 1 0 0 -0.589</td>
<td># rotates the object</td>
</tr>
<tr>
<td>ctuquad 1 1</td>
<td># creates a quad mesh with only one subdivision</td>
</tr>
<tr>
<td>cmat</td>
<td># creates a material</td>
</tr>
<tr>
<td>smatd 0.3 0.3 0.3 1</td>
<td># sets the material’s diffuse color</td>
</tr>
<tr>
<td>smata 0.0 0.0 0.0 1</td>
<td># sets the material’s ambient color</td>
</tr>
<tr>
<td>smats 1.0 1.0 1.0 1</td>
<td># sets the material’s specular color</td>
</tr>
</tbody>
</table>

remain readable for users on all sides of the table.
Chapter 6. DOG: Distributed Object-Oriented Groupware

Rendering

For rendering, a special class (renderer) iterates over the graph and changes the state of the OpenGL, rendering context at each node it visits. Either the visual properties of the state are changed (like diffuse material color of the current material), or the geometry is drawn. The renderer starts at the first visible node, the window (see Figure 6.29). The blue arrows indicate how the renderer traverses the graph below the window. The objects in the graph control how their associated objects are visited. Furthermore, what OpenGL commands should be emitted to render a node is defined in the renderer class. For example, the Viewport class makes the renderer visit the Camera class twice. First, the renderer sets the projection matrix to match the camera's state, and on the second visit, the renderer restores the previous state (the view on the GUI). In between these two calls, the renderer is sent to the scene objects to render the scene's content.

Figure 6.29: Schematic of the classes that are used for rendering the UI. The numbers indicate the order of the rendering.

As mentioned before, the LayoutManager only organizes the layout of the UI. The actual graphical content of the UI is determined by the LayoutComponents. The components do not store the geometrical description of their visualizations, but just forward the renderer to the associated group. This is the same group that is also used to describe the scene's content by containing other objects (e.g., Object, Material, Text object, etc.).
6.4.2 Pen Interactions

The pen was designed to allow the user to perform two interactions: pointing and drawing (see Section 3.3.1). For pointing, the pen has a digital representation that is also visible at remote sites (see Figure 6.30). In order to draw, the user has to push the pen on the display. The pen’s digital representation shows the color the pen is currently drawing. During the dissertation work, many different visualizations for the pen’s pointer were evaluated: crosshairs, arrow, and a dot the size of the pen’s LED. For the visualization at the remote site, the arrow seems to be the best choice as the digital representation is used for pointing out certain parts of the scene to other users. On the local display, the user experiences a parallax error. This is especially the case on the horizontal display, where a thick glass sheet was needed to withstand the forces when users lean on the surface. As the projection foil is fragile, it needed to be mounted on the back side of the screen. Besides the parallax error, the pilot studies (see Section 3.4.1) also indicated that users are more comfortable with visual feedback to know exactly where the pen is about to draw. The dot seems to be the most appropriate visualization for the pen in this context. Without cluttering the display, the dot indicates that the pen has been detected and also shows the color and line width of the lines the pen is going to draw.

To draw a line into the scene’s background, the pen’s coordinates have to be transformed into the scene’s coordinate system. A plane in the scene defines exactly where the user is drawing in the scene.

As mentioned earlier, a context menu is available (see Figure 6.31). By operating the rocker switch on the pen, the user can open and close the context menu. The user can initiate an action by clicking on a menu item.

6.4.3 Eraser Interactions

The eraser allows annotations and sketches to be deleted. This interaction is implemented in two steps. First, objects under the eraser are selected and then deleted. For the selection step, a special renderer was developed. The content of the window is rendered into a selection buffer. The selection buffer does not contain color information like the normal frame buffer, but contains IDs associated with the rendered objects. Depending on how the eraser is held on the display, an area of different size is used for selection (see Section 3.3.2). But unlike the deletion in Figure 3.12, an object in the selection is deleted as a whole.
6.4.4 Color-Can Interactions

As mentioned in Section 3.3.6, users should be able to change the pen’s color by dipping the pen into a device similar to an ink pot.
Two modes of operating the color can were developed. The color palette mode allows the user to choose any possible color (see Figure 6.32), while the color picker mode only shows a limited number of colors, but allows colors to be picked from the displayed image (see Figure 6.33). Pressing the side buttons of the color can switches between the two modes.

In the color palette mode, the user can select the hue value of the color by rotating the color can. A circle around the device shows all hue values and an indicator shows the selection. Inside the color can, a square area shows all possible variations of brightness and color saturation for the given hue value. The user can pick the highly saturated and bright colors from the outer circle. More specific colors can be picked from the square in the center.

Figure 6.32: Changing the pen’s color using the color-can TUI.

In the color picker mode, the color can is surrounded by a small number of colored trapezoids. Users can pick the desired color from these trapezoids. Cross hairs are shown inside the color can. If the user pushes the device on the display, the color
underneath the crosshairs will be stored to a trapezoid. The user can rotate the device to choose the trapezoid where the picked color should be saved.

Figure 6.33: Color-can TUI in cross-hair mode to pick a color from the background.

6.4.5 Handle Interactions

The handle was designed to manipulate digital cards (see Section 3.3.3). The digital cards are implemented as LayoutComponents floating over the scene in the background.

A Device object (see Figure 6.27) under the Workspace object (see Figure 6.17) contains the drawings made on the card. Therefore, when the handle is moved to a different display in the same workspace, it keeps the same drawings.

The following interactions from Section 3.3.3 were implemented:

- The user can sketch on digital cards. In such a case, the pen’s coordinates need to be transformed into the card’s coordinate system.
- If an unassigned handle is put on a card on the display, it is assigned to the card.
- If an unassigned handle is put on an empty part of the screen, a new card is created and assigned.
- If an assigned handle is put on the display, the card is shown.
- By pressing the sides of the handle, the card can be closed to prevent occluding the background (see Figure 6.35).
- Removing an assigned handle from the display will unassign it from the card.
- Removing an assigned handle while its sides are pressed, will remove the card from the display without unassigning the card from the handle.
6.4. Implemented Interactions

6.4.6 Ruler Interactions

The ruler's primary use is measuring parts of technical drawings (see Section 3.3.5). As the measurement of a technical drawing is determined by the scale of the drawing, the ruler also allows the scale of things underneath it to be changed.
Figure 6.36: Measuring floor plans using the ruler.

The ruler allows measurement using the marks of its digital representation. The distance between the marks reflect the scale of the displayed content. As with a caliper, the user can use a readout for precise measurements. Above the readout, the precise measurement is displayed (see Figure 6.36).

Figure 6.37: Press the ruler on the display to move the displayed content.

As explained in Section 3.3.5, the ruler also allows the displayed content to be moved.
6.4. Implemented Interactions

To do so, the user must press the interaction device on the display while moving it. Figure 6.37 shows this interaction. In the left part of the figure, the ruler is measuring, while in the right part, the user is moving the content to the right by pushing the top of the device. The changed color of the ruler’s digital representation indicates that it is in the state for moving the content around.

![Figure 6.37: Ruler measurement and movement](image)

As mentioned earlier, the environment allows 3D content to be visualized. The moving interaction was implemented so that it also works for 3D models. Figure 6.38 shows a user moving the model of a building towards the other side of the display.

![Figure 6.38: Using the ruler to move 3D content.](image)

Additionally, the ruler allows content to be scaled. Figure 6.39 illustrates this interaction. The left part shows the user pressing the sides of the ruler to start scaling. The ruler’s digital representation changed its color and hid the measurement marks to denote that it changed its state. The right part of Figure 6.39 shows the result of zooming and annotating a PDF document.

![Figure 6.39: Pressing the sides of the ruler to scale the displayed content.](image)

Furthermore, the user can use the ruler to rotate the content underneath. The rotation’s axis is perpendicular to the display. This interaction was implemented for 2D and 3D content. Figure 6.40 shows the interaction. The left part shows the user with the ruler in the rotation state. Note that the user presses the sides of the ruler while rotating. The
Chapter 6. DOG: Distributed Object-Oriented Groupware

Figure 6.40: Rotating the 3D content using the ruler.

digital representation changes to a different color. The right part shows the state after the user stopped rotating.

Figure 6.41: Assigning a pen to a ruler by holding it on the ruler’s edge. With regard to the ruler, the pen then only draws lines at certain angles (angle snapping).

In Section 3.3.5, one of the proposed interactions is allowing the user to only draw lines at certain angles to the ruler. Figure 6.41 shows the implemented interaction. First, the user holds the pen on the edge of ruler to associate it. The user’s name is then shown on the ruler’s digital representation (left part of the figure). Now the user can only draw lines at 45 degree angles relative to the ruler (see right part of the figure).

Using the ruler device, a lot of functionality was added to the environment. In retrospect, some of the interactions, like 3D manipulations, did not match the interaction concept of the environment.

Chapter 8 contains an alternative interaction for rotating 3D content that is more in line with what users know from paper-based work.

6.4.7 Cutter Interactions

As building dedicated TUIs is time consuming, no cutter input device was developed. Nevertheless, the interaction was implemented and tested using a normal pen. Content starts being cut from the background when the user presses the pen on the screen.
While the pressure is maintained, the pen’s movement on the display is turned into a transparently filled polygon (similar to the polygon in Figure 3.14). When the pressure is finally released, all sketches in the selection are copied into a group and shown on a new card.

### 6.5 Audio/Video Conferencing

As noted in Chapter 3, the collaboration environment should feature audio and video conferencing capabilities to support remote collaboration. In order to better support eye-contact between participants, the workspace was equipped with numerous audio and video components (see Figure 6.42, but see also [65] for more details).

![Figure 6.42: Schematic of the audio/video conferencing equipment](image)

The complete setup consists of six main components: audio input and output devices (1 and 2), video input and output devices (3 and 4), multi-user interaction tools (5), a display of the workplace (6), and processing resources (7).

Multi-channel audio/video (A/V) conferencing was realized using the Access Grid software suite (AG) [19] [20]. AG is used by many universities for remote collaboration, interactive environments, and visualization using large displays. It provides the session and computer resource management and uses existing A/V conferencing software like
the MBone software tools VIC and RAT [93] for encoding and decoding A/V signals. AG has “producer” processes that encode A/V signals and transmit them over the network. A “consumer” process receives a signal from the network, decodes it, and renders it on a display or plays it back on a loudspeaker. AG allows “producers” and “consumers” to be distributed on multiple computers to bear the load of encoding/decoding many A/V signals.

From the standpoint usability, it is desirable to have all processes run on a single computer. Due to the high load of video encoding, decoding, and the recognition of the interaction devices, multiple computers had to be used. In one setup, the detection of the interaction devices ran on a computer (2.4 GHz Intel Pentium 4), all other services ran on a second computer (Dual 3 GHz Intel Xeon). This partitioning provided minimal interference to the time critical detection of the input devices (see Chapter 4). A later setup had detecting input devices and driving the workspace display on the same machine, which has the advantage that local collaboration could be tested by running only a single machine. For the video encoding and decoding of four A/V channels in real time, a dual 3GHz Intel Xeon was powerful enough. It was equipped with a WINNOV Videum 4400 VO frame grabber extension card to acquire four video channels simultaneously. Besides a normal dual-head graphics card, an additional PNY Quadro NVS 440 with four outputs was installed to drive four LC displays.
Figure 6.43 shows the configuration of the setup. Since the loudspeakers are integrated into the monitors, a correct spatial correlation between the video and the audio signal could be realized. The monitors are arranged around the table to facilitate a more intuitive discussion between local and remote users. Because the cameras are mounted on top of the LC screens, users have better eye-contact with remote users.
Chapter 7

User Study

Evaluations of collaborative setups and interaction techniques can be done using performance, process, and subjective measures [10].

Performance measures the time users need to complete a certain task under different experimental conditions. Designing a task without biasing it towards one system is complicated. For example, pre-studies should be made to show that the task is relevant and not biased towards one specific experimental condition. Thus, comparing collaborative setups based on performance measures requires much work for designing the experiment's task. For this reason, an evaluation based on performance metrics was determined to be out of the scope of the presented work.

Process measures try to objectively quantify the communication behavior of the users. Some examples are evaluations that count how often turn taking took place or how often certain words were used in collaboration.

Subjective measures make use of the participants’ perceptions and experiences. Measures can be acquired through questionnaires, surveys, or through interviews.

For the presented prototype, the users’ qualitative feedback seemed most appropriate with regard to the development status of the environment: especially the acceptance of the setup and the proposed interaction concepts were of interest. Another goal was to compare the interaction in this environment with other systems, for example, with a traditional paper-based method and with the touch-sensitive input device “’InteracTable”. The user study should also allow the way participants work in our and in other setups to be characterized.
7.0.1 Task and experimental design

The user-study setup should meet the following requirements:

- It should guarantee that the measurement results are not biased by prior knowledge of the subjects.
- It should require multiple interaction devices to be used in a consecutive and parallel order.
- It should be applicable to all three setups.
- It should require a dialogue between both subjects.

An architectural design task was developed in which users had to perform distance measurements on a floor plan and to integrate sketches and symbols into the plan. The users had no prior experience in SDG, TUI, and tabletop systems, or in architectural design. All subjects (2 female and 20 male) had backgrounds in mechanical engineering.

The tasks were as follows:

- Measure the distance from a position in the floor plan to the nearest exit. The subjects had to mark the escape route in black and to note its length on a piece of paper.
- Measure the rooms’ surface areas. Following given rules, the subjects had to mark the positions of the needed fire extinguishers using red color.
- Measure the line of sight. Within a certain range, exit signs should be visible. Using green color, the subjects should mark the positions of exit signs.

Besides a face-to-face discussion, the task required measuring, taking notes, and changing the pen’s color. Thus, all major tools of the system were addressed. The paper-based setup consisted of a floor plan, a cardboard ruler, Post-its, and two sets of pens. For the InteracTable, the SMART Notepad software was used. Each user had a pen for interacting on the table, although the pen is a single-user input device. Since the software did not supply any suitable ruler, an additional application was used that displayed a ruler on the screen. Positioning the ruler and rotating it are accomplished by dragging the ruler with the pen. In the InfrActables setup, two pens, a ruler, a color
tool, and a handle were provided. In all setups, the subjects stood around the table in a face-to-face position. All floor plans had the same size and could be neither rotated nor zoomed.

### 7.0.2 Analysis of the experiments

The user study should help clarify the following questions:

- Is there really a simultaneous interaction on a tabletop setup during a brainstorming session?
- How good is the InfrActables setup compared to other devices?
- How well does the InfrActables setup support teamwork?

In order to answer these questions for in the InfrActables setup, position logs of the input devices were taken. Therefore for the pens, the ruler, the color can, as well as the handle their positions where logged at a fixed time interval.

Figure 7.1 shows that the pens were used all over the tabletop as the task required drawing lines and symbols. The users’ activities are balanced, meaning that both users contribute to the same degree. The diagrams also show right-handed and complementary interaction areas for the two users due to the face-to-face position. Also, the ruler was used all over the tabletop as a shared device (Figure 7.2). Again, the position log
showed the expected results since all tasks required measuring using a ruler. When the ruler was not in use, it was placed at the floor plan’s left-hand side.

Unlike the pens and ruler, the handle and the color tool were rarely moved (Figure 7.3). The almost static position of these devices can be explained by the fact that the subjects recognized the metaphors of the notepad and the inkwell. The subjects behaved correspondingly and did not move these objects around very much.

The typical usage of the devices is also reflected in the final screenshot after completing the task (Figure 7.4).

In the experiments, temporal activity logs of all devices were generated. Figure 7.5 is a representation of the users’ behavior. Since they had never worked with the system before, they randomly played around with the devices to learn how the system reacts (Minutes 1 and 2). This phase is followed by a time interval in which hardly any device is moved. During this time, the users read the task description and discussed the problem. In Minute 5, User 1 (Pen 1) started and assigned his pen to a color. User 2 (Pen 2) did the same and then moved the color tool aside. The users intuitively shared the task as one user took notes while the other did the measurements using the ruler. In Minutes 10 and 11, both users assigned their pens to new colors and started working on the next subtask. The diagram (Figure 7.5) shows the tendency of the subjects to change
roles in writing and measuring. User 1 started writing and triggered working on the task. By and by, User 2 took over the initiative and added sketches to the floor plan. Thus, the pens and ruler were simultaneously used most of the time. This was not true for the handle. Since a notepad was assigned to it, it had the character of a private device and was never passed to the other user.

The temporal activity logs showed that a real simultaneous interaction took place in the InfrActables setup, including simultaneous writing as well as simultaneous measuring and writing. This also applies to the videotaped paper-based working method. However, the requirement for simultaneous interaction was not fulfilled by the InteracTable since the touch sensitive overlay only supports single-user input. In an anonymous survey, users were asked to rate the three different setups and the way they support writing, measuring, color changing, and taking notes. The rating had to be done by ranking the different setups for the specific tasks. Finally, questions like the ones proposed by [34] were asked, for example, awareness, concentration on task, and ease of collaboration. Further, we asked the subjects for their individual system preference for daily use.

As shown in Figure 7.6, InfrActables comes close to the paper-based setup. Concerning the attribute Feeling of Writing, InfrActables is rated worst. This is due to the plastic housing of the pen's IR-LED used as a tip.

This gave an unfamiliar haptic feedback to the users, who expected the sensation of a felt tip writing on paper. The InteracTable had a more realistic feedback due to the combination of materials (felt tip, plastic surface). Performing Measurements is rated as...
better for the InfrActables than for the analog method. This might be due to the digital readout of the InfrActables’ ruler. Changing Colors is rated as better in the InfrActables than in the analog method. This could be due to the fact that changing colors in the analog setup requires a greater effort than dipping the InfrActables’ pen into the color tool. Changing colors is rated as worst for the InteracTable, because it requires opening a dropdown menu. Taking Notes was rated as best in the paper-based setup, followed by the InfrActables, which offered a notepad using the handle tool. However, this could not compete with its analog counterpart due to its limited size. The InteracTable performed worst because the software did not provide any notepad and users had to write on the floor plan.

Another outcome of the user study is the degree to which the different systems support collocated teamwork. Figure 7.7 shows the results of the user query. Here, the InfrActables environment is as good as the analog working method concerning Support in Problem Solving. This is probably because both test environments offer suitable tools for solving the task. This was not the case with the InteracTable, since a notepad was missing and the ruler was difficult to use. Concerning the Awareness of Collaboration
Partner, all systems were close together. The main reason for this is the face-to-face position that simplifies the awareness. Concentrating on the Task was rated very differently. The analog working method performed best, followed by the InfrActables. It is assumed that there are still distractions resulting from the prototype state of the system. For example the setup and the software UI is not as polished as the commercial InteracTable. The InteracTable performed worst, which is mainly due to the touch-sensitive surface that prohibits any natural interaction. Users had to concentrate on not interacting simultaneously and not touching the tabletop at two positions. For the same reason, the Support of Collaboration with Partner is poor for the InteracTable, while the InfrActables is even better than the paper-based method. This might be because measuring and color changing is slightly faster in the InfrActables than in the paper-based environment. In conclusion, the users judged the new InfrActables environment to be acceptable for daily use.
Figure 7.6: Comparison InfrActables, InteracTable, and paper-based setup (5 = very good).

Figure 7.7: Results of user query.
Chapter 8

Conclusion and Outlook

In this dissertation, a computer-mediated environment for supporting collocated and distributed work groups was presented. As the main task of users in work groups is to communicate, the system should impose as little cognitive load as possible. Users unfamiliar with the system should find it easy to interact with it: the interactions with the environment should be intuitive.

It has been shown that intuitive interactions basically mean that the user already knows them, and thus the system should leverage this knowledge. As paper-based work methods are still widely used, they should be used as a reference. Paper-based setups allow multiple users to interact at the same time using multiple tools. Therefore, this property was considered important during the development of the environment. Also with reference to paper-based setups, the environment was designed so that interactions would also work across multiple collocated displays. Besides a digital pen, TUI devices were used. TUI devices feature a digital and a physical representation that can be used to resemble the tools users know from paper-based setups.

No existing interaction technology was available to adequately support the simultaneous interactions of multiple pens and TUIs on a large display. Therefore, the InfrActables input technology was developed, which allows the identity and the state of input devices to be detected by emitting a bit code through IR LEDs mounted on the input devices. These signals are detected by a camera behind the back-projection display. The bit code is flexible and can be adapted to different numbers of input devices and device states. Besides the basic technology, several optimizations for the synchronization signal, the bit code, and the image segmentation were presented. The technology was implemented using a bit code supporting eight devices, each with four states. This resulted in a detection rate ranging between 66.67 Hz for the normal case using the 200-Hz camera and 12.17 Hz for the worst case using the 73-Hz camera. Several elec-
Electronic devices were built for the input technology. Besides a synchronization device, the following input devices were developed: a pen, an eraser, a color can, a ruler, a handle, and a laser pen.

Software packages were developed to support the work group in the environment. One software package (Tracking) allowed events from multiple different input systems (InfrActables, PS/2 mouse, GLUT keyboard/mouse, Polaris 3D tracking system, and AR-Toolkit) to be received and distributed these events based on a subscription system to other components of the software. Another software package (DOG) is the actual client application, which allows the users to sketch on digital cards and documents in the PDF, JPG, and other formats. The components of the client software and their interplay were explained. The third-party libraries used to create the client application and the principles for integrating them were presented. A data structure was presented that contains all collaboration data: session data, user interface, and the content of the discussion. With this data structure, it is possible to reuse the geometry definitions making up the scene (e.g., the tires of a car), but it is also possible to reuse identical structures (e.g., the assembly of a car). Remote collaboration was realized using replication of the data structure to all nodes taking part in the collaboration. The operational transformation algorithm synchronizes the states of the computers taking part in the collaboration. Further, the interactions implemented in the client software were presented.

A user study was carried out to assess the developed system. A task was developed that had to be solved by two test persons using paper-based tools, the InfrActables system, or a commercial touch-sensitive InteracTable. The interactions of the participants were logged and showed that participants utilized the possibility of interacting simultaneously. A questionnaire was used for getting feedback from the participants. For most interactions, users rated the developed system as second best after the paper-based setups. For some tasks like measuring or changing color, the InfrActables system was rated as first.

### 8.1 Outlook

The following sections propose possible future developments. Some suggestions aim at further refining the prototype, while others are ideas for future research.
8.1. Outlook

8.1.1 Embedded System

Integrating the input technology in a single device would be a desirable next developmental step. Such a device would perform all tracking tasks and communicate the detected interactions to a computer running the client application. The device would incorporate the camera, the synchronization flash, the synchronization electronics, and a processor that would run the detection software (see Figure 8.1). As all components would be in a single device, it could be operated more reliably.

![Figure 8.1: Combining the components of the input technology into a single device.](image)

8.1.2 Using the Amplitude of the Tracking Signal

Another improvement would be to make the communication between the input technology and the detection software more efficient. Section 4.3.4 explained how the amplitude of the signal is used to improve the motion prediction. Section 4.3.4 also discusses the difficulties of the intensity falloff caused by the position and orientation of the pen.

Cameras offer 256 to 512 brightness levels per pixel. It seems therefore promising to use these levels for encoding more data into the signal’s amplitude. An approach that might be more robust concerning the falloff problems mentioned before would be to encode data as the difference between subsequent signals’ amplitudes (brightnesses).
Figure 8.2: Data encoded in the amplitude. The signal is shown as emitted by the input device.

The device would emit data signals interleaved with the reference signal, allowing the data signals to be normalized. Four bits could be encoded in a data frame if 16 different steps of brightness compared to the reference could be distinguished. The data frames between the reference frames could also contain different bits of a longer code.

In the presented input technology, three fixed thresholds were used (see Section 4.3.4). Future input technology could employ variable thresholds that could be adjusted separately for each pen. If a variable-threshold pen appears darker in the camera because it has been moved to the border of the display, the threshold would be lowered; if the pen is tilted more towards the camera, the threshold for this particular pen would be set higher. This approach needs a reference value in order to adjust the thresholds. Using the stop bit (reference value) might not be enough, as five data bits might need to be transmitted first. More stop bits could be transmitted to address the issue. As an alternative, a technique similar to the Manchester Code could be used (for example, see [126]). As the vertical arrows in Figure 8.3 illustrate, a signal difference in subsequent frames denotes a logical 1 or 0. This requires two frames to encode one bit. Since the proposed system is still synchronized through an IR flash, it would be easy to determine which transition between subsequent frames contains the data. Because no stop bit is used in the proposed system anymore, transmitting a 2-bit state would require four frames (the current approach requires three frames). The current solution uses fixed threshold values to determine the logical value of a received signal. In the Manchester Code, the difference in received brightness values is used for the same
purpose. Therefore, the emitted difference in brightness must be just a bit higher compared to how the user would normally influence the brightness value by, for example, tilting or moving the pen. As this difference is expected to be low, the low-level signal should be detectable on the whole display. The additional data points would cause more move events to be detected. Further, the motion prediction would be updated more often and therefore make the system more robust.

![Figure 8.3: Encode data in the signal difference. The arrows denote the differences used to encode the bits.](image)

The implemented technology only communicates the state of the tracking system from the detection software to the input devices. The input technology could be further improved if this data connection supported sending data from the detection software to the individual input devices. As already mentioned, the position and the orientation of a pen affects its LED’s measured brightness during detection. Using the brightness values from the previously captured frames, the detection software could signal the devices to increase or decrease the emitted power of the LED. Every nth frame, the system would send a gain factor to each input device to adjust the device’s intensity. When the user tilts the pen towards the camera, the control software signals the device to lower the emitted power. On the other side, when the user moves the input device towards the edge of the display, the control software would require the devices to increase the emitted power. When a previously undetected device is first placed on the display, it would need some safe default settings. This approach combines the strength of the signal emitted by the diode with the brightness measured by the camera.

As mentioned in previous sections, a user tilting the pen also changes the amplitude. Further experiments would need to clarify how quickly such a movement can happen during normal writing or drawing. Comparing the speed of these movements to the detection rate of the input technology would reveal how feasible the proposed developments are.
8.1.3 Integrating Passive Optical Markers

Not all interactions need the full power of the InfrActables input technology. The position and the identity of the input devices might often be enough. Attaching an optical marker to an interaction device and detecting it using the camera would be sufficient to support such cases. Examples of such optical markers are the tags of ARToolKit (see [75]) or the Quick Response code (QR code, ISO/IEC 18004:2006) applied for interactions in [74], an example is shown in Figure 8.4). In both cases, a software library can be used to detect the orientations and IDs of multiple codes in the image.

Figure 8.4: Example QR code containing: http://icvr.ethz.ch.

For the integration into the collaborative environment, the existing IR flash and camera could be used. Instead of triggering the input device, the IR flash would illuminate the bottom side of the objects on the display. If this side of an object were tagged with an IR-reflective optical marker, the position, orientation, and identity of the object could be detected. Even transparent objects can be imperceptibly tagged using IR-reflective heat mirror foil (see [131]).

Integrating the tracking of these optical markers into the InfrActables technology could be done by using every fourth frame for detecting the markers. As no synchronization signal would be sent, this frame would not contain dots of the InfrActable devices. It would only be processed by the library to detect the markers, not the InfrActables system.

The following interactions are envisioned:

- The back side of a mobile phone would be tagged with the e-mail address or the phone number of the owner. If the mobile phone is put on the display, a snapshot
of the current screen content would be taken and sent to the owner.

- All sides of a truncated octahedron (see Figure 8.5) could be tagged with different QR codes. If the octahedron were assigned with a 3D object on the screen, the orientation of the 3D object would be changed accordingly. Changing the orientation in such discrete steps rather than continuously is expected to be more in line with the fixed perspectives known from technical drawings on paper.

- The handle device is used to manipulate virtual cards. As an alternative, a thin Plexiglas sheet could be tagged in each corner. This would allow the system to align the digital card on the screen to the Plexiglas sheet. The user could write and sketch with the existing pen on the sheet. This interaction seems interesting as it more closely resembles how real cards are handled. Nevertheless, since no state would be detected, more advanced interactions could not be implemented.

- Putting a tagged card on the display would load a specific document. As in paper-based setups, all documents of a project could be organized in a physical tray.

Figure 8.5: Using a truncated octahedron for changing the 3D orientation.

8.1.4 Experiments with the Embodiment of Devices

The usability of the input devices could possibly be improved by increasing their embodiment (see Section 3.2.6). Interactions would be more closely related to the physical
device and less to the digital representation of the device on the screen.

- The color the pen is drawing could be indicated on the pen’s tip. A multi-color LED would shine in the respective color. This would require the client application to be able to control the LED on the pen.

- Similar to the tagged Plexiglas sheet proposed before, a slim tablet PC supporting the InfrActables input technology could be used to replace the handle. Therefore, the user could draw on the tablet PC the same way he draws on the large display. Unlike the Plexiglas sheet, the device could also be operated independently of the large display. As with the transparent sheet, the drawback would be that the size of the cards would be limited to the size of the tablet PC.

The more output the devices can generate, the more bandwidth must be provided from the application to the input devices. Currently, only a synchronization signal with a low bandwidth is used. As shown in Figure 4.5, the synchronization signal occupies the communication medium for only a short period. Most of the time, the medium is unused and would be available for transmitting data to the devices.
Appendix A

Evaluation of Existing Programming Interfaces

In the following, programming interfaces (API) for user input devices were evaluated for the use with InfrActables presented in chapter 4.

A.1 WIN32

Applications written for Microsoft Windows 95 and Windows NT 3.1 use functions belonging to the Win32 API [29]. Although Windows NT 3.1 was introduced more than ten years ago, even Windows Vista is still compatible.

In order to receive input events, an application can register a “WindowProc” callback function with the type of “WNDPROC”. This user-defined function will handle the messages from the window manager. The function’s arguments are the handle of the window the messages should be received for (HWND), the number denoting the type of the message (UINT), and two parameters containing the actual data of the message (WPARAM and LPARAM). For example in the WM_MOUSEMOVE message, the WPARAM contains if one or more “virtual keys” (like CTRL, SHIFT) were pressed while the mouse has been moved. The LPARAM (32 bits) contains the x coordinate of the mouse in the lower word (16 bits) and the y coordinate in the upper word (16 bits). The WM_LBUTTONDOWN event is received, if the user presses the left mouse button while the mouse cursor is above the program’s window. WPARAM and LPARAM are the same as for WM_MOUSEMOVE.
A.2 Direct Input

Direct Input [25] belongs to the DirectX system. DirectX was developed to give certain applications direct access to the hardware to reduce the overhead imposed by OS functions. It was designed for performance critical applications like computer games. Beside support for keyboards and mice, Direct Input also supports game controllers, joysticks, and force feedback devices for entertainment. In MS Windows 98, an application could gain direct access to individual mice input streams. This technique was applied in the MID project [67]. Unfortunately, Windows 2000 did not allow this anymore.

A.3 OS Mouse Driver and Mouse Filter

Developing custom mouse drivers is another way to bypass the single user limitation. This approach seems unnecessary complicated in the presence of the Raw Input interface (see next section). But at the time of Windows 2000 there was no other way.

The approach to enable multiple users using a custom mouse driver (more concise a mouse filter [26]) was described by [140] and enabled bi-manual and multi-user interaction in the CPN project [7]. A mouse filter was developed. It receives events from lower-level mouse drivers. Unlike at the level of the Win32 API, the mouse filter receives the events with a reference to the emitting input device. Thus, at the level of the mouse filter, a multi-user application could be supported. As the mouse filter is inside the OS, it is not directly accessible to other applications. The mouse filter not only passes the received event along the regular way of the OS, but also provides an alternative way for applications to receive mouse events. Multi-mouse aware applications must use the CPN program library which registers to the mouse filter. When the library now directly receives events from the mouse filter, it converts them into a user defined window message. These messages are posted to the own application and are processed within the “WNDPROC” functions (see Subsection A.1).

A.4 Raw Input

The “Raw Input” API allows an application to get input from any human interface device (HID). It was introduced with Windows XP [28]. This API is not limited to keyboards or mice. Many other input devices like joysticks, touch screens or microphones are
treated as HID. It also allows an application to communicate with a special input device before its capabilities were fully integrated into the Windows Messaging System. An application can call the GetRawInputDeviceList to query the list of raw devices. Further the GetRawInputDeviceInfo function allows retrieving more specific information in the RID\_DEVICE\_INFO struct. In that struct, further structs are available for mice, keyboards, and other HIDs. For example the RID\_DEVICE\_INFO\_MOUSE struct has an ID, number of buttons and sampling rate field. The ID does not allow distinguishing individual pointing devices, as for example two mice have both an ID of 256. Virtual input devices, like the pointer device combining all other pointer devices, in the system show an ID of 2. RegisterRawInputDevices allow an application to register the messages from one or multiple HIDs. The messages will be passed to the “WNDPROC” function and are marked as WM\_INPUT messages. The message denotes in the WPARAM if the event occurred while the window was in the foreground or not. The LPARAM contains the actual raw input message. The header contains information about the device sending the message. The body contains a “union” of specialized “structs” for mice, keyboard, and unspecific devices. Beside normal mouse position and mouse button state variables, the raw input mouse message contains also an “ulExtraInformation” field, allowing a device driver to pass information to an application. The usage of this field is not standardized. As an example, an Alp touch pad encountered in many laptops passes uses value of 47500000 in the field to denote, if the button down event was created with the buttons beside the touch pad and not with a tap on the pad.

The SDGToolkit project [129] used Raw Input allowing the usage of multiple mice and keyboards. On top of the Raw Input, they created SDGMouse and SDGKeyboard classes. The association is done using the pointer to the input device handle, provided with every WM\_INPUT message.

A.5 TabletPC SDK and Ink API

The Tablet PC SDK allows developing applications supporting pen based interaction and speech recognition. Developed applications will run only on Tablet PCs and new Ultra-Mobile PCs.

Of course the tablet’s stylus can be used as a normal pointing device, but in addition applications can support “inking” for pen-based interaction. To enable inking in an application, the developer can build on existing controls like the InkCollector [27] or the InkOverlay. If the user draws a line with the stylus, the “Stroke” callback function of
Appendix A. Evaluation of Existing Programming Interfaces

the InkCollector will be called. Beside an object containing the geometric data of the stroke, also a pointer to a IInkCursor is provided. Among other attributes, it provides an ID, name, and tablet property. The ID property can be used to distinguish multiple input devices. If a tablet stylus with two active tips is used (one for drawing and one for erasing), the application can use the ID to determine what side was used. A gesture recognition component can be defined to evaluate the user's strokes. It will notify the control using the “Gesture” callback function.

As this input interface provides IDs and names for the pen creating an event, it seems suitable for SDG applications. Nevertheless, a test with two mice showed that the implementation is not finished, both pointing devices showed the same values for ID (1), name (“Mouse”) and name of the tablet (“\DISPLAY1”). This makes it impossible for an SDG application to differentiate the two mice for operations like assigning properties to them (e.g. line color).

A.6 Wintab API

Wintab [104] was developed in 1994 as a programming interface for digitizing tablets and other advanced pointing devices for use with Microsoft Windows Version 3.0 or higher. The API was developed by digitizer manufacturers and is normally shipped with the device drivers (wintab32.dll). A Wintab-aware application can poll for events using the WTPacketsGet function. Alternatively events are sent to the application using Windows messages marked as WT_PACKET. WPARAM (see A.1) contains a serial number the the event, and LPARAM contains the handle to the Wintab application context. The parameters can be used in a subsequent call to retrieve the event packet. This “struct” not only contains the input device’s three dimensional position, but also the pressure applied by the user, and the orientation represented as azimuth, altitude, and rotation values. Further, the type of the input device creating the packet is given. Obviously, the API is biased to be used with digitizers. But the API also defines how extensions can be integrated to support new input technologies not covered by the interface’s current definition.

A.7 Avalon API

Avalon is the presentation subsystem of Windows Vista. It features a new API for input [24]. This API was not available at the time the system was developed.
A.8 Conclusion of Evaluation

Combining pen-based and tangible interaction in a multi-user application requires a flexible event system. First, multiple input devices must be identifiable to the application. But, as TUIs are rather unnormed input devices, they require a rather flexible definition of the device state. One TUI may have no button at all, while an other TUI requires multiple buttons and a high-resolution variable.

The only way Windows XP offers, is the use of the Raw Input interface. But it provides no support for input devices going beyond the 2D pointer. This would require the use of “untyped” HID devices. Events are then basically an envelop for a custom binary protocol. The event system of XP then only serves as a communication systems, but at the overhead of developing device drivers. An advantage of developing such a driver would be that beside the binary costum protocol also the standard mouse pointer protocol could be supported. This would allow interacting with any Windows application using a single stylus of the input technology. The concurrent use of multiple input devices would confuse UI and even make the UI unusable.

The Wintab API provides an interface that suits the requirements better. Here, the orientation of TUIs is supported. Many attributes present in the Wintab events go beyond simple 2D Pointers. Unlike the well documented client API, developing a “driver” for Wintab is not documented. This would complicate the development. Functionalities going beyond a 6D pointer require the development of an extension. A drawback of Wintab is, that it is only available for Windows. It’s future is uncertain as Microsoft provides similar functionalities through the TabletPC SDK (Ink API). A key advantage of providing a Wintab driver would be that it would allow the developed input technology to be used with applications supporting Wintab. Most software packages for graphic design and CAD support Wintab, for example AutoCAD, PhotoShop, Corel Draw and so on. Nevertheless, it is unlikely that these application support the concurrent use of multiple Styli. As this is one key goal of the project, supporting this standard would be of limited use.

Non of the standard input interfaces provided what is needed. As practiced by many VR systems, it seemed reasonable to develop a custom event system at application level. This approach allows the code to be ported beyond different OSs and provides a maximal flexibility for the research on the new input devices.
Appendix B

Curriculum Vitae

Name: Christoph Schwab-Ganser
Date of birth: December 26, 1972
Nationality: Swiss

Professional Experience
05/2007 - Present  Software Developer working on video analytics and public data APIs, Google Switzerland GmbH
04/2003 - 05/2007  PhD student and Research Assistant, Innovation Center Virtual Reality (ETH Zurich)
03/2001 - 03/2003  Research Associate and project team member of ETH World. The goal of ETH World, a strategic initiative of ETH Zurich, was to develop and introduce technologies for communication and cooperation independent of time and place
1998 - 2004     Part-time freelancer as IT consultant
1997 - 1998     Intern in two architecture firms

University Education
1994 - 2000     Master of Architecture, ETH Zurich
1993 - 1994     Two semesters of chemistry, University of Basel
Appendix B. Curriculum Vitae

B.1 Scientific Publications


- R. Hofer, C. Ganser, A. Kunz; MatrixView: Improving Immersion in Videoconferencing; User Interface Software Technology (UIST) 2006; ACM Press; October 15-18, Montreux, Switzerland.


- C. Ganser, T. Kennel, A. Kunz; Digital Support for Net-based Teamwork in Early Design Stages; Journal for Design Research; Volume 1 2006; Delft University Press.

- R. Hofer, C. Ganser, A. Kunz; Tracking Technology for Multiple Device Interaction; User Interface Software Technology (UIST) 2005; ACM Press; October 23-26, Seattle WA USA.

- C. Ganser, T. Kennel, N. Birkeland, A. Kunz; Computer-Supported Environment For Creativity Processes In Globally Distributed Teams; International Conference on Engineering Design (ICED05); The Design Society; August 15-18 2005; Melbourne Australia.

- Ch. Ganser; AR-Model: Collaborative Building Visualization; First IEEE International Augmented Reality Toolkit Workshop; 29. September 2002; Darmstadt Germany.
List of Tables

1.1 Dimensions of Collaboration [99]................................. 2

3.1 Interaction Design Principles ........................................ 24
3.2 Skill Levels As They Relate to Human Mental Functions ............ 27
3.3 Results of the Pilot Studies of Two Alternative Tracking Systems ... 54

4.1 Bit Code for ID and Device State .................................. 66
4.2 Synchronization Signals Used to Query All Bits of Code ............. 73
4.3 Reduced Set of Synchronization Signals .............................. 74
4.4 Eight-Bit Code with Different Lengths for ID and State .......... 77
4.5 Using the $AND$ Bitwise Operation for Discarding Pixels Below a Threshold Value ........................................ 83
4.6 Quadlet Bitmap for Threshold Decimal 11 .......................... 84
4.7 Threshold Values with Lowest and Highest False Positives .......... 85
4.8 Pseudo Code for Assigning LEDCenters to LEDPoints ............ 86

5.1 Sample Values for the Fields of the Event Address Object .......... 114

6.1 Types of events for manipulating the data structure. GroupN::Ptr_t is the type of the pointer used to point to a GroupN object .................. 154
6.2 Implementing the hide command using a class template for member functions accepting one argument .................. 155
6.3 Script to create a rectangular surface (each line's command is commented after the # sign) .................. 159
# List of Figures

1.1 Simultaneous engineering schema with arrows to denote possible collaborations ........................................ 1

1.2 Informal working group meeting (Source: Google Switzerland GmbH) ........................................ 4

1.3 Outcome of a creativity session (Source: ICVR, Keiser 2003) ......................................................... 5

1.4 Work group and its tools (Source: ICVR) .................................................. 6

1.5 ARModel: Collaborative Visualization using Augmented Reality .................................................. 8

2.1 PC Single-User concept (a) and Single Display Groupware concept (b); source: Stewart ........................................ 12

3.1 Horizontal (left) and vertical (right) large-display workspace ...................................................... 20

3.2 Configuration of horizontal and vertical environment elements ........................................ 21

3.3 Schema of physical workspace layout .................................................. 23

3.4 Paper-based setup: Possible artifacts brought in by users include pen, paper, printed matter, and laptops ........................................ 26

3.5 Information flow in the extended SDG using multiple displays .................................................. 28

3.6 Distances in user computer interaction (Hutchins, 1985) ...................................................... 30

3.7 Photomontage to illustrate the idea of using an input device (the grey device is hereafter called handle) to control the position and orientation of a digital card. The digital card is labeled with “Bodenheizung” ........................................ 33

3.8 Dedicated TUI device for grouping sketches ........................................ 34

3.9 Interaction model of the GUI: MVC model (Smalltalk-80, modified for TUI by Ullmer and Ishii) ........................................ 34

3.10 Interaction model of the TUI: MCRpd (Ullmer/Ishii 2001) ...................................................... 35
3.11 Digital set-up: different types of pen-TUI interactions. ................. 38
3.12 Erasing with different parts of the eraser device causes different area sizes to be cleared. .................................................. 40
3.13 Handle interactions. ................................................................. 41
3.14 Card-background interactions using handle and cutter knife. ............. 42
3.15 Measuring interactions with the ruler TUI. ................................ 43
3.16 Changing the background or drawings using the ruler TUI ............... 44
3.17 Interactions with ruler and pen. .................................................. 45
3.18 Color tool interactions. ............................................................... 46
3.19 Multi-Screen interaction with the handle TUI. ................................ 47
3.20 Laser pen interaction. ................................................................. 48
3.21 Folding fan interaction for dropping documents from a pie menu to the workspace. ......................................................... 49
3.22 Pen for the use with super bright Color LEDs. ............................... 50
3.23 Demo Application showing the acquired image, reference color, and detected pen. ....................................................... 51
3.24 Setup of the pilot study using an optical tracking system for SDG. .... 53
3.25 The pilot study’s interaction devices in use ..................................... 54
4.1 Simplified bit code. ................................................................. 65
4.2 Effect of user interactions on bit code transmission. ......................... 67
4.3 Version of the bit code used in the prototype. ................................ 68
4.4 Schema of the setup. ................................................................. 69
4.5 Timing of the components. ......................................................... 72
4.6 LED topology. ........................................................................... 75
4.7 Using multiple LEDs for high-resolution state variables. ................... 76
4.8 Recognition software layers. ....................................................... 78
4.9 Principle setup of the system. ...................................................... 79
4.10 How the LED appears in the camera’s frame. From left to right: normal view, non-zero pixels colored white with a gain of 0, a gain of 260, and a gain of 680. ................................................................. 81
4.11 Segmentation of convex and concave shapes. ..................... 82
4.12 Thresholds used for manipulating LEDPoints. The dotted horizontal lines represent the thresholds. The y-axis shows the maximum pixel value of the LEDCenter. ................................................................. 88
4.13 Application to determine camera settings and thresholds. The camera frame is displayed in the background. ................................. 89
4.14 Relative brightness of a LED in a perpendicular position on the projection screen (16 by 12 measurements) ................................. 91
4.15 Relative brightness of an LED tilted toward the positive y-axis (left) or the positive x-axis (right). ................................................................. 92
4.16 Calibration pattern (left), manual calibration by holding an IR LED on projected spots (right). ................................................................. 94
4.17 Evolution of the Stylus .................................................. 94
4.18 SyncBox schema. .......................................................... 95
4.19 Typical device schema. .................................................. 96
4.20 Stylus and the PCBs of two styli. ........................................ 98
4.21 First (left) and second prototype of the eraser. ................. 100
4.22 Hardware of the color can and LED arrangement used. ........... 100
4.23 Final version of the ruler. ............................................... 101
4.24 Initial ruler prototype (left); sliding contact detail (right). ........ 102
4.25 Schematic showing constraint movement of the slider. ........ 102
4.26 Second and third (right) generation of the handle. ............ 103
4.27 Relative sensitivity of color sensor (AVT Marlin F-033C Camera). 
Source: AVT. ........................................................................ 104
4.28 Relative sensitivity of monochrome sensor (AVT Marlin F-033B Camera).
Source: AVT. ........................................................................ 105
4.29 Diffusion characteristics of projection screens at 850 nm (infrared). . . . 108
4.30 Infrared diffusion characteristics of projection screens (each curve is nor-
malized) ................................................................. 109

5.1 Schematic with the different functional parts of tracking ......................... 114
5.2 Calibration possibility of the Pointer class .............................................. 116
5.3 SDG application created with Macromedia Flash ................................... 120

6.1 Expected data flow .............................................................................. 124
6.2 Dependencies among the client software packages ................................. 126
6.3 Software layers of the system: the blue components are part of the DOG
client application; orange components are the libraries that were used
to implement DOG. Among the libraries, “Tracking” and “CGUT” were
developed within the presented work ...................................................... 128
6.4 Example graphs .................................................................................... 130
6.5 Simplified graph of a 3D scene ............................................................... 132
6.6 Variant 0PnC: nodes have only multiple children nodes. ...................... 133
6.7 Variant 1PnC: nodes have multiple children nodes and a parent node. .. 134
6.8 Variant nPnC: nodes have multiple children and multiple parent nodes. 135
6.9 Variant 1PnC1C: generic nodes share specialized leaf/core objects. ..... 136
6.10 Class diagram of variant 1PnC1L: specialized nodes share specialized
node container classes ............................................................................ 137
6.11 Example graph of variant 1PnC1L. Nodes ON 2, ON 4, and ON 5 all
share the same leaf OL 2 .......................................................................... 137
6.12 Variant 1PnC1L: controlling how the leaf is shared among nodes. ....... 138
6.13 Variant 1PnCS: synchronized GroupNodes are realized through the list
of referring Nodes in the GroupLeaf object ............................................. 139
6.14 Variant 1PnCS: example of two synchronized Groups with a FontNode
and a GroupNode as child nodes ............................................................. 139
6.15 Variant 1PnCS: to maintain consistency, calling cloneLeaf() on any node
will also clone the leaf nodes of parent nodes ........................................ 140
6.16 Variant 1PnCS: shows the effect of calling cloneLeaf() to change the bell’s knob model in a scene containing three identical bike handle bars. Red relations are reverted, whereas green relations and objects are newly created. .......................................................... 141

6.17 Data structure with nodes and associations .................................................. 142

6.18 Different architectures for synchronous groupware through sharing and synchronization of application states .......................................................... 145

6.19 Overlapping events for adding a new node to an existing node (1.B). . . 147

6.20 Inconsistency through overlapping events changing the same property of the same node. .......................................................... 147

6.21 Overlapping delete and change events are addressed by retaining deleted objects for some additional time. .......................................................... 148

6.22 Inconsistency through events arriving out of order. .............................. 148

6.23 The server receives events from clients and distributes them to the other clients. .......................................................... 150

6.24 Small arrows show how the client (red) and server (blue) move through the synchronization’s state space by sending and receiving events (green). 152

6.25 The client and server diverge by more than one event (when the server sent \( s_3, c_2 \) had not yet been processed). .......................................................... 153

6.26 The result of transforming \( c_2 (\epsilon_2) \) is used as input for transforming \( s_3 \) to \( s_3' \). 153

6.27 Scheme of the classes used to implement the interactions. ...................... 156

6.28 Schematic of the classes that react to input events. ............................. 159

6.29 Schematic of the classes that are used for rendering the UI. The numbers indicate the order of the rendering. .......................................................... 160

6.30 Annotating a floor plan using the pen. .................................................. 162

6.31 The user on the right operates the context menu of the pen. The menu is oriented towards the nearest border of the display. .......................... 162

6.32 Changing the pen’s color using the color-can TUI. ............................. 163

6.33 Color-can TUI in cross-hair mode to pick a color from the background. . 164

6.34 Writing notes on a card attached to the handle TUI. .......................... 165

6.35 Opening the card attached to the handle and writing on it. ................. 165
6.36 Measuring floor plans using the ruler. .................................................. 166
6.37 Press the ruler on the display to move the displayed content. .............. 166
6.38 Using the ruler to move 3D content. ...................................................... 167
6.39 Pressing the sides of the ruler to scale the displayed content. .............. 167
6.40 Rotating the 3D content using the ruler. .............................................. 168
6.41 Assigning a pen to a ruler by holding it on the ruler’s edge. With regard to
the ruler, the pen then only draws lines at certain angles (angle snapping). 168
6.42 Schematic of the audio/video conferencing equipment ....................... 169
6.43 Audio/video conferencing while working on the interactive table. .......... 170

7.1 Position logs of the pens. ................................................................. 175
7.2 Position log of the ruler. ................................................................. 176
7.3 Position logs of handle and color tool. .............................................. 177
7.4 Typical screenshot of a completed task. .............................................. 178
7.5 Temporal activity log of all devices. .................................................. 179
7.6 Comparison InfrActables, InteracTable, and paper-based setup (5 = very
good). ........................................................................................................ 180
7.7 Results of user query. ........................................................................ 180

8.1 Combining the components of the input technology into a single device. .. 183
8.2 Data encoded in the amplitude. The signal is shown as emitted by the
input device. ............................................................................................. 184
8.3 Encode data in the signal difference. The arrows denote the differences
used to encode the bits. ........................................................................... 185
8.4 Example QR code containing: http://icvr.ethz.ch. ......................... 186
8.5 Using a truncated octahedron for changing the 3D orientation. ............ 187
Bibliography


[111] John Rohlf and James Helman. Iris performer: a high performance multiprocess-


