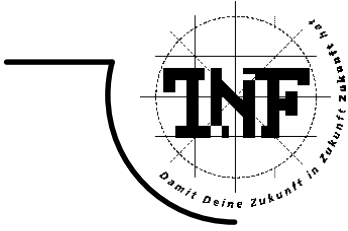




JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis



Autonomous Wearable Displays Display Unit

BAKKALAUREATSARBEIT
(Projektpraktikum)

zur Erlangung des akademischen Grades

BAKKALAUREUS DER TECHNISCHEN WISSENSCHAFTEN

in der Studienrichtung

INFORMATIK

Angefertigt am *Institut für Pervasive Computing*

Betreuung:

Univ.-Prof. Mag. Dr. Alois Ferscha

Eingereicht von:

Erhart Jürgen Martin

0255827

Linz, Oktober 2006

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum und eigenhändige Unterschrift

Abstract

This bachelor project is part of the research project SPECTACLES of the Institute for Pervasive Computing at the Johannes Kepler University, Linz. The goal of the project is to develop an autonomous, wearable computer system (like a PDA) and integrate it into glasses. Acting as output device to the user, the display unit consists of graphical chipsets, a video interface, a microdisplay and optical elements. Beside the hardware components, software is needed, to set the correct refresh rate of the image data as well as offering a way to access the display. This bachelor project documents the selection of hardware made for the SPECTACLES display unit, the available methods to merge the user's natural scene with the video output of the system as well as the implementation made to fulfil the projects requirements.

Contents

1	Introduction	1
2	Requirements	2
2.1	Software Requirements	3
2.1.1	Display Management	3
2.1.2	Application Management	3
2.1.3	Message Management	3
3	Related Work	5
3.1	Wearable Devices with Head-Mounted Displays	5
3.2	Merging Real World Objects with Virtual Objects	6
3.3	User Interfaces for Wearable Devices	7
3.3.1	Input Devices	8
3.4	Project Comparison	9
4	Hardware	11
4.1	Graphic Chipsets	11
4.2	Video Interfaces	12
4.3	Display Modules	12
4.4	Optical Elements	17
5	Implementation	20
5.1	DisplayManager Component	21
5.2	ApplicationManager Component	25
5.3	MIME-Type Resolver Component	29
5.4	MessageViewer Component	31
6	Conclusion	33
	Bibliography	35

List of Abbreviations

FPGA Field Programmable Gate Array

GUI Graphical User Interface

HMD Head Mounted Display

LCD Liquid Crystal Display

LED Light Emitting Diode

MIME Multipurpose Internet Mail Extensions resp. Multimedia Internet Message Extensions

Mgr Manager

oLED organic LED

PDA Personal Digital Assistent

List of Figures

4.1	Diagram of the display unit. From the image rendering graphic chipset to the user's eye.	11
4.2	Intel PXA270 block diagram [4]	12
4.3	Scheme of an LC display with backlight	13
4.4	Scheme of an LC reflective display	14
4.5	Scheme of an oLED display	14
4.6	Refractive prism combiner [14]	18
4.7	Folded catadioptric [14]	19
5.1	The SPECTACLES framework.	20
5.2	The dependencies of the DisplayManager.	21
5.3	Scheme of the display sectioning.	22
5.4	Sequence diagram of an application registering a window.	23
5.5	The dependencies of the ApplicationManager.	25
5.6	Scheme of the ApplicationManager window.	25
5.7	Screenshots of the running AppManager with one, two and three icons.	26
5.8	Scheme of the icon exchange when the "left" signal is received.	26
5.9	Scheme of the icon exchange when the "right" signal is received.	27
5.10	Sequence diagram of an icon registration and notification with the application manager.	28
5.11	The dependencies of the MIMEResolver.	29
5.12	Screenshot of the MessageViewer component displaying a message.	31
6.1	Scheme of the optical path including ambient light.	34

Chapter 1

Introduction

During the last years, the development of microdisplays, compact processor- and chip-designs rapidly advanced. With an increasing demand for mobility in work and lifestyle, handheld devices became very popular [13]. Further explorations in this field are necessary, to satisfy this demand.

This bachelor project is part of the research project SPECTACLES of the Institute for Pervasive Computing at the Johannes Kepler University, Linz, Austria. The goal of SPECTACLES is to develop an autonomous wearable computer system (like a PDA) and integrate it into glasses. While the project is merely a case study on how an autonomous system can be integrated into glasses, there are of course interesting fields of use. As shown in other research projects, it is possible to increase the safety of a person during her workday with wearable devices and displays, as Wilson et. al. [19] showed with their work of increasing the safety and efficiency during indoor firefighting, using a head-mounted display. Also the consumer electronics can benefit from such devices, since this market is a major consumer of microdisplays [9].

The SPECTACLES project is split into several subprojects, each one processed by a different person. They are developed parallel as components, and a component can use others to fulfil it's task. One of these subprojects is the display unit, which I will describe in detail in this bachelor project. It contains several hardware parts, like a microdisplay and optical elements, and software components to manage the display and its power consumption.

In the remainder of this document, the project goals and implementation are presented. Although the specifications of a PDA, and the functionality of SPACTACLES seem to be similar, the requirements for displays, batteries, power management, etc. are not. The requirements of the system are described in Chapter 2. These are not only hardware specific, but contain informations about the SPECTACLES software requirements too. Before the hardware and software details are described, related projects are explored in Chapter 3, as re-use of other projects might shorten the SPECTACLES development. To create a common starting point for further details, the hardware layer is described in Chapter 4. The hardware is chosen to satisfy the requirements of SPECTACLES and with the main design goals in mind. Followed by a description of the SPECTACLES software layer with the component framework, Chapter 5 gives an overview of the software system and describes the implementation made for the SPECTACLES display unit.

Chapter 2

Requirements

The goal of the SPECTACLES project is to create an autonomous wearable system, which is integrated into glasses. Therefore, the system should not be too heavy or have a clumsy design. Although the physical design of the glasses is neither a question of software development nor a part of this bachelor project, the used hardware is influencing it, as the hardware parts have to be integrated into the frame of the glasses. It is impossible to use high performance parts, like latest high-end processors and graphic-cards, as they are too large and producing too much heat.

Acting as an output device to the user, the display unit is used to display the system status and additional information, which are of user interest. The physical size of the display should be small, whereas the pixel resolution of the display should be high to be able to create high quality images. Although it looks like both of these requirements cannot be fulfilled at once, a microdisplay is able to do so. The typical size of a microdisplay is about 15 x 15 mm, with pixel resolutions up to SXGA (1280 x 1024 pixels). Depending on the type of display, additional hardware is needed, which is described in chapter 4.

The following list contains the most important requirements of the display unit:

- high resolution
should be at least QVGA (320 x 240 pixels), preferably XGA (1024 x 786 pixels) or higher
- low power consumption
- a single supply voltage
- easy adaptable control- and video-interfaces
e. g. RGB sequential input
- low weight
the display itself is very lightweight but glass prisms and lenses are heavy
- small dimensions
the display unit should fit into the frame of SPECTACLES

The requirements above are in line with the requirements for the whole project. Table 2.1 shows these requirements, which have to be applied to every hardware component of SPECTACLES. In addition to these requirements, SPECTACLES needs software to function.

The necessary software components for the display to fulfil the defined project goals are described below.

2.1 Software Requirements

As stated in the introduction (Chapter 1 on page 1), the SPECTACLES software consists of multiple components, implemented by different persons. Each component is either responsible for a part of hardware, e. g. the Display Manager for the physical display, or fulfils a task to ensure the functionality of the system. This component framework should support later extensions and implementations of new components as far as possible, with simple interfaces. Since the hardware resources are limited, all components have to be optimized in performance.

2.1.1 Display Management

It is necessary to control the access to the physical display. The SPECTACLES device lacks of input devices which are available on a desktop computer or a PDA, because a conventional keyboard or mouse is not available. The design of the graphical user interface must therefore consider the very reduced ability of the user to interact with the system. In addition, most of the available microdisplays, which can meet most of the requirements for the SPECTACLES, have a low resolution (320 x 240 pixels). Consequently, it is not a good solution to use multiple windows on the screen at the same time, as the user might not be able to change the focus of the windows. Knowing these facts, it is clear that an application or component should be able to control the entire, or a large part of the display at once. All other applications have to wait until the region is released. A component is required which manages the access to the display. This component should register all applications which like to use the display and grant access to only one application at a given time. To support future development of applications, this component should offer classes for simple creation of application windows which are taking response for the registration and communication with the display management component.

2.1.2 Application Management

The user should be able to launch applications herself. This requires a component which visualizes available applications, and offers a way to start them. This component must operate with a minimum of user input, as the SPECTACLES cannot use conventional input devices. Unlike current desktop management systems, which are using the whole display area to display the applications, this component should take less space, to have a large portion of the display free for application windows, etc.

2.1.3 Message Management

A Bluetooth interface is used to receive data from other devices, like PDAs, mobile phones or sensors. It should be possible to receive messages from other mobile devices and display them on the SPECTACLES. A single or multiple message viewer components are necessary, capable of displaying different message types, e. g. text, images, videos. To reduce the processor load, only one message viewer should be active, which is needed at the moment. An additional component is needed, to identify the message viewer which is suitable for a specific message. This mechanism can use MIME types to identify the correct message viewer component.

Table 2.1 shows the requirements for the SPECTACLES device, defined by the specification of the research project.

Category	Specification	Resulting Requirements
electronics	The SPECTACLES should have a built in power supply for the integrated hardware components. Additional battery packs, e.g. worn by the user on a belt, are not desired. The battery lifetime should be as long as possible. Communication with other SPECTACLES and sensors is required.	low power consumption, few different supply voltage levels, scaleable power consumption, deactivate-able hardware components, wireless communication
physical	All hardware components have to be integrated into the frame of the SPECTACLES. Additional space is not available, i. e. an additional device, worn by the user, with parts of the SPECTACLES hardware is not desired. The user should be able to wear the glasses for long durations.	lightweight components, small components, low heat dissipation, robust design, comfortable fitting
software	The software components must support the electronic requirements and regulate the hardware power consumption, if the hardware is not capable of regulating itself. The software should be extendable by new modules.	performant software, low memory utilisation, battery level surveillance, low system load

Table 2.1: The requirements for the SPECTACLES project

Chapter 3

Related Work

There are various fields of use for the SPECTACLES device. For example, a user can wear the system, which supports her with information of importance to her actual tasks. During the construction or the repair of vehicles, airplanes, etc. workers have to leave their working place to take a look at the construction plan or lookup specific information. With SPECTACLES and suitable software components, it should be possible for workers, to view the plans directly at their working place. With special software, SPECTACLES could act as a tourist guide, and supply the user with information about cities and sightseeing, bus and train connections and timetables, as well as other information of interest for tourists. In cars with navigation systems, SPECTACLES could be used to display the route by arrows or highlighting the correct street, without causing the user to look at the small screen of the navigation system, thus increasing security for the user.

These are just a few examples where SPECTACLES could be used, but all examples have one thing in common. The user is mobile within her environment, and this must be considered in the development of the display unit as well as in the behaviour of the components using the display.

3.1 Wearable Devices with Head-Mounted Displays

A head-mounted see-through display gives the user the ability to see the natural scene combined with the output of a computer system. This augmented reality is useful, as it can support the user with additional information which would be unavailable without the display system. As stated in the example above, a head-worn display can help a user to fulfil her job more efficiently. But it must be noted, that the head-worn display must be carefully designed. Geelhoed et. al. [6] analyzed in their work, if head-mounted displays have a major influence onto the human perception, causing changes to the visual system. They developed a prototype head-mounted display and compared it with a Sony Glasstron head-mounted display. After evaluating the results of three experiments with various people, they concluded that a head-mounted display does not cause changes to the visual system, like nausea, but cause a vergence look, caused by the fact that the eye is focused onto one distance over a long time. Therefore, complete occluding head-mounted display, i. e. HMDs without video/optical see-through technology, should not be used for a long time, as the eye cannot change the focal distance. The focal distance of the display, i. e. where the display appears to be, suggested by optometry experts, should be at three meters. A too short focal distance can mean, that people with reading glasses, due to presbyopia, cannot focus onto the display. If the display should be used for reading texts, it is required that the edges of the display are as sharp as the center of the screen. Special lenses have to be used for such purposes.

In their research for the Fire Information and Rescue Equipment (FIRE) project, Wilson et. al. [19] evaluated various Head-Mounted displays and integrated them into a firefighter

mask. The FIRE project aims to give firefighters a system of information technology tools for safer and more efficient firefighting in large buildings. The final prototype was used to display a map of the building. The firefighter can lookup his position within the building and view a floorplan with his current position highlighted. The firefighter is located by a wireless sensor network, which is integrated in small beacons. If a building does not have such beacons already, the firefighters can deploy them as they are small enough to put them into the equipment bag of a firefighter. The commander of the firefighters can see the position of the firefighters on a laptop to coordinate them, without asking them of their current position over radio. The evaluation of displays was made under the criteria, that the display must not restrict the firefighter during his work and the firefighter must not be distracted by the display. Thus it was clear that only a head mounted display with small dimensions can be used, preferably integrated into the firefighter mask, which protects the firefighter from smoke, hot air and debris. During the evaluation, firefighters of different firedepartments were asked to test the prototypes and comment about the pros and cons of them. It showed, that the display must be robust and should not occlude the field of view. In addition, the display should be useable without the need of an input device, as the firefighters need their hands free. The final prototype was integrated into the firefighter mask with an optical see-through system.

During their research, Hua et. al. [8] evaluated head-mounted projective displays for collaborative environments. Since current available collaborative environments, like the CAVE or a curved screen are projecting geometric correct images only for one user, or users standing close to each other and viewing from the same direction and angle – as the eye position of a user is used for the calculation of the stereoscopic image positions –, the goal of their work was to create a similar environment, but with geometric correct images for multiple users. Instead of using a traditional head-mounted display system, they developed a projective head-worn device, which does not project the image directly into the users eye, but onto a retro-reflective surface in front of the user. Due to the attributes of the retro-reflective material, the light is directly diverted back to the user, whereas a diffuse projection surface diverts light into all directions. With their head-mounted projective display (HPMA) prototypes, it is possible for multiple users to view images from different angles and positions, without distortions as they occur in the CAVE or on curved screens.

3.2 Merging Real World Objects with Virtual Objects

In an augmented reality system, virtual objects are merged with the users natural scene, either by optical- or video-see-through systems. An example scenario for SPECTACLES is a navigation support system. For example, the output of a navigation computer, e. g. in a car, is used by the SPECTACLES device to highlight the street where the user has to drive. It is necessary, that the device output is calibrated with the users view, to highlight the correct street.

Tang et. al. evaluated four different variants of the single point active alignment method (SPAAM) to calibrate an optical see-through head-mounted display for a correct placement of virtual objects in the natural scene [17]. During their tests with 21 subjects in 4 different trials, they measured the accuracy of the calibration and compared the results, and found out that the human factors contribute significantly into the accuracy of calibration. They

concluded, that this fact has to be considered by developers of optical see-through head-mounted displays with augmented reality applications.

Genc et. al. [7] used vision based trackers and a camera to calibrate the virtual output towards the natural scene. By using the SPAAM method where a camera is used to capture calibration points in the environment – instead of the user, who points at specific object or point using a crosshair displayed on the head-mounted display – they reached very accurate results. Retroreflective markers were placed in the environment. An infrared light source, placed near the camera, is used to illuminate the markers without creating light dots perceptible by the user which may be disturbing.

In their ARQuake project, Thomas et. al. [18] developed an augmented reality application with a first-person perspective. The project goal was to construct a first-person perspective application where the user is able to walk in the information space, the point of view is completely determined by the user's head and the virtual environment is actually set in the physical world. By using the graphics engine of the computer game Quake, GPS and other positioning sensors, they constructed an environment where the user is able to interact with the virtual world and can move around free in the physical world. A see-through head-mounted display is used to merge the real with the virtual world. The project was intended to present the abilities of augmented reality, and that it is possible to use augmented reality with inexpensive, off-the-shelf software.

3.3 User Interfaces for Wearable Devices

Schmidt et. al. [16] mentioned in their work, that the graphical user interface of a wearable device must not restrict or occlude the user's field of view, as this can lead to injuries. In addition, the input device of the interface should be easy to use and simple to understand, because the user is mobile in her environment. The design of the user interface in their project considered this. All selectable items are placed on the top, bottom, left and right side of the display. The center of the display is free of image output, thus enabling the user to see her environment while navigating through the menu items of the interface. Because of this border layout, the menu gets out of focus and disappears in the user's field of view, when she is focusing onto objects in the real world. To navigate in the menu, a single input is needed. Another input signal is used to select the highlighted item. Submenus can be realized too as all menus are restricted to a maximum of 8 items.

Campbell and Tarasewich [2] showed with their work, that even a very small amount of output can be used to communicate a lot of information to the user. By using three LEDs with different colors, which can be illuminated in different intensities, a total of 108 different states – that's an equal of 6.75 bits – are possible. These states are used to inform a user, that she received a message. By running trials with different persons, they found out, that even 108 different states can be distinguished after a few runs to learn the meaning of each LED and intensity. This indicates that very small displays can be used to present up to 6 bits of information, without long learning trials for the user.

As the input device capabilities for mobile and wearable devices are drastically reduced, compared to a standard desktop computer, new communication channels need to be explored. Sandor and Klinker [15] developed a prototyping software for user interfaces in ubiquitous

augmented reality. Their distributed wearable augmented reality framework (DWARF) is a system used to connect different input and output devices with an application. It is possible to use local and remote sensors and input devices, as the DWARF distributes input and output signals via a peer-to-peer network connectivity. New devices are found, configured and included into the DWARF network automatically. The system with its fine granularity and loose coupling is flexible, and because of the implemented communication protocols, it is suited for real-time applications. As new input devices can be added by implementing a suitable media analysis component, the DWARF system is extensible towards inputs devices which are not yet developed.

3.3.1 Input Devices

During the evaluation of different pointing devices, Card et. al. [3] realized that a head tracked pointing device is too unaccurate for exact pointing. If the target, where the user should point at, is too small, it is not possible with a head tracked device, as the muscles in the neck have a too low angular resolution. A head tracked device can be used for pointing at large targets, or moving the pointer fast between objects, but to point exactly at an object, additional pointing devices are required.

An important attribute for a graphical user interface of wearable devices was considered by Lumsden and Brewster [10]. The user of a normal desktop computer can use all her senses to interact with the computer. A mouse and keyboard can be used, the eyes are focused onto the display and the user sits still in front of the computer. On a mobile device however, the user does not have all her senses available for the device, as she might be walking while using the device. Ignoring this, can cause serious injuries, if the device is dragging all of the user's attention onto itself, the user might run into obstacles. By using hand gestures and audio feedback, they created a user interface with a very small amount of visual interaction requirements. It is not necessary for the user to focus onto the display for selecting menu items.

Audio feedback is also used by Brewster and Cryer [1] who tried to use the few display space on a mobile device efficiently. As mobile devices lack of large screens, the interaction elements have to be small, in order to place enough of them onto the display, to let the user interact with the system. Yet, too small elements may not be identified by the user, which makes the selection of elements hard or even impossible. Thus the information which can be displayed by mobile devices is limited. To increase the information output, Brewster and Cryer used small buttons in combination with audio feedback. The result of their experiment showed, that audio feedback can help the user to find and use an element, even if it is small, whereas small elements with no audio feedback result in higher workload for the user.

Common user interfaces for wearable devices like mobile phones are using timeouts to change the functionality of buttons or menu items. This is necessary, as each button has multiple functionalities, and the user interface needs a way to determine which function is requested by the user. Marila and Ronkainen [11] evaluated the length of timeouts and the adaption of the timeout length towards the user's response time. In different trials they provided none, auditory and visual feedback of the timeout and compared the results, especially the learning effects of the timeout length were analysed. Their result was, that a specific timeout length is memorised by a subject for the time during the trial, but there are no long time learning

effects. Because of that, adaptations of the timeout length towards the user's response time, should be made only when the device is not actively used.

3.4 Project Comparison

The projects presented above are containing interesting aspects for the development of the SPECTACLES device. Their advantages and disadvantages in correspondence to SPECTACLES requirements are listed below, in the order they were presented.

Safety of Eyeglass Displays

As Geelhoed et. al. [6] mentioned, it is necessary to design the eyeglass display carefully, with safety aspects in mind. When a user wears the display over a long time, her eyes are locked onto the optical distance of the display, i. e. after the use of the display, the eye is not able to focus precisely onto the natural scene. A similar effect can be observed when we read a book over a long time.

Although SPECTACLES has a fixed optical distance, it is an optical see-through display in difference to the occluding display, used by Geelhoed. This allows the user to see the natural scene while using the display, which may reduce this effect. Additional research would be necessary to test this behavior of the eye with a prototype of the SPECTACLES device.

Fire Information and Rescue Equipment

The FIRE project by Wilson et. al. [19] showed how the security of a firefighter can be improved by using head-mounted displays. Their evaluation of displays based on similar criterion as SPECTACLES, though they used a battery pack and additional hardware to supply the display with power and image data. However, the SPECTACLES device needs to be designed as a single piece of hardware, with no additional parts attached via wires. Therefore, displays requiring different supply voltage levels cannot be used, as the production of these voltage levels will increase the size of the power supply electronics. High energy consumption is an issue too, as a battery pack is not available, as it is in the FIRE project.

Head-Mounted Projective Displays

The project of Hua et. al. [8] is very interesting for a collaborative environment, as each user has a correct, three dimensional image, or can view the scene from a different direction without having distorted images. However, as SPECTACLES is aimed to be used both indoor and outdoor, it is impossible to use a projective display, as the whole environment cannot be covered with retroreflective materials.

Head-Mounted Display Calibration

As mentioned before, if virtual output needs to be inline with the natural scene, the head-mounted display has to be calibrated. Without a calibration, objects will be misplaced and may become distorted. As Tang et. al. [17] showed, a user driven calibration of the system is possible, and leads to good results, but as a large amount of human factors influencing this calibration process, I suppose to use a camera for calibration instead, as used by Genc et. al. [7] in their project. A main drawback on this approach is the additional need for a camera, and some kind of markers, visible for the camera, to calibrate the system.

ARQuake

The ARQuake project shows the potential of see-through head-mounted displays. A very

important fact for outdoor displays was mentioned by Thomas et. al. [18] as only bright colors should be used in such a display, because the image cannot be seen in daylight, if the colors are too dark. A black image appears as transparent. This is an important point for the colour selection for the user interface on SPECTACLES. For example, black text will be very hard to read, as it appears transparent for the user, i. e. the user can see parts of the natural scene instead of the black letters.

User Interfaces

The user interface of Schmidt et. al. [16] is designed with the fact in mind, that the interface should not occlude the users field of view. The interface is placed at the edges of the display, to move it out of the focus when the user is looking at the natural scene, as it is constant displayed. The user interface of SPECTACLES however needs to be deactivated, when not in use, for power saving purposes. Therefore, the user interface is only visible if the user is interacting with the system, and a placement outside of the main focal area is not required. Whenever the user is not interacting with the system, a small display area can be used to notify the user of system events. As Campbell and Tarasewich [2] concluded in their work, only three LEDs can communicate up to 6.75 bits of information.

Another important aspect for the user interface design of SPECTACLES is the fact, that suddenly appearing pop-up windows should be avoided as much as possible, as they can distract the user in situations where she needs her full attention towards the natural environment. This aspect was a main criterion for Lumsden and Brewster [10] to find new forms of input devices. Audio feedback, also used by Brewster and Cryer [1], is used to guide the user through the items of the interface menu. Since audio output is not available at the time this bachelor project was created, it is not an option. Although it should be considered for further research and development.

Marila and Ronkainen [11] experimented with different length of timeouts, to adapt towards the user's response time. Currently, it is not planned to enter text on the SPECTACLES device, a timeout similar to those used in mobile phones for text writing is not necessary, but timeouts are very useful when the display needs to be deactivated. The length of the timeout should be dependend on the actual available powerlevel.

Except for the DWARF framework of Sandor et. al. [15], all related projects are concentrating onto a specific task. They are optimized towards their field of use, which results in good results, but may lead to issues if the systems should be used in fields where they are not initially designed for. The main goal of the SPECTACLES component framework is to be as versatile as possible, by using as less ressource as possible, to offer a lot of functionality. This means, the framework and the core components themselves should not be designed for a specific task.

The DWARF framework can be used to design user interfaces with various input and output devices, and provides an interface to applications. The SPECTACLES component framework is similar, but it integrates every piece of hardware, not only input and output devices.

Chapter 4

Hardware

The hardware components used for SPECTACLES were selected at the beginning of the research project. The main criterion used to find suitable parts was low power consumption, as the battery lifetime should be maximized through intelligent use of resources. The battery capacity influences the weight of the SPECTACLES device as large and durable batteries are heavy. Another important criterion was the supply voltage. The hardware components should use as few different voltage levels as possible, because additional electronic is required for each level. That would increase the amount of hardware which has to be integrated into the frame of SPECTACLES. Figure 4.1 shows a diagram of the SPECTACLES display unit. To display an image, the graphic chipset converts the graphic data into signals which are sent to the video interface of the connected display. The video interface converts the signals of the graphic chipset into signals, which can be used by the display module. The display module converts the electric signals into light signals which are projected through the optical elements to the user's eye.

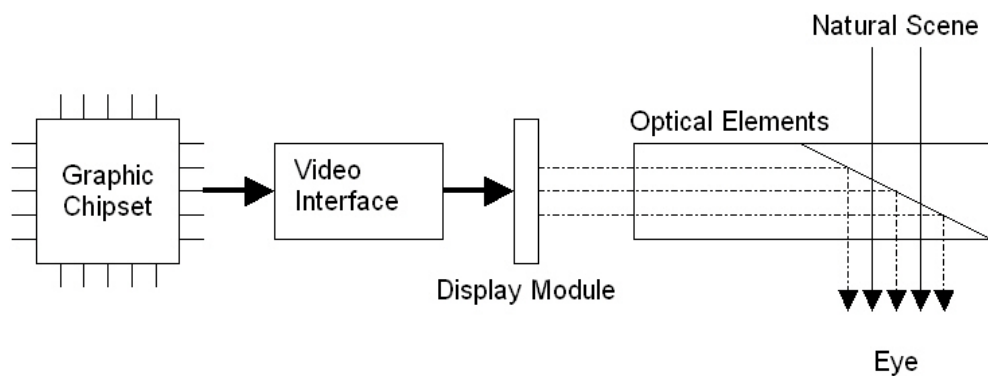


Figure 4.1: Diagram of the display unit. From the image rendering graphic chipset to the user's eye.

4.1 Graphic Chipsets

The central processing unit of SPECTACLES is an Intel PXA27x with an integrated dual-panel LCD controller. The controller is capable of hardware colorspace translation from the common video decoder output format YUV4:2:0 to RGB. Most microdisplays accept RGB input which makes the need for a conversion from YUV4:2:0 to RGB necessary. While software conversion requires additional CPU cycles, the hardware conversion works without the CPU, increasing video decoding performance [5]. Since an LCD controller is already integrated in the CPU, further graphic chipsets were not investigated, as additional chipsets

would increase the physical size of the processor board, but small dimensions are desired for SPECTACLES.

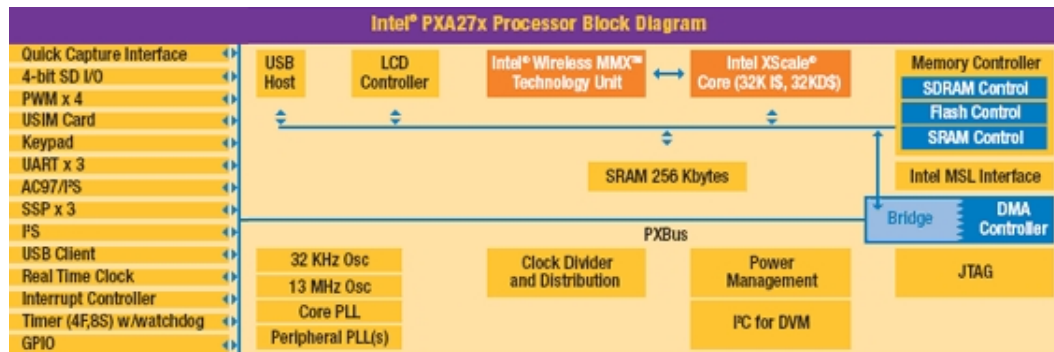


Figure 4.2: Intel PXA270 block diagram [4]

4.2 Video Interfaces

As stated above, most microdisplays can accept RGB analog input, but the format of the input is not standardised. Because of this fact, an interface circuit is needed in some cases, to convert the RGB output of the PXA270 CPU to the specific microdisplay format. Most microdisplays can accept standard VESA, DVI or NTSC/PAL signals as input, which makes them very useful for a desktop application, as current available graphic cards use these formats. Those microdisplays need additional interface electronics to convert the VESA, DVI or NTSC/PAL signal into an analog RGB signal. Such an interface can be realized with an FPGA for example, which can be used to change the clock signal as well as the sequence and frequency of the color signals. They require additional supply voltages and are increasing the power consumption.

Since the PXA270 is capable of generating RGB output signals, it would be counterproductive to convert these signals to a standard VESA, DVI or NTSC/PAL signal, to be able to use microdisplays which can accept these signals as input. Unfortunately, most microdisplays are designed to accept these video signals as they are used in television, camcorders or digital cameras. Consequently, a microdisplay is needed, which can accept RGB signals as input, and does not require much interface electronics.

4.3 Display Modules

For SPECTACLES, small displays are needed, because the available space is very limited. With this fact in mind, microdisplays seem the perfect match, since they are very small and have low power consumption. The setup of microdisplays can be distinguished in the following three types.

- Liquid Crystal Display with backlight

A transmissive LCD-based microdisplay is combined with a backlight to project an image. The backlight is the main power consumer in this system and when switched off, an image is not visible. The display consists of two polarisation filters and the liquid crystals on a thin film of transistors. With voltage applied to them, the crystals are regulating the amount of light which can pass the polarisation filters. Colored images are achieved in two different ways. Frame sequential coloring uses three different LED colors to rapidly display a red, green and blue image (typ. at 180 Hz per color frame), which are combined in the eye to a full-color image. Micro-displays with per-pixel color filters are using red, green and blue color stripes at the width of a single transistor element. This display type needs only a white backlight.

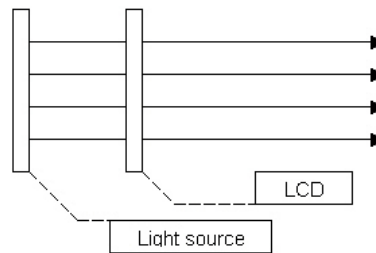


Figure 4.3: Scheme of an LC display with backlight

- Liquid Crystal on silicon reflective display (LCOS)

An LCOS display consists of a film of liquid crystals (similar to an LCD microdisplay) which is placed over a reflective surface. The liquid crystals are regulating the amount of light which is reflected by this surface. As shown in figure 4.4 on the next page, a half-silvered mirror is placed above this display combination. Light, falling in from above the half-silvered mirror passes it, and is reflected back to the half-silvered mirror by the reflective surface. In this direction, light cannot pass the half-silvered mirror, and is deflected. This kind of display needs a light source of the colors red, green and blue. There are two approaches for the light source. Like Frame sequential color for LCD displays, the light source consists of three LEDs (red, green, blue) and creates red, green and blue images, which are combined to a single full-color image in the eye. The other approach is a “colorwheel” in front of the display. The “colorwheel” consists of three transparent foils in the colors red, green and blue. To project an image, a white backlight is used. It transmits light through the “colorwheel”, which changes the color of the light to red, green or blue, onto the reflective display. Compared to LCD displays with backlight, reflective displays have much higher resolution and better contrast. This technology is mainly used for video beamers, and it requires a relative large optical setup.

- oLED active matrix display

The oLED display is a technology which uses organic polymers. These polymers produce light if connected to an electric energy source. While providing very efficient and color-intensive images, the main limitation of oLED systems is the rapid aging of the organic materials: Red and green oLEDs have a lifetime of 5,000 to 10,000 hours, whereas blue oLEDs have a lifetime of only 1,000 hours. The power consumption of an oLED display is much less than the consumption of an LCD or LCOS display, as additional backlights

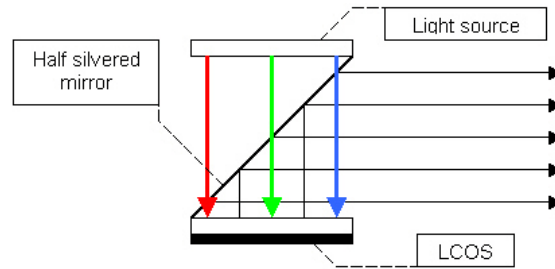


Figure 4.4: Scheme of an LC reflective display

are not needed. Because of this advantage, and the brighter colors and higher contrast in comparison to LC displays, the oLED will be a future replace of the LC display as soon as the aging problem has been solved.

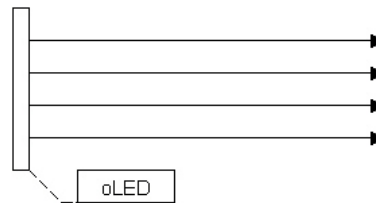


Figure 4.5: Scheme of an oLED display

Head-Mounted Displays

To develop a display unit for SPECTACLES either an already available head-mounted display or microdisplay components can be used. Several HMD modules exist that can accept VGA input.

	<i>Resolution (in pixels)</i>	<i>Power utilization</i>	<i>Interface</i>	<i>Power supply</i>
Microoptical CV-3	640 x 480	2 W	RCA connector, NTSC/PAL signal	7.2 V
Liteye LE500	800 x 600	500 mW	15pin VGA, NTSC/ PAL or serial RGB	6 V

Table 4.1: List of evaluated Head Mounted Displays

The CV-3 is the newer, marketable, version of the CO-3 which is a single microdisplay component, and was suggested for a first hardware prototype of SPECTACLES, as it is already available at the Institute for Pervasive Computing. The package contains the optical viewer, a rechargeable battery pack, a battery charger, various mounting systems and a users

manual. The CV-3 is plug and play with any strict VGA sources and applications, including wearable computers, notebook displays, medical imaging and test instrumentation.

<http://www.microopticalcorp.com/Products/video.html#CV3>

The LE500 from Liteye is a helmet mounted oLED display. As shown in the comparison chart, the power consumption is much less than the CV-3 LCD.

<http://www.liteye.com/Default.aspx?tabid=47>

Microdisplay components

For developing a custom display unit, multiple components are needed: The display itself, a backlight module in case of LCD based technologies, and optical elements (lenses, mirrors, prisms) that merge the display output with the optical path of the eye (see Section 4.4 on page 17 for details). The following table 4.2 is a comparison chart of available display modules under consideration of the SPECTACLES requirements.

	<i>Resolution (in pixels)</i>	<i>Power utilization</i>	<i>Interface</i>	<i>Power supply</i>
Brilliant Z86D-3 18 x 14.2 x 3.1 mm	800 x 600	N/A	LVDS/TMDS	5 V, 6 V
CRLOpto SXGA-R2-H1 60 x 37 x 3.1 mm	1280 x 1024	N/A	DVI,	N/A
Displaytech LV311-DN 20 x 18.8 x 11.4 mm	420 x 240	175 mW	CCIR601, CCIR656, RGB serial	2.5 V, 3.3 V, 5 V
eMagin SVGA+ rev.2 19.8 x 15.2 x 5.1 mm	800 x 600	200 mW	VESA/VISIS	-3 V, 3.3 V, 4 V
Kopin CyberDisplay 230k LV 6 x 9.8 x 1.4 mm	320 x 240	14 mW	display specific	3.3 V
Kopin CyberDisplay 922k 11.6 x 11.46 x 1.4 mm	640 x 480	85 mW	display specific	3.3 V, 9.5 V
Microdisplay MD832G9 Curtis 17.5 x 22.5 mm	800 x 600	60 mW	N/A	5 V
Microemissive ME3203 13 x 12 mm	320 x 240	45 mW	Serial RGB, BT.656	-2 V, 3.3 V

Table 4.2: List of evaluated microdisplay components

The Brilliant Z86D-3 display is of reflective design. It needs a red, green and blue LED as light source, and a half silvered mirror to work properly. This display is used by Brilliant for projection televisions.

<http://www.brilliantcorp.com/products/nearEye.html>

CRLOpto uses also a reflective design for its SXGA-R2-H1 display which needs the tri-color LED light source and the mirror. Therefore the deepness of 3.1mm is not correct, because the dimensions are only from the display itself. The display needs a special adapter board where the powersupply is included.

http://www.crlopto.com/products/reflective_list.htm

The LV311-DN from Displaytech is a reflective display with a built-in light source and a half silvered mirror. The display can be used as is, without adding additional LEDs and has a very high resolution. The main drawback of this display is the size.

http://www.displaytech.com/products/personal_view/index.html

The SVGA+ rev.2 is an oLED active matrix display from eMagin. It needs no extra backlight or lightsource but a negative supply voltage to control the brightness of the display. Because of its resolution and standard VESA interface this display seems a good choice for the SPECTACLES.

<http://www.emagin.com/svgaplus.htm>

The two Kopin displays listed above are LC displays. They need a white LED as backlight which will increase power consumption drastically. The 230k LV model needs only a 3.3V power supply, but its resolution is not high enough to meet the SPECTACLES requirements. The 922k display model however has a high resolution, but needs an additional 9.5V power supply. Therefore these two displays are not the best choice for the final version of the SPECTACLES.

http://www.kopin.com/products/index_cyberdisplay.html

Another reflective display based on LCOS (liquid crystal on silicon) technology comes from Microdisplay. The MD832G9 Curtis needs the additional tri-color LED light source and a half silvered mirror. While this increases the space the display needs on the SPECTACLES, LCOS based displays have better resolutions and higher contrast compared to LCD backlight displays, so they may be a better choice if it comes to resolution and contrast.

<http://www.microdisplay.com/?a=ehd>

The ME3203 is an oLED active matrix display from Microemissive. Although of its small power consumption and the single supply voltage of 3.3V, this display is not suitable for the spectacles need for a high resolution. To control brightness, an additional negative supply voltage is needed. To create this voltage, only a few parts are needed, which are connected to the display itself. The control of the brightness is done using the control bus of the display.

<http://www.microemissive.com/>

With the requirements for the display unit in mind, the best choice would be a microdisplay with high resolution, low power consumption and the need for only one power supply voltage. In addition, a standard input interface is needed, to keep interface electronics as small as possible. As shown in table 4.2 on the preceding page, this is not quite possible with the actual models of microdisplay components.

The main power supply voltage of SPECTACLES is 3.3 V, as the main voltage of the used battery will have a fixed value of 3.3 to 3.5 V to keep the battery pack small and lightweight. Therefore it is important that the microdisplay uses voltages equal or lower than 3.3 V, as these are already available. Each higher voltage would require additional power supply setups, like voltage boosters, to gain a higher voltage.

As the display unit needs to be integrated into the frame of SPECTACLES, and it is desired that the optical path is integrated too – in comparison to optics which are replacing a part of the user’s field of view – it is necessary to use small display components. Hence reflective display components are not a good choice, as they require large setups, either with three types of LEDs or a “colorwheel”, which would not fit into the frame of SPECTACLES. Power consumption will be higher too, as additional electronic is needed to control the three LEDs. It is clear that the used microdisplay should be either of LCD or oLED technology, and current issues of oLED displays plead for the LCD display technology. But as soon as the described aging issues are solved, oLED displays will be the best choice for SPECTACLES as they do not require an additional backlight like LCD displays do. For the first hardware prototypes, the Kopin Cyberdisplays are a good choice, as interface electronic is not a criteria for prototypes, but for later versions, oLED displays should be preferred, especially when it comes to optical elements and the physical dimensions of the display unit.

4.4 Optical Elements

Optical elements are needed to combine the user’s natural scene with the display output. The use of microdisplays is necessary to satisfy the requirements for small dimensions of the display unit. Also, microdisplays consume less power than small LCD screens, which could be placed in front of the user’s eye instead of the glass pieces of SPECTACLES. The size of microdisplays, especially their display diagonal of few millimeters makes it nearly impossible for the human eye to use such a display without magnifying lenses. Since the user should be able to use SPECTACLES and perceive his environment at the same time, a method is needed to merge the user’s natural scene with the output of the system. There are two ways of image combining.

- Video See-Through

The world in front of a user is captured by a camera. The system output is merged with this image and displayed by a small screen or magnified microdisplays in front of the user’s eye. The screens are non-transparent, i. e. HMDs using this method cannot be worn by a user when the system is in a passive state – because all screens are deactivated – without restricting the user’s field of view.

- Optical See-Through

The output is projected through various prisms and mirrors onto a projection surface in front of the user’s eye. This surface is transparent, which means that the user can see the world around her even if the system is deactivated. The method can be realized in two different ways.

- The display and lens system replaces a small part of the users view.

- The display is merged with a combiner prism into the view of a user, without occluding parts of the user’s field of view.

For optical see-through systems, either the optics can replace a part of the user’s field of view, or a prism is used, to merge the user’s natural scene with the output of the display. The first variant is the simpler and in most cases the compact one. The second variant needs complex optics which are consuming a large amount of space. Various prisms and lenses are needed to make it possible for the human eye to focus onto the display.

- **Refractive Prism Combiner** (figure 4.6)
The Display is focused by single or multiple lenses to achieve a more compact design. Without the lens system, the human eye would be unable to focus onto the display output, because the combiner prism is very close to the eye.

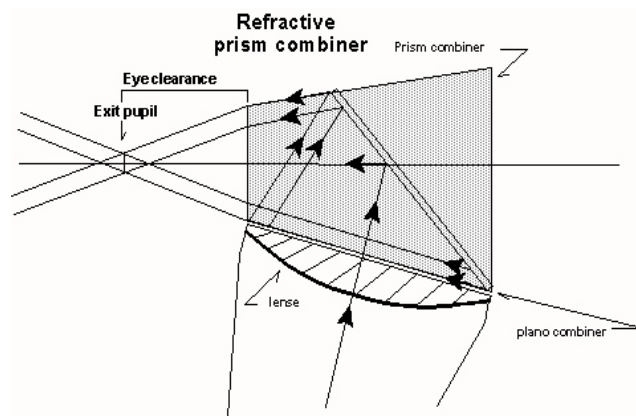


Figure 4.6: Refractive prism combiner [14]

- **Folded Catadioptric** (figure 4.7 on the following page)
The folded catadioptric is similar to the prism combiner, but it exploits total reflection within special glass prisms. With this exploitation, the length of the route from the display to the eye is stretched, moving the display virtually away from the eye. Thus making it easier for the human eye to focus onto the display.

A third way for image combining makes use of holographic elements. The main advantage with such holographic elements is their very compact design, as the optical elements are only several micrometers thick and can be placed directly onto a glass piece of SPECTACLES. But the main downside of holographic elements makes them inefficient for SPECTACLES at this time, because they are very expensive and time-consuming to create.

For the current project stage, a Kopin CyberDisplay 230k was chosen. It has the lowest power consumption of the displays presented in table 4.2 on page 15 and requires a single power supply of 3.3 volts. The display is capable of 16 bit RGB color images with a resolution of 320x240 pixels, which is the highest available resolution for SPECTACLES at the moment, compared to power consumption and physical size. The Kopin CyberDisplay 230k requires interface electronic to convert the pixelclock of the Intel PXA27x LCD controller into a display specific format. This interface is realized by two J-K Flip-Flops and a set of NOR gates, which was assembled by myself at the Institute for Pervasive Computing.

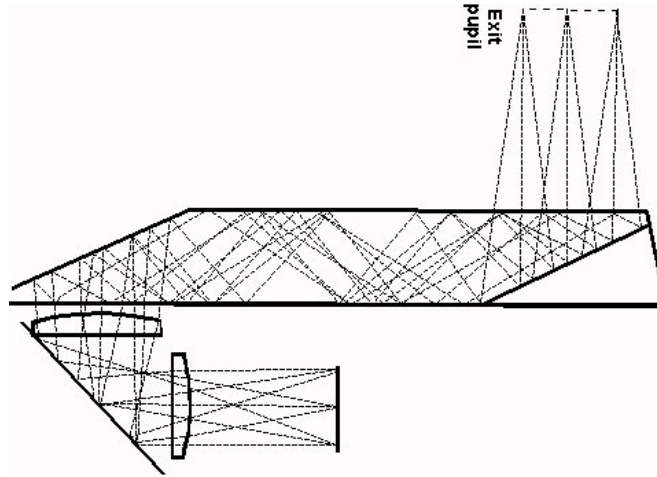


Figure 4.7: Folded catadioptric [14]

To ensure a long battery lifetime, all unnecessary power consumption should be avoided, therefore, the SPECTACLES display is not active all time. The display unit as a main power consumer may be deactivated if it is not in use by any application or component. With this fact in mind, it is clear that video see-through systems are not a good choice for SPECTACLES as power consumption will be higher in comparison to optical see-through systems, because additional power for the camera is needed. Video see-through may also produce a higher processor load for image processing. In addition, the user would have to take off the SPECTACLES device as soon as the display is deactivated, because she could not see the environment in front of her. Optical see-through systems which are replacing a part of the user's field of view are a bad choice too, because of the same facts. The user will have the display system in front of her eyes all the time, even if there is no video output. Such a restriction of the field of view is uncomfortable for the user and may lead to dangerous situations, if the user is not able to fully perceive her environment.

With the above said, optical see-through systems with prism combiners seem a good choice for the SPECTACLES. Their main drawback, large and heavy glass prisms, may be slightly improved by using plastic prisms instead, as they are more lightweight.

Chapter 5

Implementation

The operating system of the SPECTACLES device is an ARM-Linux based kernel, as used for PDAs and other handheld devices. Thus making it possible to use standard Linux commands and programs like a Linux console. The Gtk+ toolkit is used as the graphic environment. Gtkmm, the C++ extension of the Gtk+ C version, was suggested as a good choice, but during the implementation phase, the needs for these additional libraries were nullified, as all widgets and commands used so far are from the Gtk+ libraries.

To control the hardware components, activate and deactivate them as needed, a software component is implemented. The SPECTACLES component framework (as shown in figure 5.1) bases on the system libraries and system services. Applications running on the framework should avoid direct access to the underlying Linux system, but use the framework to fulfil their tasks. This requirement to applications makes clear that the SPECTACLES framework must offer as much functionality as possible.

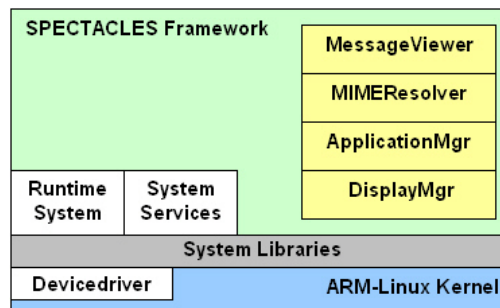


Figure 5.1: The SPECTACLES framework.

To ensure a long battery lifetime, power consumption has to be as low as possible at all time. Hardware, which is currently not in use, should be turned off or at least switched into a low power mode, if possible. The component which controls the actual battery level and the processor speed is called PowerManager. The PowerManager defines a power aware component and manages them. All power aware components offer methods to be power regulated by the PowerManager. Based on the processor load, the PowerManager can change the core frequency of the PXA270 processor and the memory bus accordingly.

If the battery level becomes low, the PowerManager will notify all power aware components to reduce their power needs. The component itself must decide what it will do when it receives this notification. In addition, before a component can be activated, it must request this action at the PowerManager. If there is insufficient battery power left, the PowerManager will permit the activation of this component. The component used to activate and deactivate an other component is called ComponentManager.

Inter-Process Communication

The communication between the components is realized with an extended version of the tuplespace which was implemented at the Institute for Pervasive Computing for this project. “Normal” tuples which are placed in the tuplespace are also called persistent tuples. They can be placed with an *out* command, and are removed with an *in* command. The *read* command can be used to get a copy of a tuple in the space. The last two commands are blocking, i. e. when a thread calls one of these commands, it must pass a scheme of the tuple it is interested in and the thread is blocked until a tuple in the space matches the scheme. The scheme is called *antituple*. Of course, non-blocking variants of the *in* and *read* commands are available too.

In addition to tuples which are placed in the space until an *in* command removes this tuple, a new type of tuple can be used, called “volatile” tuple which is just placed virtually into the space. That means, a volatile tuple is not placed into the space, but copied into a buffer of the thread which controls the tuplespace. All components interested on a specific tuple must register with this thread and specify the tuple they are interested in. When the thread receives a volatile tuple, it checks the list of registrars and calls the method `process()` of the registered component, if it is interested in this tuple. This is similar to the JAVATM event mechanism. After all components in this list have been checked, the data of this volatile tuple is deleted.

5.1 DisplayManager Component

The DisplayManager component is used to control the available space on the display, as well as the physical display itself in cooperation with the PowerManager. It is a power aware component, i. e. the component is executed within a new thread, started at it’s creation (i. e. when the constructor of the DisplayManager is called). The PowerManager can change the power level of power aware components when battery power becomes low, or the system is idle. Power aware components are also used to control physical hardware. The power aware component is derived from the less specialized active component, which cannot be controlled by the PowerManager. When the DisplayManager is activated, it registers with the PowerManager and the connection to the tuple space is established. The DisplayManager waits then for incoming requests.

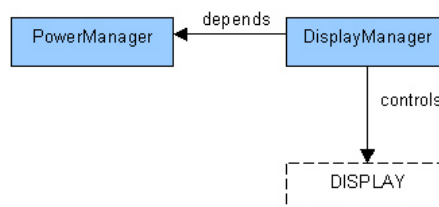


Figure 5.2: The dependencies of the DisplayManager.

Whenever a component wants to show content on the display, a region has to be requested from the DisplayManager. Because of the small size of the display, it is required that an

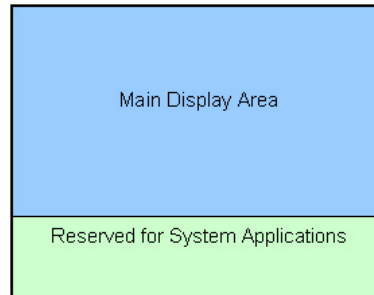


Figure 5.3: Scheme of the display sectioning.

application can control the whole display area at once, as windowing of multiple applications results in too small windows (figure 5.3 shows the display layout, the large part can be used by a component while the small part is reserved for the system). In addition, as the performance of the processor board is adapted to the available power – with sinking battery power, the processor speed is reduced by the power manager – multiple windows and applications produce too much processor load, resulting in slow reactions on user input. Therefore, a mechanism is needed to decide which window is displayed, and which windows have to be hidden. A simple, yet powerful solution is to use priorities for each window. Before a component can request the display area, it must register with the DisplayManager. In this registration, the component must specify its priority level, which is used by the DisplayManager to decide if the component can get access to the display, or has to wait until a higher priority window releases the display area. The priority is specified as an integer, which ensures fast comparison of the priority of the current displayed window, and the requesting window. The priorities are defined as follows:

- priority 0
The lowest priority level. This value should be used only by components which require the display rarely. The DisplayManager will deactivate the display unit if the PowerManager sets the DisplayManager to a lower power level.
- priority 1 to 3
Low priorities. If the display is accessed by a component with a low priority level, the DisplayManager may deactivate the display unit to lower the power consumption, if the PowerManager requests it, depending on the power level.
- priority 4 to 6
Normal priority levels. The display unit will be deactivated only if the power level is set to a very reduced level, i.e. from high power consumption to very-low power consumption.
- priority 7 to 9
High priorities, especially for components using videostreaming or other tasks with high power consumption. The DisplayManager will deactivate the display unit only if the PowerManager requests a deactivation of the DisplayManager component for a complete shutdown of the display unit. Requests for a lower power consumption will be ignored.

- priority 10
The highest priority, reserved for the SPECTACLES system-core components, like the PowerManager or the ComponentManager.

After the registration, a component can request access to the display area. If its priority is equal or higher as the priority of the component which is currently using the display, it gets access, and the current component must hide its window. Figure 5.4 shows the registration process of an application with a lower priority than the application which is using the display at the moment. The requesting application is enqueued into a list which is sorted descending by priority. When the active application is hiding or closing its window, the DisplayManager takes the application with the highest priority out of this queue. The application is notified that it can use the display now.

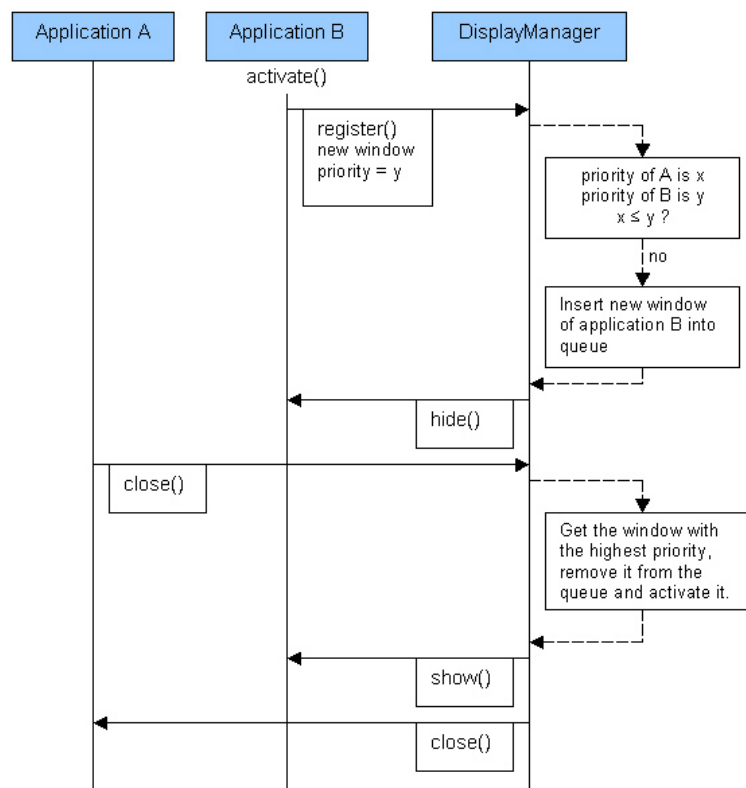


Figure 5.4: Sequence diagram of an application registering a window.

In a separate thread, the DisplayManager controls the Gtk+ main event loop. In this helper thread, Gtk+ events, like the instantiation of new widgets or the modification of widgets, are handled and executed. When the DisplayManager is deactivated by the PowerManager, this thread is suspended too, stopping all Gtk+ event handling. In addition, the DisplayManager will request its deactivation if there are neither applications left in the queue nor is an application currently using the display. Without a request, the DisplayManager cannot determine if there are windows left in need of event handling. In this case, an unregistered window can send Gtk+ events to the underlying Gtk+ and XServer layers, but they would not be executed until the Gtk+ main event loop, respectively the DisplayManager helper thread, is started again. Because of these facts, every component has to request its window actions at

the DisplayManager, to ensure all Gtk+ events are handled properly. For later developments of components using Gtk+ widgets, the MainWindow class was implemented. This class is an active component which registers with the DisplayManager as soon as its constructor is called.

The MainWindow Class

The MainWindow contains a pointer of the type GtkWidget, which should be used to create the top level window of an application. When an application uses a window which is inherited from the MainWindow class, it can use the functions `show()`, `hide()` and `close()`. these functions are emitting the necessary tuples to the DisplayManager, requesting the specified action. Consequently, an application using such a window can call the functions and don't have to concern about the emission of any tuple to the DisplayManager as this is already done by the MainWindow.

An application can request to hide its window, or it can receive a command by the display manager to do so. To reshown a hidden window, or to show a new window, a tuple has to be emitted. The DisplayManager will check the priority of the request, which is defined upon creation of the window in the corresponding constructor. If it is higher or equal to the priority of the actual displayed window, the requesting application is allowed to show its content on the screen. The actual displayed window is enqueued into a list of suspended windows. A tuple is sent to the owner of this window notifying it to hide the window, whereas the requesting application gets a tuple that it is allowed to show its window. As soon as the request is acknowledged, the application can take full control of the screen. That means, for every other action within the display area, no further requests are necessary.

When the DisplayManager receives a request of an application with a higher priority than the application currently using the display, the active window will be enqueued and its status flag within the queue is set to `show_win`. This flag determines if a window was enqueued by a higher priority window or because the application which owns the window requested it. With the flag set to `show_win`, it will be reactivated as soon as the higher priority window is closed or hidden. If an application don't want its hidden window reactivated, it has to send a tuple, telling the display manager to set the status flag of the window to `hide_win`. Unless the application sends a request, the window will not be shown again. Hide requests are always acknowledged positive.

Before the shutdown of an application, a request to close the window must be sent. Such requests are always acknowledged positive and used to ensure that no window remains in the queue of suspended windows respectively the active window is changed.

All these tuples, used as an command interface to the DisplayManager, are generated automatically when a window, inherited from the MainWindow class, is used. An application just needs to call `show()`, `hide()` or `close()`.

5.2 ApplicationManager Component

It is necessary that SPECTACLES provide a system to let the user start various applications himself. The ApplicationManager is an implementation of a user interface with an accelerometer as the input device. It uses a small area at the bottom of the screen, to display a list of icons. The ApplicationManager component depends on other components to be able to fulfil its task. Figure 5.5 shows these dependencies.

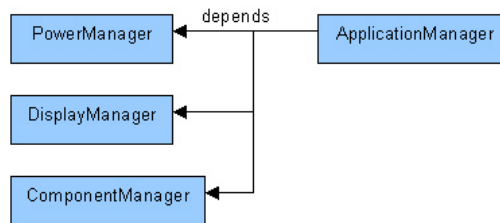


Figure 5.5: The dependencies of the ApplicationManager.

The ApplicationManager component controls the input of the accelerometer as long as no application is started. When an application is started by the ApplicationManager, the focus of input signals is given to the new started one, as it can now control the entire screen. The focus of input signals is returned to the ApplicationManager, if the application is closed by the user or deactivated by the ComponentManager. Figure 5.6 shows the position of the ApplicationManager window where the icons are displayed. The window uses a part of the system application area which was defined in Section 5.1 on page 21. The remaining display area on the left and on the right of the ApplicationManager window can be used for system notifications, e. g. an icon of a battery is displaying the remaining battery power.

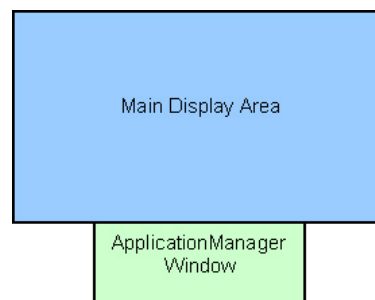


Figure 5.6: Scheme of the ApplicationManager window.

An application is represented by an icon within the ApplicationManager. One of these icons is displayed larger than the other icons, and it is referred as the **selected** icon. By moving the head to the left or to the right, the user can scroll bidirectional through the list of icons. This list is displayed cyclical, i. e. when the user scrolls “over the end” of the list, the beginning of the list is displayed again, and vice versa. Figure 5.7 on the next page shows screenshots¹ of the running ApplicationManager to visualize the icon² layout.

¹The background color has been changed from black to light gray for printing purposes.

²The icons are taken from the KDE 3.3 icon set.



Figure 5.7: Screenshots of the running AppManager with one, two and three icons.

Applications are distinguished into the following types, defining if they are required to register an icon at the ApplicationManager or not.

- **System Application**
An application which is needed by the SPECTACLES to operate as specified in the requirements, for example the ComponentManager, the PowerManager or the Display-Manager. Neither of these applications need an icon in the icon list, since they cannot be started by the user, but are already running or started by the system-core.
- **GUI Application**
An application which is using the display but is not needed by the SPECTACLES core system. These applications can be started by the user for additional functionality or extensions of the basic system. An example for such an application is the Message Viewer (see Section 5.4 on page 31 for details). When a new message is received, the Message Viewer is activated by the system, but does not display the message content on the screen. It registers with the AppManager and creates a new icon for the message. When the user selects and activates this icon, the message will be displayed on the screen and the icon is removed.
- **Background Application**
This kind of application is started by an other application. It is not necessary or impossible for the user to start this application, because, for example, command line parameters are needed.

To fulfil it's task, the ApplicationManager needs user input. As the user input possibilities of the SPECTACLES are very limited, the ApplicationManager should use as few different input signals as possible. We use head gestures tracked by an accelerometer as input. The conversion from head gestures to input signals, like pressing a key on a keyboard, is part of another subproject of the SPECTACLES research project, as well as the definition which gesture emits which signal. The ApplicationManager needs three of those signals.

- “left”
When the AppManager receives this input, the list of icons is scrolled to the right. The icon on the left side of the currently selected icon becomes the new selected icon (figure 5.8).

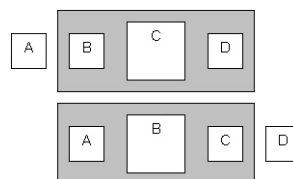


Figure 5.8: Scheme of the icon exchange when the “left” signal is received.

- “right”

This signal induces the AppManager to scroll the list of icons to the left. The icon on the right side of the currently selected icon becomes the new selected icon (figure 5.9).

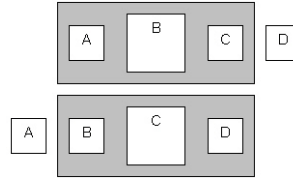


Figure 5.9: Scheme of the icon exchange when the “right” signal is received.

- “launch”

This signal tells the AppManager to start the application corresponding to the selected icon. Depending on the registration of the icon, different actions can be taken.

A design goal for SPECTACLES components is to achieve as much functionality as possible, with few code and libraries, as the memory space on the device is limited. Although the first prototype of the ApplicationManager was able to fulfil it’s task of starting and managing applications, evaluations of the prototype have shown that the ApplicationManager was unable to identify the actions required to start an application. This means, the ApplicationManager requested a startup of every application at the ComponentManager, regardless of the application type. Since applications can be components – started by the ComponentManager –, launched in the console, or just requesting an input from the user, this behaviour had to be changed for later prototypes.

All GUI-Applications must register an icon if the user should be able to launch them. Figure 5.10 on the next page shows a sequence diagram of the registration and notification process. In the first prototype, the application had to pass it’s component name to the ApplicationManager, and this name was used to request a startup at the ComponentManager. If the application needs to be executed in the Linux console, and passes a command as name, e.g. `/usr/share/application --start`, the ApplicationManger would not recognize this and will request the startup of a component at the ComponentManager, instead of starting it in the console.

The second prototype took this into account, and the icon registration was extended. An application must specify an action that the ApplicationManager should take when the user selects and starts the application. Three different types exist.

- notify

The registered application wishes to be notified when the users “clicks” it’s icon to take further actions (e.g. a new message is received, so a new icon is added to the ApplicationManager to let the user decide when the message is displayed).

- console

The application is started in the console. The registrar’s name is a Linux console commando, e.g. `name = "/usr/local/application"`.

- component

The registered application is a component within the SPECTACLES framework. The name of the registrar – i. e. the name of the component – is used to request a startup at the ComponentManager.

As a single application might have more than one icon registered, additional extensions were needed. Especially when the icons are registered as *notify* types, since the ApplicationManager sends a response to the corresponding application, but this application would not be able to determine which one of it's icons was selected and started. Coming along that an application can remove icons, it is not clear for the ApplicationManager which icon needs to be removed.

The current prototype is aware of this and again, the registration process is extended for another field. Each application can now pass a note to the application manager, which is returned to the application in the response message. When a unique identifier is used as note, the application and the AppManager can identify an icon, which makes the correct removal of an icon, and the response to the user selection possible.

During the implementation of the MIME-Type resolver (see Section 5.3 on the following page for details) and the evaluation of the ApplicationManager prototypes, the above described limitations were encountered.

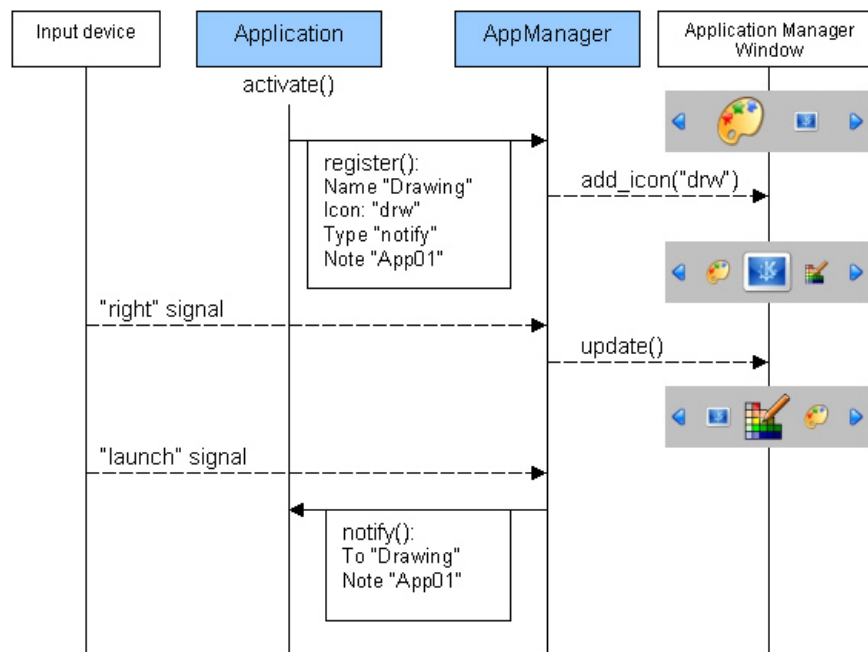


Figure 5.10: Sequence diagram of an icon registration and notification with the application manager.

5.3 MIME-Type Resolver Component

As stated in the requirements, SPECTACLES should be able to display different types of messages on the screen, e.g. text, images, videos etc. It is planned that SPECTACLES displays SMS or MMS as the user requests it. To display those messages, a message viewer component is needed which is capable of all formats. In addition, the component (further referred as “Viewer”) should be able to display future message types too. It is clear that it is not possible to implement such a component, as future message formats are currently unknown. Instead of one Viewer for all message types, multiple Viewers capable of one or few message types should be used. This variant can still be able to display all current message types, but with the advantage of better extensibility. To avoid lots of Viewers running and listening if there are messages to display, a single component is used to determine the correct Viewer for a message and activates it. This component is called the MIME-Type Resolver, or MIME-Resolver.

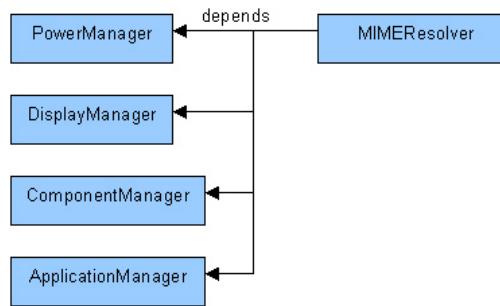


Figure 5.11: The dependencies of the MIMEResolver.

The MIME-Resolver uses two different tuples to manage messages. The content of a message is stored within a MessageData tuple. To cause a Viewer to several actions, e.g. open a message, MessageEvent tuples are used. As the MIME-Resolver is a layer between raw message data and the Viewer components, the MIME-type of a message has to be set by the component which creates the message.

Message Data

A message is stored as a persistent tuple in the tuplespace. Either the message content itself is integrated in the tuple, e.g. a string of characters for text messages, or it contains a description of the message content location, e.g. a filename. Since all components have access to the tuplespace, they are all able to emit such a message data tuple. Therefore it is possible for a component to display messages on the SPECTACLES screen without any knowledge of the underlying Gtk+ runtime libraries. The message data tuple contains additional information about the message origin and a field to mark a message as read. Below is an example of a MessageData tuple.

```
< "data", source, mimetype, messageID, origin, read, data >
```

The first field of the tuple is used to identify it as MessageData tuple without checking every single element and must be set to “data”. The second field contains the name of the component which placed the tuple in the tuplespace, followed by the MIME-type of the message, and a unique message ID. The field *origin* can be used to specify the original sender of the message, i. e. the telephone number or the name of the sender as a character string. The field *read* is used to mark messages as read, when the user has opened the message. In the field *data*, the message text is stored – in case of text messages – or a filename if the message content is stored on the SPECTACLES file system.

When a component emits a message data tuple, it has to generate a unique identifier. This identifier has to be unique for the component, but not for the entire message system, as the MIME-Resolver uses the identifier and the component name to clearly identify a message. This behaviour is suggested for all Viewers too, as this identification method relieves the development of components using the message system, because it is easier to ensure unique identifiers within a single component instead of an identifier which is unique for the whole SPECTACLES system.

Corresponding to the message content, a MIME-type must be set. According to this MIME-type, the MIME-Resolver selects a suitable Viewer and requests its startup at the ComponentManager. The resolving mechanism uses persistent tuples which are defining a Viewer for a specific MIME-type. The MIME-Resolver searches for these persistent tuples and adds all found MIME-types with the corresponding Viewer component name into a list, when a message with an unknown MIME-type is found. A MIME-type is unknown, when it is not contained in this list. If no Viewer can be found after a search in the tuplespace, the Viewer for the type “text/plain” is used.

Message Events

To control the Viewers independent of their implementation, an event driven environment is used. A message event is represented by a message event tuple, which contains information of the event type, the message identification and the name of the component which emitted the original message data tuple. The following events are defined.

- “new”
A new message is received. This event causes the MIME-Resolver to check the message MIME-type and request the activation of the suitable Viewer.
- “open”
Open the message and display it. This event is mainly emitted by the MIME-Resolver, with the correct message identification set.
- “update”
The message content has been changed, update the window content too.
- “mark”
Mark the message as read. The MIME-Resolver updates the message data tuple when this event is detected.

- “close”
Close the window which is displaying the message. It is not necessary to destroy the window, as the same message may be displayed again later.
- “delete”
Delete the message. The MIME-Resolver removes the message data tuple from the tuplespace.

If a component puts a new message data tuple into the tuplespace, the MIME-Resolver will not react on it, until a *new* event is emitted, too. The MIME-Resolver component offers two static member functions to put such a data tuple into the space. With an additional option, the calling component can decide if the *new* event is emitted too, or will be emitted later by itself. Below is an example of a MessageEvent tuple.

```
< "event", destination, messageID, event >
```

Similar to the MessageData tuple, the first field is used to identify the tuple as MessageEvent tuple and must be set to “event”. The second field, *destination*, is used to specify the correct Viewer for a message. Each Viewer should check if it’s component name equals the *destination* field, to avoid multiple Viewers from opening a message. The message ID is equal to the ID in the MessageData tuple, and used by a Viewer to find and display the message content. The field *event* is set to one of the message events defined above. The MIME-Type Resolver offers additional static functions for simple creation of MessageEvent tuples.

5.4 MessageViewer Component

The MessageViewer is a simple component to display text messages, like an SMS, on the SPECTACLES display. The component uses a class derived from the MainWindow class (a description of the MainWindow can be found in Section 5.1 on page 21) to display the text in light gray with black background.

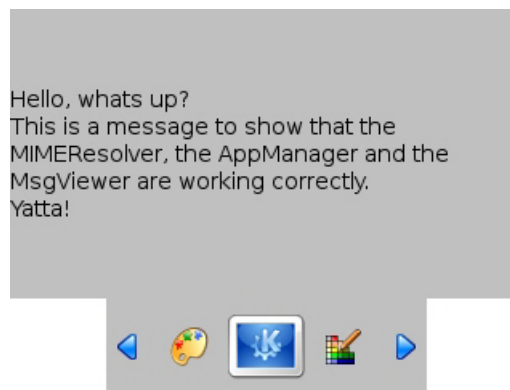


Figure 5.12: Screenshot of the MessageViewer component displaying a message.

In the initial prototype, only one message could be displayed per `MessageViewer` component, as the used `GtkWindow` was destroyed after the component detected a special tuple. Soon, this was encountered as a strong limitation of functionality. In addition, after the message window is closed, the component needs to be deactivated and reactivated by the `ComponentManager`, to be able to display another message, resulting in higher CPU load and memory usage.

The current prototype avoids this behavior. As described in Section 5.3 on page 29, the messaging system is now based on events, which are controlling the `MessageViewer`'s behavior. The used window is no longer destroyed when a *close* event is received, but it is hidden, and the message text is removed. If a new message should be displayed, the `MessageViewer` puts the new text into the window, and displays it, if the `DisplayManager` grants access to the display unit. The `MessageViewer` has a priority of 5. Figure 5.12 on the previous page shows a screenshot of the running `MessageViewer`. The message text is extracted from a `MessageData` tuple, found with the given `messageID` in the `MessageEvent` tuple. If the message could not be found, the `MessageViewer` displays the text "No Message found".

Although the `MessageViewer` was mainly designed to display text messages received via Bluetooth, it can be used by every other component in the SPECTACLES framework, too. A component needs to emit the text message as `MessageData` tuple and create an *open* event (static functions of the `MIME-Resolver` can be used for this purpose). The `MessageViewer` will open and display the text on the SPECTACLES display unit.

Chapter 6

Conclusion

The presented implementation and selection of hardware components was made with the SPECTACLES requirements in mind. To avoid bugs and improve performance, I have used standard Linux C libraries – which are excessively tested by the Linux community – whenever possible. Although there are various projects with a similar goal, the SPECTACLES user interface and display control is different from other mobile devices and their development. Most all projects are case studies of how a head-worn device or devices with a head-mounted display as output can help a user on her daily activities. This means, that these projects are not necessarily concerned about the size of the hardware components, but of their functionality. This said, it was difficult to re-use approaches of other projects, since they are limited toward a special field of use, whereas SPECTACLES is not. The functionality of the components were designed to be versatile and easy to extend. However, during the implementation of additional components, e.g. the MIMEResolver, already implemented components and tuple-interfaces had to be changed and extended, as they were not able to handle the needed functionality. Because of this fact, the implemented components may not be versatile enough and need additional work.

It should be noted, that the SPECTACLES project is still evolving at the time this bachelor project has been made. Additional functionality has been added and new components have been developed to achieve the project milestones. Also, the components presented in this work have been partially reworked by myself to guarantee further compatibility with the SPECTACLES core component system. The input device, an accelerometer, is currently the only input device available, and the display unit is the only available output device. For upcoming versions of the SPECTACLES device, an audio input/output channel will be available too. Thus opening a great advantage for the user, as she can be supplied with additional audio feedback. During the implementation of the ApplicationManager, the accelerometer input device was not available, as it was still under development. A usability study may be needed to test whether the current implementation of the ApplicationManager is easy to use, or not.

The presented hardware components, especially the microdisplays, are of course improved by their developers over time. Therefore, new and better displays might be available on the market now, which were not available during the review taken for this bachelor project. Since SPECTACLES should use the components fitting best to its requirements, a new exploration of microdisplays may be necessary.

Further Work

The display unit of SPECTACLES can contain a backlight LCD, a reflective LCOS or an oLED display. For the first two types, a light source is needed to create bright images. As for the LCD display, which uses only a single white light source – in case of per-pixel color filter

technology –, a special extension of the optical path could help to save battery power. If the user is outdoors, or in a bright indoor environment, ambient light could be used as background light source for the display. As shown in scheme 6.1, a half-silvered mirror merges the path of the LED and the ambient light. Depending on the brightness of the ambient light, the internal LED light source could be reduced or even turned off, thus saving battery power. The question if the ambient light is bright enough to act as a light source should be investigated in further research. It should be kept in mind, that amplifying lenses may increase the efficiency of this ambient light application, but can also cause major damage to the users eye, if the sunlight is focused and directly projected towards the users eye.

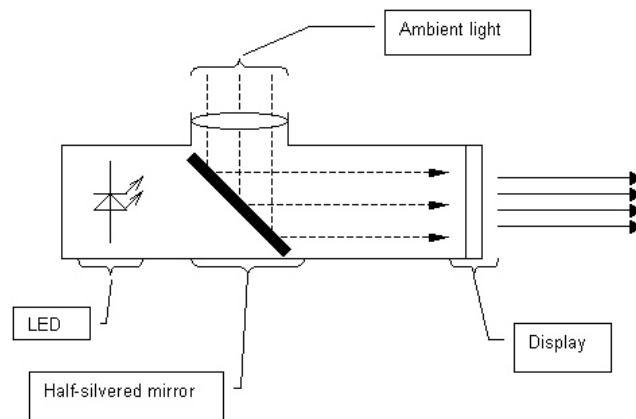


Figure 6.1: Scheme of the optical path including ambient light.

Although with currently available hardware, it might seem impossible to construct an autonomous system like the SPECTACLES which is integrated into glasses, current trends of processor and memory development as well as the sizes of microdisplays will make it possible.

Bibliography

- [1] Stephen Brewster and Peter G. Cryer. Maximising Screen-Space on Mobile Computing Devices. In *Proceedings of CHI '99 extended abstracts on Human Factors in Computing Systems*, pages 224–225, 1999.
- [2] Christopher S. Campbell and Peter Tarasewich. Designing Visual Notification Queues for Mobile Devices. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1199–1202, 2004.
- [3] Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. The Design Space of Input Devices. In *Proceedings of the CHI '90 Conference on Human Factors in Computing Systems*, pages 117–124, 1990.
- [4] Intel Corporation. *Intel® PXA270 Processor Product Brief*, 2004. <http://www.intel.com/design/embeddedpca/applicationsprocessors/302302.htm>.
- [5] Intel Corporation. *Intel® PXA27x Processor Family Video Decoder Display Performance Optimization*, 2004. <http://www.intel.com/design/embeddedpca/applicationsprocessors/302302.htm>.
- [6] Erik Geelhoed, Marie Falahee, and Kezzy Latham. Safety and Comfort of Eyeglass Displays. In *Proceedings of the second International Symposium on Handheld and Ubiquitous Computing*, pages 236–247, 2000.
- [7] Yakup Genc, Mihran Tuceryan, Ali Khamene, and Nassir Navab. Optical See-Through Calibration with Vision-Based Trackers: Propagation of Projection Matrices. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality*, pages 147–156, 2001.
- [8] Hong Hua. Development of Head-Mounted Projective Displays and its Application for Collaborative Environments. In *Proceedings of the first NFS PI Workshop on Robotics and Computer Vision*, 2003.
- [9] Jan Krikke. T-Engine: Japan's Ubiquitous Computing Architecture Is Ready for Prime Time. *IEEE Pervasive Computing*, April 2005.
- [10] Joanna Lumsden and Stephen Brewster. A Paradigm Shift: Alternative Interaction Techniques for Use With Mobile & Wearable Devices. In *Proceedings of the 13th Annual IBM Centers for Advanced Studies Conference CASCON'2003*, 2003.
- [11] Juha Marila and Sami Ronkainen. Time-out in user interface: the case of mobile text input. *Personal and Ubiquitous Computing*, pages 110–116, May 2004.
- [12] Tomoyasu Nakatsuru, Yasuyoshi Yokokohji, Daisuke Eto, and Tsuneo Yoshikawa. Image Overlay on Optical See-Through Displays for Vehicle Navigation. In *Proceedings of the second IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 286–287, 2003.

-
- [13] Binh Pham and On Wong. Handheld Devices for Applications Using Dynamic Multimedia Data. In *Proceedings of the second International Conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 123–130, 2004.
- [14] Clarence E. Rash. *Helmet-Mounted Displays: Design Issues for Rotary-Wing Aircraft*, 2001. http://www.usaar1.army.mil/hmdbook/cp_index.htm.
- [15] Christian Sandor and Gudrun Klinker. A Rapid Prototyping Software Infrastructure for User Interfaces in Ubiquitous Augmented Reality. *Journal for Personal and Ubiquitous Computing*, pages 169–185, 2005.
- [16] Albrecht Schmidt, Hans-W. Gellersen, Michael Beigl, and Ortwin Thate. Developing User Interfaces for Wearable Computers: Don't Stop to Point and Click. In *International Workshop on Interactive Applications of Mobile Computing IMC2000*, pages 142–146, 2000.
- [17] Arthur Tang, Ji Zhou, and Charles Owen. Evaluation of Calibration Procedures for Optical See-Through Head-Mounted Displays. In *Proceedings of the second IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 161–168, 2003.
- [18] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, Michael Morris, and Wayne Piekarski. ARQake: An Outdoor/Indoor Augmented Reality First Person Application. In *Proceedings of the fourth International Symposium on Wearable Computers*, pages 139–146, 2000.
- [19] Joel Wilson, Dan Steingart, Russell Romero, Jessica Reynolds, Eric Mellers, Andrew Redfern, Lloyd Lim, William Watts, Colin Patton, Jessica Baker, and Paul Wright. Design of Monocular Head-Mounted Displays for Increased Indoor Firefighting Safety and Efficiency. In *Spaceborne Sensors II. Proceedings of the SPIE—The International Society for Optical Engineering*, volume 5800, pages 103–114, 2005.