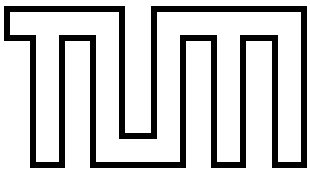
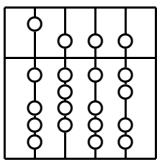


Technische Universität
München
Fakultät für Informatik



Systementwicklungsprojekt

Driver Development for TouchGlove Input Device for DWARF based Applications

Johannes Wöhler

Aufgabensteller: Univ-Prof. Bernd Brügge, Ph.D.

Betreuer: Dipl.-Inform. Christian Sandor

Abgabedatum: 15. Juli 2003

Erklärung

Ich versichere, dass ich diese Ausarbeitung des Systementwicklungsprojektes selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Juli 2003 Johannes Wöhler

Abstract

For augmented reality & ubiquitous computing, new innovative input devices have to be developed because the users working place has completely changed. The user is not sitting in front of a keyboard anymore. He uses the computers while standing or walking around. This thesis covers different aspects of developing new input devices for augmented reality & ubiquitous computing: The driver development, the integration of input devices into augmented reality systems and an example input device.

Preface

Purpose of this Document This thesis was written as a SEP at the Technische Universität München's Chair for Applied Software Engineering. From October 2002 through July 2003, five other Computer Science students and I designed, developed and tested an Augmented Reality application in the field of architecture. Each thesis covered a part of this application. This thesis is about the development of input devices.

Structure of this Document This thesis contains work about different aspects of input devices. The first chapter describes DWARF essentials and the ARCHIE sample application. The second part discusses a general hardware driver framework of input devices. The next part talks about the problems involved in classifying input devices and giving the user the possibility to interact with the system the way he wants. The fourth chapter is about an example input device called TouchGlove.

Contents

List of Figures

0 Introduction

Augmented reality (AR) is a new way for humans to interact with the computer. This new way of interaction requires new and specialized software and hardware. Augmented reality frameworks like the DWARF framework have to cover all functionalities, which are required for providing an useful AR application to the user. An important part of an augmented reality system is the human computer interface (HCI). This work will concentrate on the input device part of HCIs for augmented reality systems.
For AR, the requirements for input devices are completely different in comparison with ”normal” input devices like a computer mouse. Often, an input device has to be wearable, easy to use and should not disturb the user while doing other things. The mobile aspect is also central for AR frameworks. The user should be able to use his wearable input device in different places with local applications without re-configuring them.
In my thesis, I cover different aspects of developing new input devices for AR frameworks in general and for the DWARF framework in special.

Goals There were different goals I wanted to achieve with my work: Firstly, I wanted to find a way for integrating hardware drivers easily and in a standardized way into an AR framework. One main advantage of such a driver framework would be the possibility for driver developers in the future to reuse already developed driver parts. This part of my work is described in the chapter called *General Driver Framework*. This chapter is mainly for readers, who want to develop hardware drivers for AR frameworks.
The second goal was to find a way to connect input devices flexibly with input data consumers. This satisfies the requirement of mobility in AR systems. That part of my work is scientific research and described in chapter *Dynamic Matching of Input Devices in Mobile Environments*. A great part of the chapter discusses this problem on an abstract level. Therefore, readers, who are not familiar with DWARF, can read it also.
The third goal was to integrate an example input device into the DWARF framework, which can be used in mobile environments. The chapter *TouchGlove DWARF Services* describes the implementation of the DWARF services for my example device called TouchGlove. This part of my work is mainly interesting for developers, which want to integrate the TouchGlove input device in their application or want to implement other hardware drivers for input devices.

1 DWARF and ARCHIE

This chapter provides a general overview about Augmented Reality projects and frameworks, in particular DWARF the Augmented Reality approach of Technische Universität München. After over viewing the guidelines that lead to the DWARF framework its current state of development is outlined.
At the end of this chapter the ARCHIE project is introduced as a group project of several SEPs[1] and diploma theses. The completion of the ARCHIE project provides new functionality to DWARF thereby making it more mature.

1.1 Augmented Reality: A Stake-holders Point of View

The original intent in the development of computers was to provide support to people whose work was too difficult or too lengthy to solve manually, like large mathematical equations. New technologies arose as computers gained speed and more peripherals were connected to them. But the basic intention kept the same. Computers are supportive tools.
The increasing spread of computer networks in the last decade of the 20th century allows the distribution of services allocated to specific tasks. For example, rendering of 3D scenes is a resource intensive procedure which can be separated to another hardware, while a second machine can handle necessary remaining tasks of an application. The distribution of dedicated services to various platforms can get used in the Augmented Reality domain, because applications using this discipline have to aggregate various areas of computer science, where each may require a lot of computation.

Using Augmented Reality to support people can happen in diverse ways. But for the discipline of Augmented Reality two classes of computational assistances can be identified. On the one hand, there are independent tasks that can be supported by Augmented Reality, while on the other hand, the diversion of computers through the environment provides resources for Ubiquitous Computing. Both classes are described and in advance a combination is described.

1.1.1 Independent Tasks

Closely focused on a task, users may perform task-centered activities like maintenance or navigation [6]. To realize applications of this kind, developers can rely on paper based

[1] System Entwicklungs Projekt - a project every computer science student at TUM has to absolve

guidelines like maintenance guides or city maps. These guides can get formalized in state machines executed by taskflow engines [27]. And the Augmented Reality application leads the user through the task step by step.
Because of the runtime environment being known in advance, applications are comparatively easy to realize. Due to their nature these applications provide no flexibility to the users. Only the specified task can be realized usually only in the location specific to the application.

1.1.2 Ubiquitous Computing

Another aspect influencing Augmented Reality is Ubiquitous Computing [42]. Many possibly dedicated computers are placed in the users' environment, offering various services. These services can be of any kind, let it be telephoning, printing, viewing movies, or even ordering pizza. The support by the computer hereby should be invisible and intuitive to the user.
No predetermined taskflow is specified for these systems. However they are only useful if users have a clear idea on how to perform atomic actions or even whole processes so they might sometimes require assistance.
The provided services are sometimes fixed in one geographic location and don't support automatic reconfiguration corresponding to the environment.

1.1.3 Intelligent Environments

The combination of task-centered Augmented Reality and Ubiquitous Computing can result in Augmented Reality-ready intelligent environments. The aggregation of both aspects supplies services provided by the environment. Seamless interaction between these services and a mobile Augmented Reality system give each user a way to dynamically perform tasks as described in section 1.1.1.
For example, as the user enters or leaves a room, his Augmented Reality system recognizes context changes and informs the user about new services. Options could be offered via aural or visual channels. A head mounted display (HMD) can display information of tasks available from the current location. Also the HMD can be utilized to visualize the chosen applications' user interface by rendering e.g. virtual 3D scenes. If a corresponding tracking service is available the user can leverage this to get an accurately aligned view matching the current perspective.
Systems using such intelligent Augmented Reality-enabled environments are powerful, as they can accommodate not only predetermined taskflows but also spontaneous desires by the user.

1.2 Related Work

At the current time there are several research projects on Augmented Reality all over the world. The resulting software architecture of the systems differ wildly ([36], [37]), but two

general directions can still be seen.

In the first one prototypes are built by research groups which often result in task-centered systems for e.g. basic tracking concepts or car development concepts [39]. Usually they are highly specialized and monolithic. Many of these systems provide small demonstration setups for particular tasks. Even though the realized tasks essentially have a similar focus in other systems, the re-usability of their technology is quite difficult.

Other projects focus on middleware technology covering central Augmented Reality tasks and by this provide frameworks for applications. Although the concepts of software engineering [8] have been known for some time, they have not been widely applied in the Augmented Reality problem domain. But there are some projects tackling this issue.

The Computer Graphics and User Interface Lab of Columbia University has assembled different types of Augmented Reality applications [9] from a common basis. Their work focuses on providing a common distributed graphics library as a framework for reusable software components.

Mixed Reality (MR) Systems Laboratory of Canon Inc. developed a basic frame for mixed reality applications [39]. Their set includes HMDs and a software development toolkit for building MR/AR applications. The provided library supports common functions required in Augmented Reality applications, but still it cannot be spoken of a framework.

German Ministry of Education and Research founded the project ARVIKA[2] which is primarily designed as a Augmented Reality system for mobile use in industrial applications. The architecture is user centered, but relies on fixed workflows. It does provide a configurable access to offered features.
The industry, in particular a sub department of the Volkswagen AG needing software engineering technologies, already used some basic ARVIKA systems for car crash simulations [39].

An example for a multidisciplinary research program is *UbiCom* (Ubiquitous Communications) from the Delft University of Technology. Their architecture combines mobile units with stationary computing servers and focuses on mobile multimedia communications and specialized mobile systems [18].

Another approach is lead by the *Studierstube* project at Vienna University of Technology. That group uses concepts of software architecture among other things, but only as far as to keep parts reusable for testing new user interface paradigms [30] or for reconfiguring the tracking subsystems [25].
This can however only partially be seen as a framework for multi-user and multiple applications in Augmented Reality.

[2] www.arvika.de

At last we will take a final view on projects about Ubiquitous Computing. Today several approaches and technology systems share the idea of providing services to users through a star-shaped architecture, such as Ninja [11] or GaiaOS [12], [28].
Extendible Augmented Reality frameworks should rely on a decentralized architecture instead of the architecture of the approaches of these projects. Although some of them providing service federation, they don't seem to offer context and configuration propagation.

1.3 DWARF

The Technische Universität München also has a research project on Augmented Reality, which is called DWARF. The name is an acronym representing the guidelines for the general system architecture. DWARF stands for **D**istributed **W**earable **A**ugmented **R**eality **F**ramework.

The DWARF infrastructure provides an extensible, flexible and modular software framework for reusable Augmented Reality relevant components.
The framework can be seen in four abstraction levels. Figure 1.1 shows that layers.
The bottom layer is the layer of dynamic peer to peer systems. It provides connectivity and communication mechanisms for processes.
On top of this layer, the solution domain resides, supplying general components for the domains of Augmented Reality, wearable and ubiquitous computing. Services for tracking and environmental context are located here.
The third layer is descibed by the application domain. Components reusing general tasks of the sublayer reside here.
The top layer is built by the concrete applications available for the users.

A suitable framework for the Augmented Reality domain has three aspects: the announced *services*, a connecting middleware and a common architecture providing a basic concept to enable applications [6]. This framework allows components to be reused between applications and to dynamically configure them. One could e.g. imagine that the same tracking system provides position and orientation of certain objects to different applications.

Services *Services* are dedicated components covering general Augmented Reality tasks. Each *service* provides certain abilities to the user or to other *services*. On the other hand they can rely on input from other *services*, suppling filtered, analyzed, and rebuild information to other components or users.

Middleware A distributed application dynamically matches, connects and configures *services*, so that these can communicate directly corresponding to their needs respectively their abilities. The amount of *service* and their combinations can be changed dynamically by the middleware at runtime if required.

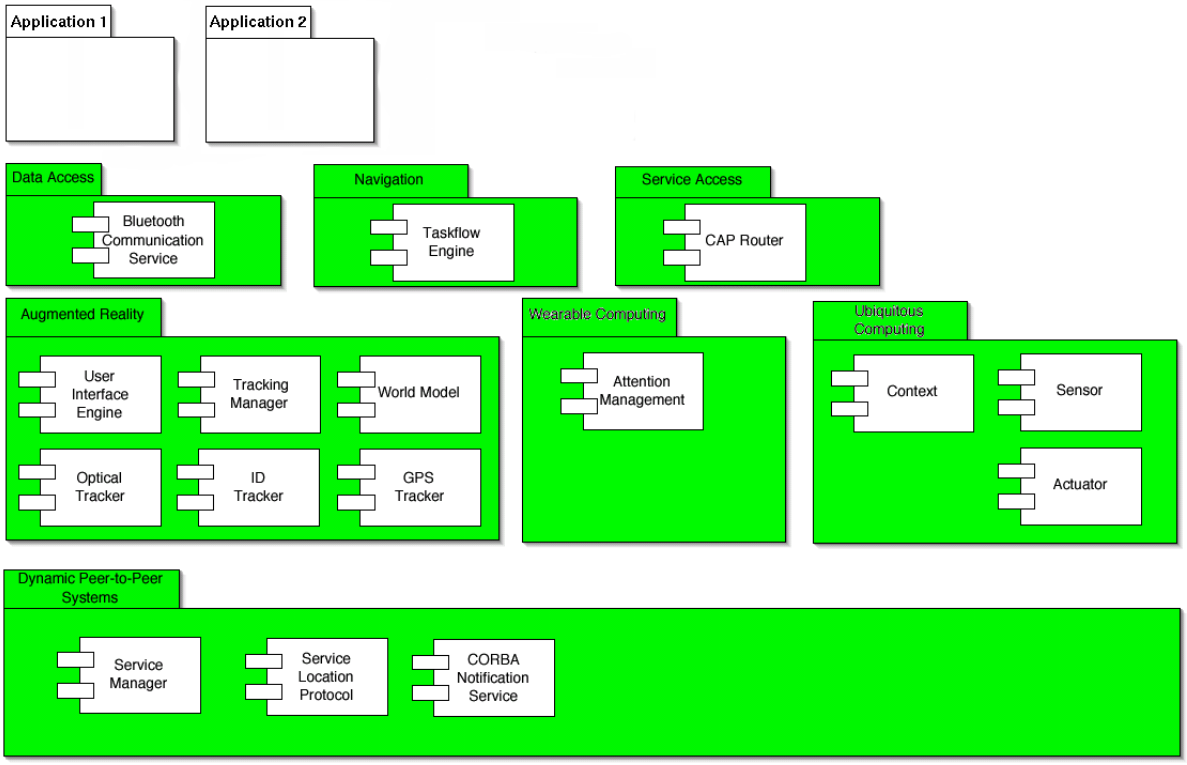


Figure 1.1: A layered architecture for the DWARF framework

Architecture A conceptual architecture describes the basic structure of Augmented Reality systems that can be built with it. To properly integrate different *services* with each other the respective developers have to agree on the roles of their *services* and on interfaces between them.

For our realization at TUM the main part of the framework consists of the *service-manager* and the service communication infrastructure. DWARF is designed as a distributed, and thereby wearable framework. Personalized software components can reside on different hardware components, even on mobile devices [6], enabling Ubiquitous Computing [42]. The *service-manager* handles all *services* and dynamically starts and stops required components on the corresponding hardware platforms.

The following section describes this context particularly.

1.3.1 Services

At DWARF applications each *service* can potentially run on its own hardware device as an independent process. It is described in a *service description*. Additional information about service parameters is stored here using attributes. There could, for example be an attribute

`accuracy` for a tracking device, or a `location` attribute for a printer.

In DWARF those *service descriptions* are specified in conjunction with *needs* and *abilities*. These describe on a high level how *services* can connect. Looking at two connected *services* one has an *ability* and the corresponding partner has the matching *need*.

Each of these two has a *connector* specifying the communications protocol between them. Current protocols provide communications via event notifications, remote method calls (CORBA[3]) and shared memory. Two communicating *services* must have the matching communication protocols.

Needs and *abilities* also have a `type` which distinctly defines the corresponding interface. Thus for two matching *services*, one has a *need* with the same *connector* and the same `type` as the other servives' *ability* has.
Hence `types` in *needs* and *abilities* can be used in various ways, e.g. a selection predicate is settable. The coding rules for these predicates follow the LDAP RFC 1558, 1960, 2054[2].

By the use of `minInstances` and `maxInstances` values for multiplicity are attributable to *needs*. If for example `minInstances` is set to "1" for a *need* of a *service*, this *service* will only start properly when at least one corresponding ability of any other *service* is connected to it.

Figure 1.2 illustrates the description of two different *services* with a possible connection in easy readable XML-notation.

```
<service name="Tracker"
         startCommand="Tracker"
         startOnDemand="true" stopOnNoUse="true">
  <attribute name="location" value="GreatHall"/>
  <ability name="peoplesPositions" type="PoseData">
    <attribute name="accuracy" value="0.1"/>
    <connector protocol="PushSupplier"/>
  </ability>
</service>

<service name="Map">
  <need type="PoseData"
        predicate="(&amp;(location=GreatHall)(accuracy<1.0))"
        minInstances="1" maxInstances="10">
    <connector protocol="PushConsumer"/>
  </need>
</service>
```

Figure 1.2: Two simple connectable service descriptions

[3]Common Object Request Broker Architecture

1.3.2 Middleware

A *service manager* residing in each participating computer, is able to contribute its service descriptions to a common pool. The service-managers internally check all possible connections between *needs* and *abilities* of all *services* and dynamically connect and start matching ones on demand. [20]
Intra-*service* as well as internal *service-manager* communication take place via CORBA. Thus every *service* contains an interface to it.
The *service-managers* running on different computers find each other via SLP[4].

1.3.3 Architecture

A conceptual architecture defines the basic structure of Augmented Reality systems which can be constructed with it.
Thus it ensures that service developers agree on the roles of their own *services* within the system and on interfaces between them.

Figure 1.3 shows an example architecture for DWARF applications. It is separated into six packages. The distribution of services among the required subsystems of the general Augmented Reality architecture is shown, too.
The tracking subsystem is responsible for providing location information on real objects as positions or raw data streams. The world model subsystem holds all relevant data on real and virtual objects and provides virtual models for the presentation subsystem which generates user visible scenes. Interaction with the system is handled in the control subsystem that reacts on user input and provides input data to the current active applications. The application workflow resides in the application subsystem. Context information is handled by the context subsystem.

1.4 Extending the Space of Components

The DWARF framework is still under development. However multiple applications have been built by now with it.
Its initial version was verified by implementing an application for a guidance scenario called *Pathfinder* ([5, 20, 22, 27, 29, 40, 43]).
Following this, multiple projects (*STARS*,*Fata Morgana*, *FIXIT*, *TRAMP*, *PAARTI* and *SHEEP* [1]) increased the amount of services, provided by the framework.

1.4.1 Existing Services

Besides the classification to different levels in the last section, services can also be classified in four groups. One will see that some groups directly reference reusable framework com-

[4]Service Location Protocol

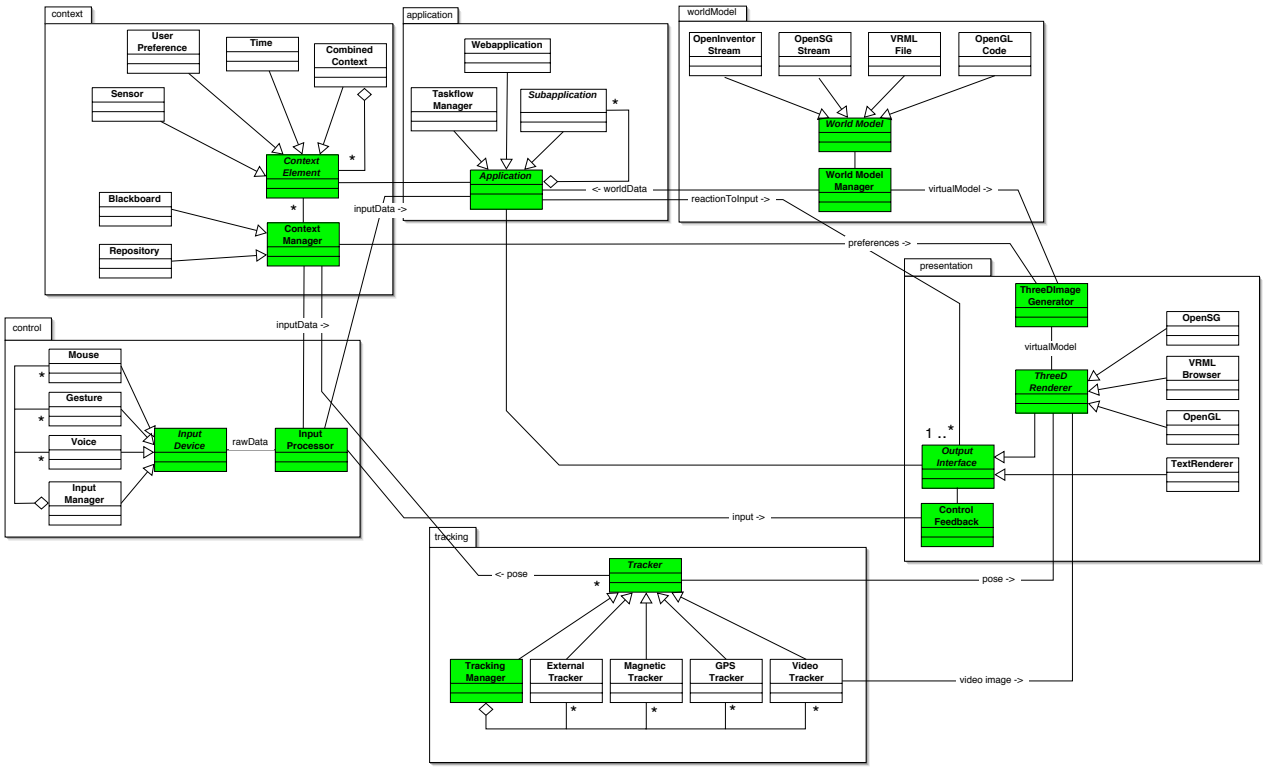


Figure 1.3: General DWARF architecture

ponents, others facilitate development while a group exists due to the development process of the framework. That group arose because the interests in the previous projects remained on the development of other components and demonstrative setups for applications, these components were use in.

Application Specific and Conceptual During the initial development stage of a new framework, not all requirements can be met. Thus to write a full-blown application some static services might be required to make up for missing framework functionality. With a perfect, stable, and feature complete framework, services like that will be obsolete.
Conceptual services demonstrate basic functionality and allow developers new to the framework to quickly familiarize themselves with it by looking at the code.

Testing A testing service assists the developer in debugging other services by simulating not yet fully implemented partner services by providing fake data. A tracking service could for example be tested with a black box test [8], made up by a test service which simply receives and displays the tracking data.

Task-focused Task-focused services are usually low-level services. They provide information on dedicated tasks. They can be configured in the range of their activity. Trackers

for instance, provide position/orientation information on certain objects. Which objects to track can be specified, but the tracker always will track objects and provide their location information.
In DWARFs current state an ART[5] system is such a task-focused service as well as for example a sound-player service.

Generic Components configurable to handle various interfaces and data-types. In contrast to task-focused components, these services provide a specific basic functionality on workflow activities that is configurable to required tasks. Their interfaces are adjustable to receive and provide task required data-types.
In this group the framework provides the User Interface Controller.

A goal of the development of a reusable framework for mobile Augmented Reality applications is to decrease the amount of components classifiable to the application specific area and to increase the amount in the other classes supplying configurable services.
Actual and new applications are made up by a combination of a set of configured generic services which require task-focused services during runtime. Current existing applications also rely on application specific components and new ones will, too. But their amount is intended to decrease in future.

1.4.2 A Requirements Generating Project

Ongoing development in university projects produced a usable framework with various components. But the DWARF framework is still far from being complete.

New properly calibrated input and output devices were needed. Also model for configuration and data persistence was required. Systems acting in intelligent environments also need components enabling the user to select services and providing Augmented Reality views on 3D-objects. The mobility aspect shall receive a contribution in form of a user-attached context aware service.
Finally, as research projects often result in unusable systems, human-computer interaction components should be evaluated for usability. So that the accustomization of these components to unskilled users takes not a great burdon to them.
The infrastructure also required major improvements, to provide new generic middleware features like multiple access via shared memory, template-services, and dynamic configuration bypassing[6].

These technologies extend the framework making it more suitable for future projects. During development of the mentioned project, new questions arose which make further

[5] Advanced Real-time Tracking - a commercial optical tracking system
[6] Though a documenting paper about DWARF middleware is being written, these features are documented on the developers board only at the time of this writing. Additionally the feature implementation is still in an experimental phase

improvements on the components possible.

While developing reusable framework components for a distributed dynamic system, the requirements for possible real applications should be kept in mind. The resulting architecture should be flexible enough to not only allow the development of the proposed new application but also support the development of completely different DWARF applications. Both approaches keep up continuous requirements engineering.
During the applications development process in the concrete project the requirements for the framework components get validated.

A new team project was planned in July 2002 to provide a number of missing components. The aggregation of multiple student study papers into team projects proved to be very rewarding for all participating members in the history of DWARF. Members learned about team-work, building large systems and could practically experience their skills. Also the production process is more interesting than in a solo project, because a system is built, on which others rely and that will be reused by other projects.
Keeping this in mind, the ARCHIE project was founded in a workshop in Konstein in the summer of 2002. In combination with the current requirements for the framework, suitable scenarios were found to give application specific requirements for the production of the new components.

1.5 ARCHIE

This section introduces the ARCHIE project. The name is an acronym for **A**ugmented **R**eality **C**ollaborative **H**ome **I**mprovement **E**nvironment.

ARCHIE is an interdisciplinary project between the Universität Stuttgart represented by Manja Kurzak, a student graduating in architecture [17] and a team consisting of seven members from the Technische Universität München. Manja Kurzak provided information about design processes and the planning of the construction of e.g. public or private buildings. Her information is summarized in the following problem statement section.

This project provided a starting point for requirements elicitation on the different single projects of all team members. It was intended to use ARCHIE in a prototypical implementation to give a proof of the requested components and their underlying concepts. To build an application with full functionality for architectural requirements was never a project goal.

The realized concepts have been shown to stakeholders like real-world architects in a live demonstration of multiple scenarios. In new areas like Augmented Reality new requirements are often generated by the client when the capabilities of the technology get apparent. To prove the flexibility and extendibility of the new DWARF components the chosen scenarios are partly independent.

1.5.1 Problem Statement

A *Problem statement* is a brief description of the problem the resulting system should address[8].

Usually a number of people with different interests are involved in the development process of a building.
The potential buyer has to mandate an architectural office to initiate the building process because the process is too complex to handle for himself. A mediator is responsible to represent the interest of the later building owners towards the architect. The architect assigns work to specialized persons such as for example, technical engineers for designing plans of the wiring system.
Although the later building owner is the contact person for the architects office, he is only one of the many stakeholders interested in the building. Furthermore landscape architects have to approve the geographic placement of the new building in the landscape. Last but not least a building company has to be involved as well.

The architectural process itself is divided into multiple activities which are usually handled in an incremental and iterative way, as some specialized work goes to extra technical engineers, who propose solutions, but again have to reflect with the architects office.
After taking steps of finding and approximating the outer form of the building fitting in it's later environment, the rudimentary form is enhanced to a concrete model by adding inner walls, stairs and minor parts. This is followed by adding supportive elements to the model like water- and energy-connection, air-conditions, etc. As mentioned, this work always has to be reflected to the architect, because problems might occur during the integration of the different building components. For example the layout of pipes might interfere with the layout of lighting fixtures or other wiring. Spatial representation enhances pointing out problematic situations.
When the plans are nearly finished, the builder needs the possibility to evaluate the plan feasibility and if in the end all issues are resolved, all necessary plans can be generated and the builder can start his work.
In addition to that the building owner always needs view access to the model during the design phase. He wants to see his building in its environment. End users should be given the option to give feedback during the design phase too. So, the architects office receives feedback from many participants about their plans.

There are some entry points for Augmented Reality. The ARCHIE project delivers proof of concept for these aspects.
The benefits of the old style architectural design with paper, scissors and glue allows direct spatial impressions, while modern computer modeling does not provide these feature. Augmented Reality can bring these back to computer modeling.
Since preliminary, cardboard box models can not get scaled to real size and virtual models reside on a fix screen, public evaluations are difficult to handle. Via abstraction of position and orientation independent sliders could be used to modify views. But 3D steering of virtual viewpoints is not as intuitive as just turning a viewers head. Adding tangible objects as cameras provide familiar access to evaluation features.

User interactions with modern architectural tools require practice. So intuitive input devices would be useful.
Also in place inspection of building plans and models would be a great benefit for all participating persons.

1.5.2 Related Work

This section lists research projects of other groups working in the Augmented Reality as well as in collaborative systems domain. Although focus remains on architectural tasks, the introduced projects aim on consulting our team in ideas and concepts transformable to the ARCHIE project.

The international project *Spacedesign* [10] resulted in a comprehensive approach using task-specific configurations to support design workflows from concepts to mock-up evaluation and review. The implementation allows free form curves and surfaces. Visualization is done by semi-transparent see through glasses which augment the 3D-scene. The approach allows cooperation and collaboration of different experts.
This project focuses on a stationary setup, useful on the task of car development, while ARCHIE should also be usable as a mobile setup.

A single system combining mobile computing and collaborative work [26] has been built by *Studierstube* [30]. The system, assembled from off-the-shelf components allows users to experience a shared space, while there are still synchronization requirements for late-joining partners. The result is far from a reusable context aware framework since it does not provide any kind of location awareness.

The *Collaborative Design System* [3] developed by Tuceryan et al. at the ECRC[7] demonstrates the interactive collaboration of several interior designer. The system combines the use of a heterogeneous database system of graphical models, an Augmented Reality system, and the distribution of 3D graphics events over a computer network. As shows in figure 1.4 users can consult with colleagues at remote sites who are running the same system.

An architectural application [35] developed by Tripathy allows the user to view an architectural model within an Augmented Reality environment. The object model can be imported from any CAD[8] application, but it can not be modified. Though several additional information of real world objects can be seen, like the wiring of the lighting, or maintenance instructions for changing the bulb. Within the maintenance record the user even can see when the bulb was last replaced.

Webster et al. developed an application for construction, inspection, and renovation. The idea of supporting architectural tasks with Augmented Reality is not new but has

[7] European Computer-Industry Research Center
[8] Computer aided design: www.cad.com

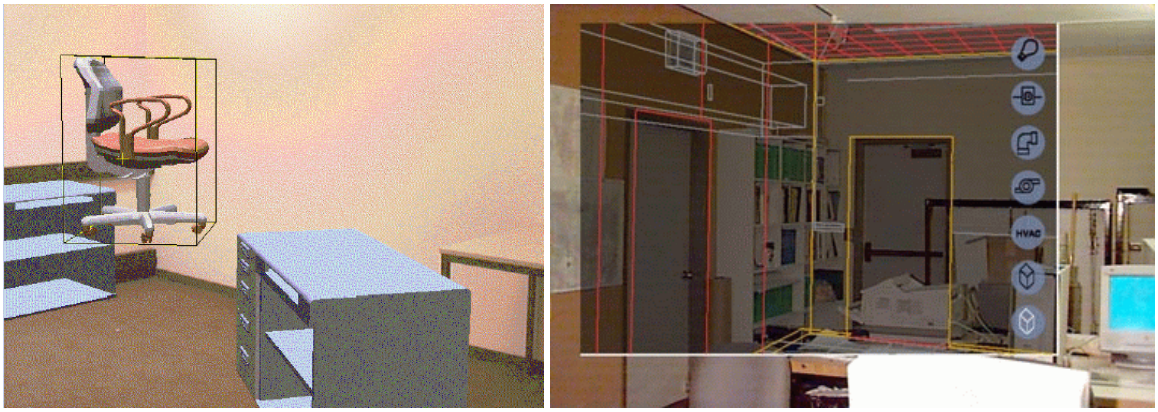


Figure 1.4: Left: *Collaborative Interior Design* [3] local user lifts chair while a remote user moves the desk; right: *An Application for Architecture* [35] overlapping a real office room with a CAD model

already spawned testbed projects such as "Architectural Anatomy" [41]. Architectural Anatomy leverages Augmented Reality by giving the user a x-ray vision which might e.g. enable maintenance workers to avoid hidden features such as buried infrastructure, electrical wiring, and structural elements as they make changes to buildings and outdoor environments.
The prototype application tracks the user with an ultrasonic tracking system and uses a head-mounted display for monocular augmented graphics. The project aims at building systems that improve both the efficiency and the quality of building construction, maintenance, and renovation.

1.5.3 Scenarios

A *Scenario* is a concrete, focused, informal description of a single feature of a system. It is seen from the viewpoint of a single user [8].
This section describes the Augmented Reality relevant scenarios of the ARCHIE system. The following list does not describe a full architectural system, because the ARCHIE project is only intended to be a baseline for the development of reconfigurable DWARF services.

Scenario: **Selecting Current Task**

Actor instances: `Alice:User`

Flow of Events:
1. Alice enters her laboratory which contains the necessary environment for the ARCHIE application such as a *ARTtrack 1* tracking system and a running *service manager* on at least one computer. Furthermore she is wearing a backpack with several objects mounted on it. There is for example a laptop that provides the viewing service for her HMD. She also holds an *iPaq* in her hand.

2. As her *iPaq* attains the range of the wireless ARCHIE-LAN, its *service manager* connects to the one running in the laboratory, and they exchange their service information to each other. Now the *selector service* knows the available applications, and the menu pictured in figure (1.5) is displayed on the iPaq.
3. By selecting either entry and confirming, Alice can start the desired task.

Figure 1.5: The ARCHIE selection menu displayed on the *iPaq*

Scenario: **Calibrating the Devices**

Actor instances: `Bridget:User`

Flow of Events:

1. When she starts the calibration method with her iPaq she also needs to have the 3DOF pointing device in her hand.
2. Bridget can now see the current virtual 3D scene not calibrated on the 2D image plane. In addition to that the calibration scene appears superimposed in her HMD, too. And she is asked to align the peak of the 3D pointing device with the corresponding 2D image calibration point. Once Bridget aligned the points properly, she confirms the measurement by touching her touch pad glove.
3. As the calibration method needs at last six measuring points to calculate the desired projection parameters, Bridget will be asked to repeat the last step for several times.
4. After confirming the last calibration measurement the newly calculated calibration parameters will be transmitted to the viewing component.

5. Now her HMD is newly calibrated and can augment her reality. So the tracked real objects can be overlayed by corresponding vitrual objects in front of her.

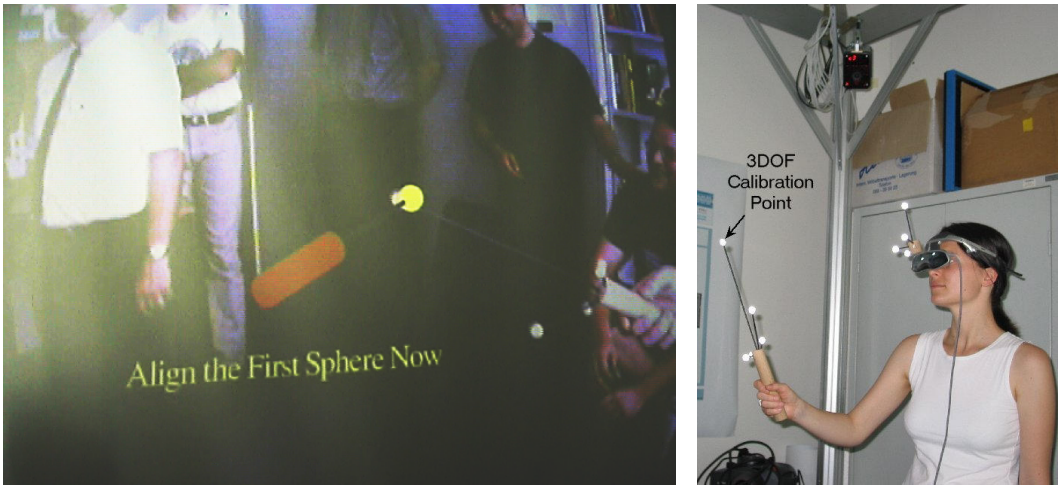


Figure 1.6: HMD calibration with a pointing device

As the working environment is calibrated and ready for use, two architects want to perform a collaborative task: They want to develop the outer shape of a new building.

Scenario: **Modeling and Form Finding**

Actor instances: `Charlotte, Alice:User`

Flow of Events:

1. Alice and Charlotte start the ARCHIE modeling application and their HMD viewing services.
2. As the system is initialized, both see the environment of the later building site.
3. Alice takes a tangible object, moves it besides another already existing building and creates a new virtual wall object by pressing the create button on her input device.
4. Charlotte takes the tangible object, moves it to the virtual walls position and picks up the virtual wall by pressing the select button on her input device.
5. Charlotte chooses the new position of the wall and releases it from the tangible object.
6. Both continue their work and build an approximate outer shape of a new building.

Scenario: **Mobility - Location Awareness**

Actor instances: `Dick:User`

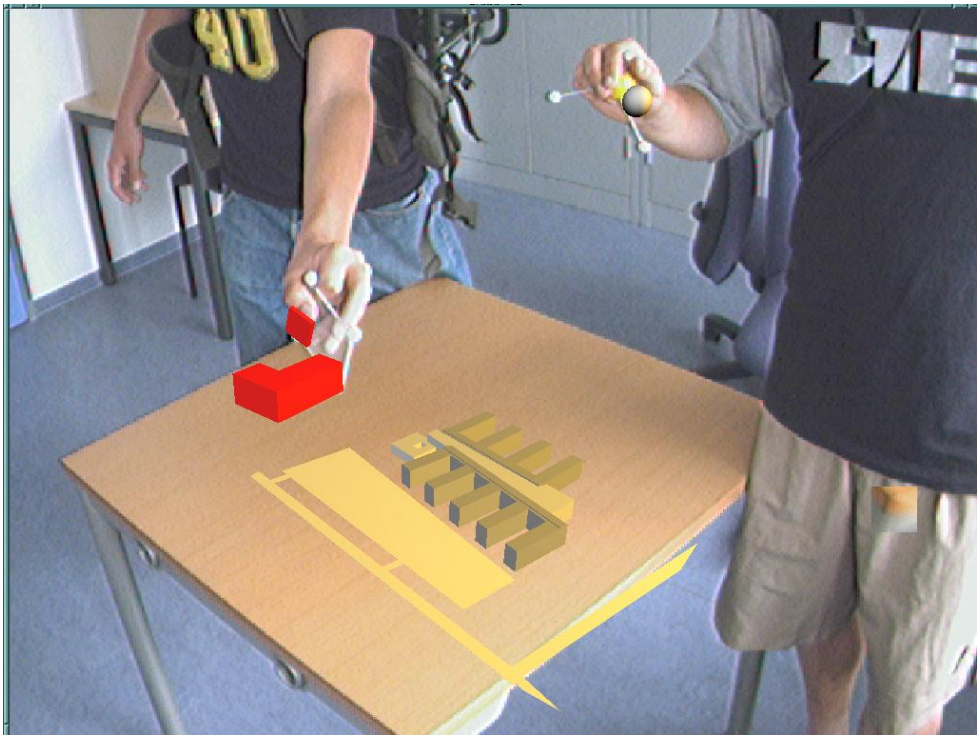


Figure 1.7: Modeling and Form Finding

Flow of Events:

1. Dick starts the location-awareness subsystem based on the ARToolkit [15], running on his laptop attached to his backback. In addition to the former setup an iBot camera is mounted on his shoulder to deliver video images.
2. The system is configured with the current room information. The information consists of the current room and the outgoing transitions (doors) to other rooms. A pie-menu giving information about the current services of the room appears on the laptop.
3. Dick exits the room while detecting an ARToolkit marker attached to the door with his iBot camera. The system changes its configuartion according to the new state (new room). The old information is dropped and Dick has a new set of services availabe now which can be used in the current environment dynamically. The pie-menu is updated.
4. Dick is able to exit and enter rooms and is enabled to use the corresponding services and room environment.

After different other Augmented Reality supported tasks are done, a group of later building end users visit the architects office, going to become introduced to the plans of the architects office.

Scenario: **Presentation**

Figure 1.8: Hardware setup for the location awareness

Actor instances: `Alice:User`

Flow of Events:
1. Alice starts the system, but instead of the previous used HMD, now a video beamer view is started, providing scenes as seen from a tangible camera object.
2. Alice takes this camera and moves it around the virtual model of the planned building.
3. The public can get a spatial understanding of the proposed building which is displayed on the beamer screen. The model shown in figure 1.9 is rendered in anaglyphic red-cyan 3D. For a realistic 3D view the visitors need to wear the corresponding red-cyan glasses.

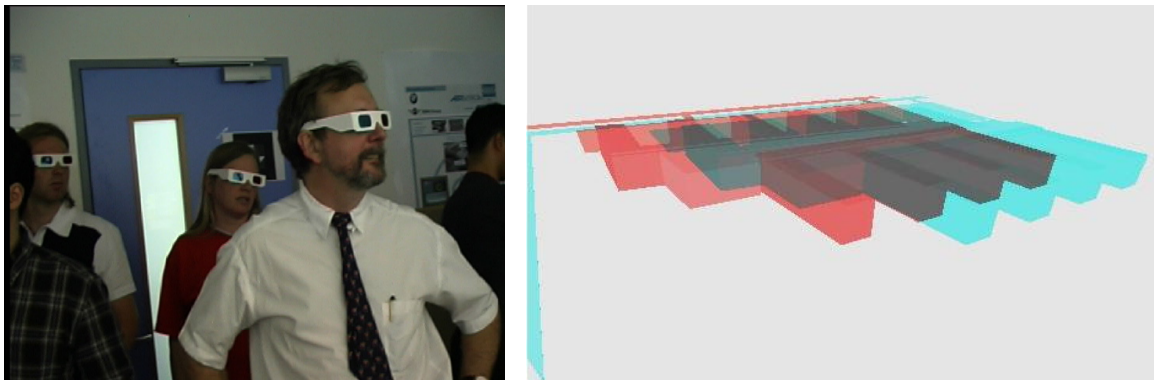


Figure 1.9: Presentation of a planned building to an audience

Scenario: **User Interaction Evaluation**

Actor instances: `Felicia:User, Gabriel:Evaluation Monitor`

Flow of Events:

1. Gabriel starts the ARCHIE application and configures it for an usability evaluation task.
2. He sets up the logging system and initializes it with the study and task name Felicia will be performing for an unambiguous log file.
3. After briefing Felicia appropriately, she is asked to perform a number of tasks which are being monitored.
4. The logging system is automatically taking task completion times while incrementing the task counter too, so Gabriel can fully concentrate on Felicias reactions to the system.
5. While Felicia is performing her tasks, Gabriel observes a number of real-time, updating charts which visualize her performance by e.g. applying standard statistical functions.
6. During the course of the study, Gabriel is always fully aware of what Felicia is seeing in her augmented display by looking at a special screen which duplicates Felicias HMD view.
7. Felicia is debriefed after she has completed posttest questionnaires handed out by Gabriel.

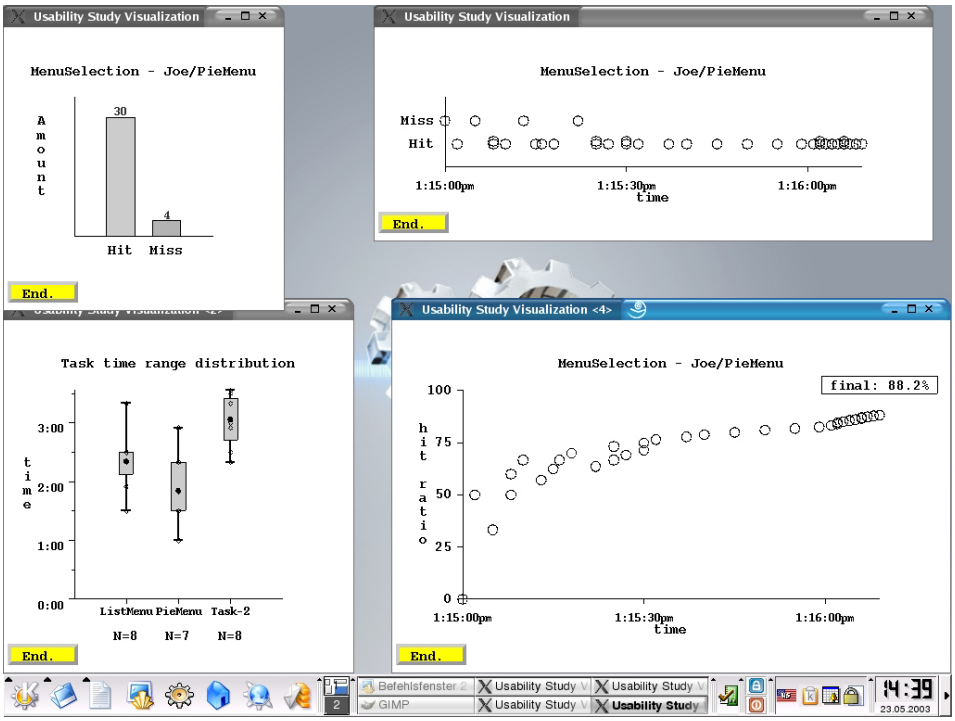


Figure 1.10: Live visualization of user performance in usability study

1.5.4 Requirements

This section describes the requirements our team elicited during the startup phase of the ARCHIE project. The methodology described in [8] was used to refine them step by step.

1.5.4.1 Functional Requirements

Functional Requirements describe interactions between a system and its environment [8]. They are independent from the implementation, but can get realized almost directly in code. This section declares the *Functional Requirements* for the ARCHIE system.

Modeling process

Helping Wizard Architectural applications have many wide ranging functions. Desktop versions provide menus, toolbars and context menus. Entering the Augmented Reality domain will deprecate some functions by intuitive interaction, but still an option for selection of available functions is necessary.

Moving and Placing Objects For a good spatial understanding of virtual 3D-building models, these should be movable in an intuitive way. Rotating and moving functions should resemble the real world physics.

Interrupted Work The application must preserve the modeling states when the users switches back and forth between different applications.

Collaborative work

Shared Viewpoints Shared viewpoints are useful for other passive participants to watch the work of the designer. The system has to support the reproduction of a single view on multiple terminals. For a larger audience a videobeamer view would be even more useful.

Personal Views During the development process one architect might chose one certain submodel for editing which is then locked to his usage until he publishes the edited submodel again for his colleagues to see. So the changes can be be propagated to all participating viewpoints for consistency.

1.5.4.2 Nonfunctional Requirements

In contrast to the *functional requirements*, the *nonfunctional requirements* describe the user-visible aspects of a system [8]. These may not directly relate to the system's functional behavior. *Nonfunctional requirements* can be seen as overall requirements a system must fulfill. They influence multiple locations allover the later realization. In decomposition they can be seen as *functional requirements*, but this view would be hard to understand for the client during requirements elicitation.
This section reveals the *nonfunctional requirements* of the ARCHIE project that lead to the design goals of general DWARF framework components.

Augmentation

Correct in Place Alignment of Virtual Objects In order to have a Augmented Reality viewing device in architectural context, the viewing display must be calibrated so that virtual graphics are rendered in the position and orientation corresponding to the real world.

Three Dimensional Augmentation Spatial impressions of the outer shape of constructions such as buildings require three dimensional views on the virtual model.

Real-time It is a basic principle of Augmented Reality applications that the user can not distinguish between real and virtual objects. To uphold this illusion the view must update rendered objects at a high speed and accuracy so no significant differences can be seen compared to real objects in behavior. Therefore the tracking subsystem should provide adequate precision, too.

Convenient Wearable Devices Because the development of complex buildings is a long duration process, devices must be comfortable to wear and use. System output and input devices such as HMDs and gloves should be attached to the user in such a way as to minimize the loss in freedom of mobility.

Ubiquity

Mobility There could be more than one Augmented Reality enabled office in an architectural bureau, so the users Augmented Reality device should support mobility by wearability and provide the execution of the application wherever possible without requiring additional setup steps. This encourages better collaboration between participants.

Application Selection Dependent on Location Often there are different projects in a company, available only at certain locations. So there should be a dynamic selection of applications and tasks depending on the users current context.

Robustness In addition to omnipresence of computers the system must handle input data gracefully from possibly very large amount of users simultaneously. The system has to differentiate input devices by users to avoid ambiguous data streams.

The development for framework components also yield requirements.

Providing Service Functionality The services provided to the framework should fit in one of the architectural layers specified in the DWARF explaining section.

Dynamic Application Configuration Framework components may rely on context information. These services must be configurable via interfaces as described in [21].

Quality of Service Changes done by a user in a collaborative session must propagate to views of the colleagues. All views within the system participating on the same application need consistency. This aspect also applies to data not directly handled in a view, like internal service configuration.

1.5.5 System Design

An overview over the architecture of ARCHIE is shown in figure (1.11).

1.5.6 Focused Tasks

A usable framework has emerged from DWARF, but it is yet far from being complete, if one can speak of completeness on such a wide ranged research topic at all. So the implementation of new components must focus on the individual research topics of our team members. This may result in some application specific components, but hopefully they will get generalized in later DWARF projects.

Since students make up the workforce, who require a precise subject assignment, approximate tasks where given to the majority of our group members from the beginning. So the framework was extended with the realization of narrow topics.
These two restrictions result in the following implementation selection:

Input Device A SEP that enables a touchpad mounted on a glove in DWARF.

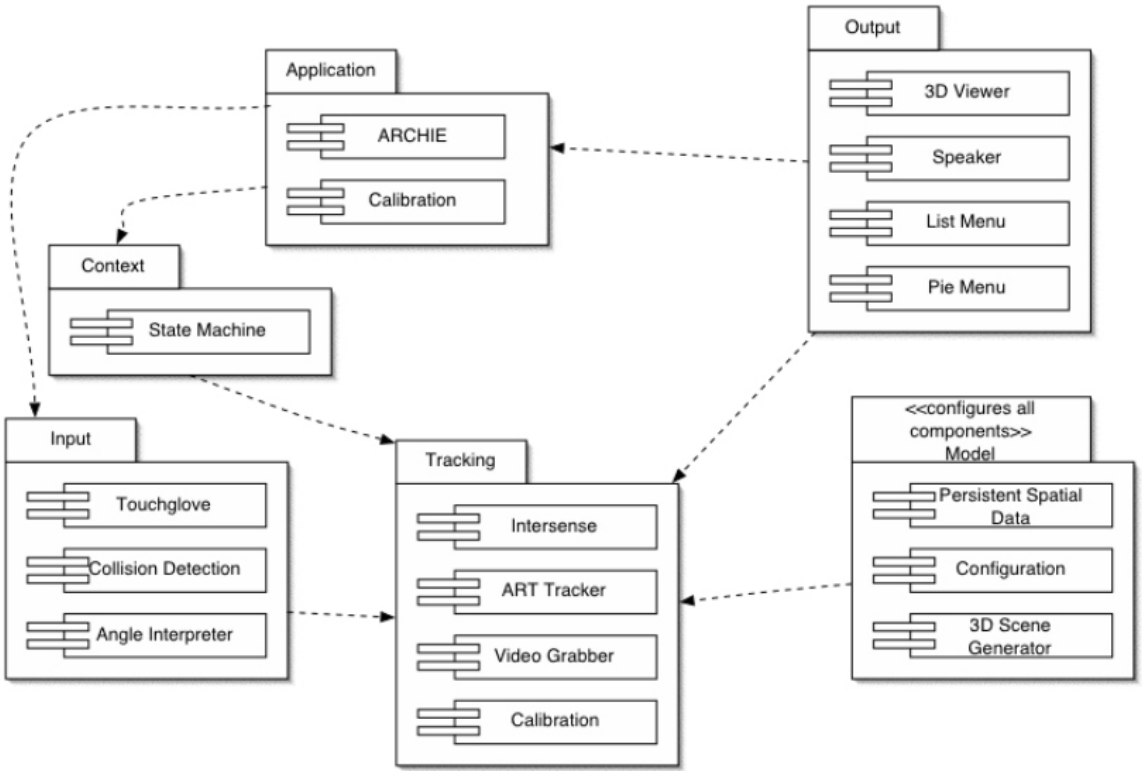


Figure 1.11: ARCHIE architecture

Output Device Another SEP provides a 3D rendering service which adapts to different output hardware.

Middleware Improvements Though the service-manager offers a variety of communication and connection interfaces, there is still a need for authentication of identifiable components. This is a SEP, too.

Location Awareness The last SEP within the team improves on optical feature tracking making location awareness possible.

Adjustment of Viewing Devices A diploma thesis that supplies calibration and configuration to different visual output devices on demand.

Managing the Users and Application Context Another thesis adding components to the framework for management of data of different types as 3D scenes and component configuration [34].
The proposed components are intended to hold information about real and virtual objects as well as to hold information about user context specific DWARF service configuration data.

Usability Evaluation of Human Computer Interfaces in Ubiquitous Computing The third thesis in our group provides a usability evaluation framework which can be leveraged to evaluate human computer interfaces within the Augmented Reality system to come to e.g. new Augmented Reality design guidelines. This includes, among others, components to take user performance measurements in DWARF, and visualizing this data appropriately to support the usability engineer monitoring a participant for usability study purposes.

To complete this list we also want to mention the work of our architectural client, who provided architectural requirements. This work has a direct focus on Augmented Reality. The thesis topic contains visualization of various kinds of radiation as warmth flow and refraction. A bargain, DWARF might hopefully provide in the future.

There are other components required for ARCHIE which do not fit in one of the above listed categories. These will get generalized hopefully in new projets.

Although the second part of this chapter focused on the ARCHIE projects, one will see the development of reusable components for the DWARF framework in all cases of our team members in the following chapters.

2 General Hardware Driver Framework for DWARF

2.1 Introduction

Input and output devices are the essential parts of Augmented Reality systems. On the one hand, you use devices which you can buy, examples are mice and graphic cards. This devices are usually delivered with a driver for one or two platforms. On the other hand, you have many self-developed device prototypes. In this case, usually no driver is available.
Even if you have a driver for your device, several problems occur: In many cases, the drivers does not fit for augmented reality systems. The standard hardware driver for touchpads of the Linux OS for example could not be used in the TouchGlove input device development, because this driver does not provide all hardware features. Another point is the integration into the DWARF framework. Often, it is not easy to integrate existing drivers in such a framework.
This results, that in a multi-platform augmented reality framework, you will be often confronted with developing a new hardware driver for one or more platforms.

2.2 Requirements

The requirement analysis identified the following requirements:

- The architecture has to be independent from the driver framework of a specific platform.
- The framework has to be modular enough to avoid any redundant parts in a driver.
- The overhead should be minimized to guarantee maximum performance.

Below, I will describe my idea of a lightweight driver framework which in my opinion satisfies this requirements.

2.3 Related Work

Existing augmented reality frameworks usually do not provide a general driver framework for all hardware drivers used in the applications. This forced me to get inspirations from

other sources. The I/O Kit from Apple [4], which is the driver framework for Mac OS X, provides a flexible framework for all kinds of drivers. To fit this framework for augmented reality purposes, I reduced its design to a lightweight architecture.
Central to the design of this driver framework is a modular layered architecture that models the hardware by capturing the relationships among the multiple pieces (hardware and software) involved in an I/O connection. The layers of the connection are stacked in provider-client relationships. This layering is similar to the OSI network stack.
There are two types of driver objects. The first is called *nub* in the I/O Kit terminology. It represents a controllable entity, such as a device or logical service. A nub may represent a bus controller, a disk, a graphics adaptor, or any number of similar entities. The second is the *driver*, which manages a specific piece of hardware. Usually, the driver inherits from a nub to provide its functionality over the generic nub interface.

2.4 General Driver Framework

My approach uses the basic concepts, which are implemented in the I/O Kit by Apple. The general driver framework has a layered architecture with two types of objects, the nubs and the drivers. The nubs fulfill the functionality of an abstract interface and the drivers are concrete implementations of that interface. Figure 2.1 shows an example for a Touchpad driver. The bottom layer is a specific driver for the serial port written for Linux. This driver inherits the RS-232 Protocol nub. Now, the Synaptics Touchpad driver can access the serial port via the RS-232 Protocol nub, which is the same for all RS-232 implementations over all platforms. The touchpad driver itself inherits the Touchpad nub, which provides the generic functionality of a touchpad to the application. This modular design minimizes the amount of redundant software development because every driver developer can reuse modules he needs. Additionally, this idea of nubs and drivers helps you to develop platform independent, because you access drivers only through a generic interface which is the same on every platform. For standardizing the interface of nubs, I created a nub hierarchy, which is shown in figure 2.2.

On the one hand, this hierarchy helps on classifying the nubs. On the other hand, it ensures that methods which are used in all nubs of a family (e.g. the family of protocols), will have the same signature.

2.5 DWARF specific Realization

After introducing the general driver framework, I will show in this section, in which way the framework can be implemented into DWARF.

In DWARF, the atomic units are services. A service provides an interface and can be dynamically connected with other services. This paradigm fits perfectly to the general driver framework introduced above. A nub can be represented as an interface, a driver as a service. The communication between the drivers is completely encapsulated into the DWARF middleware and even allows you to distribute the different layers over several computers which can run different platforms. Additionally, different layers can use the same instance of

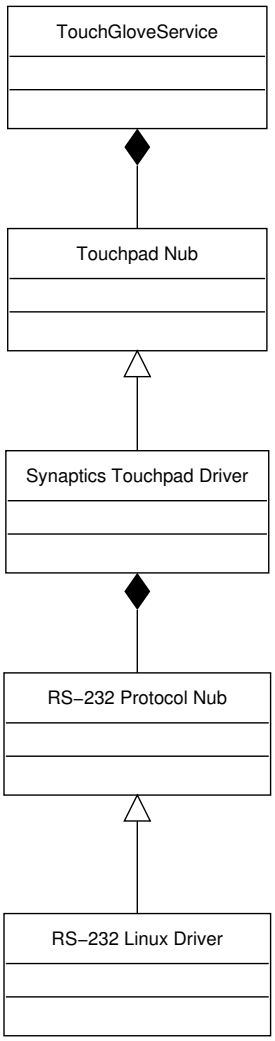


Figure 2.1: Driver layer example for touchpad

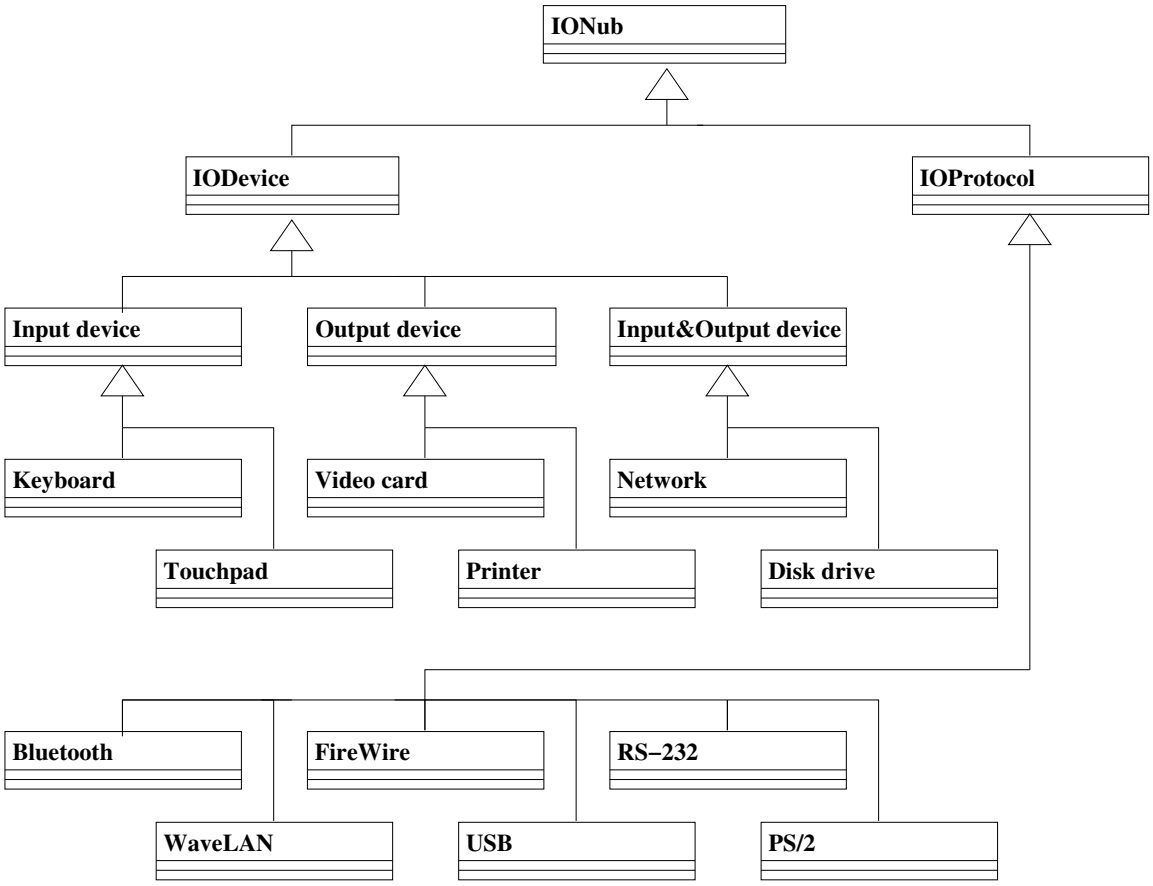


Figure 2.2: The nub hierarchy

an underlying driver without any additional coding, because this is handled by the DWARF middleware, too.

With the technique of starting more than one service in one process, performance lacks arising through the communication between DWARF services are avoided.

3 Dynamic Matching of Input Devices in Mobile Environments

3.1 Introduction

Imagine the following situation: A businessman arrives for a guest talk at the conferencing room. He wants to combine a beamer presentation with writing to a white board. At one hand, he wears a TouchGlove input device connected with his handheld. When entering the room, his handheld shows him the possible services the room provides. To avoid running to the light switch every-time he moves from the presentation to the white board, he simply connects the light dimming service of the room with one slider on his TouchGlove. Additionally, he connects one button on the TouchGlove with the presentation computer to go forward in the slides by simply clicking on this button. Now, he can do his talk without interrupts for running to the light switch or using a special device for changing the slides.
For developing AR applications that provide such functionalities, there has to be a way to connect input devices flexibly with input data consumers in mobile environments. In this chapter, I will present an idea for achieving this flexibility in using and configuring input devices.

3.2 Requirements

The requirements analysis identified, that the main problem is the matching between the interface the input device provides and the interface the input data consumer (e.g. the application) needs. In many approaches, this problem is tried to solve by ordering the input device interfaces into a hierarchy and then find a good matching algorithm. But these approaches are not flexible enough to handle all the different imaginable use cases of input devices in mobile and ubiquitous environments. Even in complex stationary environments with lots of tangible devices, such approaches restrict the user in interacting with the system the way he wants. In the following list shows you the four major requirements:

- Dynamic Configuration: This means, that input devices and the AR application can be configured dynamically. A dynamic configuration of the application is required for exchanging input devices on-the-fly.
- Device Fusion: Device fusion means, that it is possible to combine two or more devices to a virtual device. An example is the combination of a gyro and the TouchGlove input device.

- All kinds of Devices: This point is about the coverage of supported devices in the framework. The solution has to fit for all kinds of input devices, not only for some categories like mouse input devices or tracking devices.
- Easy Device Misuse: This means, that it should be possible to misuse an input device for something it was not intended for. An example here is to use directly the X axis rotation of a tracked device to control a slider in the HMD.

3.3 Related Work

Owlal [23] describes in his thesis an API called Unit that provides uniform mechanisms for controlling and customizing the data flow in an application. It focuses on the use of Unit as a layer between input devices and applications where interaction technique and behavior can be specified. In this approach, functionality is specified in self-contained units, which are connected statically with a code snippet into Unit Networks.
VRJuggler [14], a development for Virtual Reality applications, uses common interfaces for input devices. These devices are divided into several broad categories. Tracking systems for example are represented by Position devices, which provide one ore more sets of positions and orientation data.
MagicMeeting [24] describes a collaborative tangible augmented reality system. It is part of VRIB, a virtual reality interaction toolbox. This approach uses event sinks and sources with unique ids. The connections are defined externally using XML. Input devices correspond to event emitting device abstraction components. The event types can be converted with adaptors (e.g. ”SensorPose” of a marker tracker to ”Pose” of a ray picker).
OpenTracker [32] is an open architecture for reconfigurable tracking based on XML. It provides device abstraction with interfaces which are used for groups of devices.
For achieving maximum flexibility in connecting input devices with input data consumers, the requirements described above have to be fulfilled. The following table gives you an overview about these requirements and whether they are implemented in the different frameworks or not:

	Unit	**VRIB**	**VRJuggler**	**OpenTracker**
Dynamic Configuration	no	yes	yes	no
Device Fusion	yes	no	no	yes
All kinds of Devices	yes	yes	no	no
Easy Device Misuse	yes	no	no	yes

As you can see in the table, none already existing approach satisfies all requirements. This caused me to develop a new approach.

3.4 Matching

For achieving maximum flexibility, an input device in my environment has two parts: The first is the so-called syntactic interface. This matching process sometimes finds different matching possibilities. This happens for example, if several devices with same functionalities are available for one input data consumer. In the second step of the matching process,

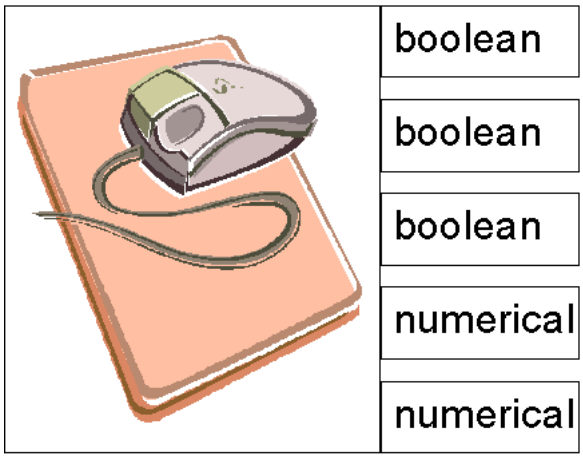


Figure 3.1: Example of 3-button mouse syntax

one matching possibility has to be chosen based on the semantic interpretation of the syntactic interface and the chosen input device is connected with the consumer.

3.4.1 Syntactic Matching

3.4.1.1 Introduction

What is the syntax of the data an input device provides ? Most input devices can be decomposed into a small set of different entity types. For example, a 3-button mouse can be decomposed into two numerical values (X and Y) and three boolean values (the buttons). These entity types are completely free of semantic information and define only the data type. The syntax for the 3-button mouse example is "2x numerical + 3x boolean".

3.4.1.2 Syntax Taxonomy

A critical issue is to define the set of entity types in a way, that the matching process does not deny matchings which could be useful in a special scenario but on the other hand does not allow matchings which are useless or confuse the user. For this, I tried to define a taxonomy of entities which satisfies both (figure 3.2).

- Boolean: All discrete data can be decomposed into booleans. Buttons, for example, are mapped to boolean.
- Limited Range: Every analog value with a lower and upper bound can be mapped to a Limited Range entity. This entity has always a range between 0 and 1. Examples are sliders or angles.
- Unlimited Range: Every irrational number which has only one or no bound. Examples for this are coordinates.
- String: Text strings.

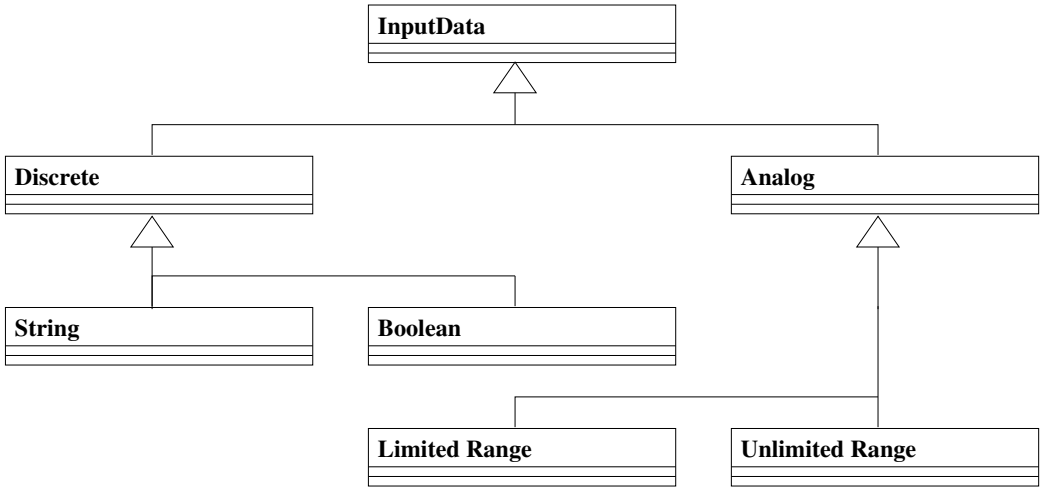


Figure 3.2: The Syntax Taxonomy

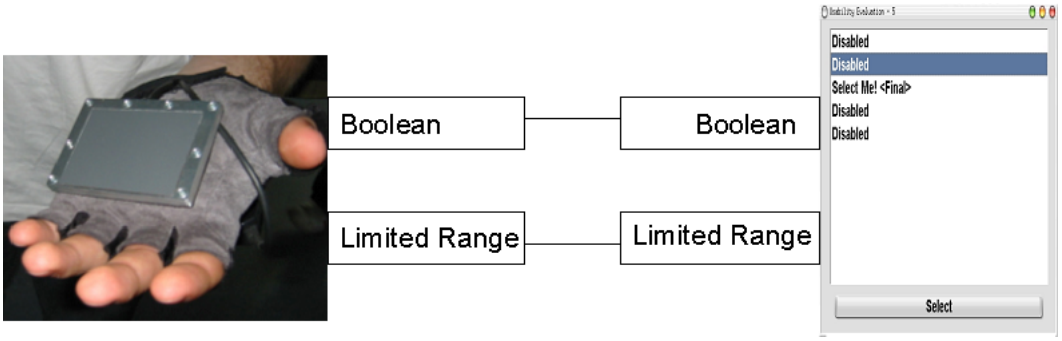


Figure 3.3: Example of equality matching between the TouchGlove and a list menu

3.4.1.3 Matching Methods

The syntactic matching process should be done automatically. For this, three different types of matching are possible:

Equality Matching Matches, if the syntax of the supplier (the input device) and the consumer (e.g. the application) is equal. You can find an example in figure 3.3. This matching is very strict.

Subset Matching Matches, if the syntax of the consumer is a subset of the syntax of the supplier. This matching allows to use only a part of the functionality of an input device. Figure 3.4 gives you an example.

Intersection Matching Matches, if the intersection between the supplier and consumer syntax is not empty. This is the most powerful matching but produces a lot of matches. It can

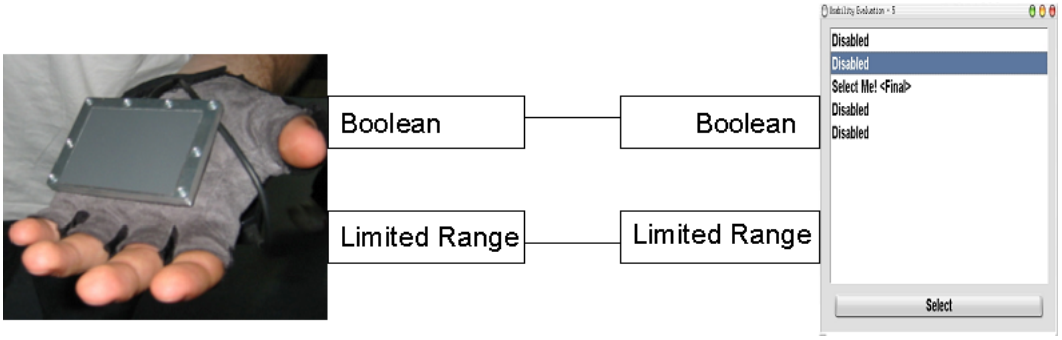


Figure 3.4: Example of subset matching between the TouchGlove and a slider shown in a HMD

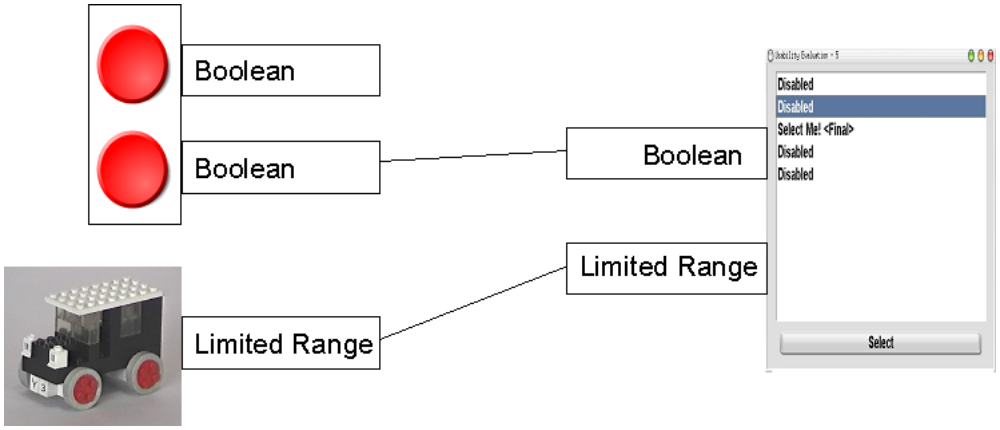


Figure 3.5: Example of intersection matching between a two-button device, a tracked tangible device and a list menu

be used to combine several input devices into one virtual input device (e.g. using two mice for navigating in a 3-dimensional space). An example for intersection matching is shown in figure 3.5.

3.4.2 Semantic Matching

If the syntactic matching resulted more than one result, the second step of matching is the semantic matching. This step should be done or at least supported by the user. The idea behind this is, that the user can connect input devices with consumers (e.g. the application) in the way he wants (if their syntax matches). What is the semantics of an input device ? It is the semantics of the entities plus the correlation (if some exists) between them. Taking the 3-button mouse as an example again, this means, that X and Y is the semantics of the two numerical values and “left button”, “middle button” and “right button” is the semantics of the three booleans. The correlation is only a layout composition, which means, that the values are independent but all implemented in one device. Due to limited research time, I only tried to identify the topics which are important related to semantic matching:

- Semantics Hierarchy: Equal to the syntactic hierarchy, a semantics hierarchy is necessary, too. This hierarchy should help using unique names for equal input device parts. Unique names are important for unsupervised matching.
- Correlations: It is important to know about the correlation between different values of one input device, because in some cases, the values depend on each other (e.g. angles in special formats). Especially for combining two or more devices to a virtual one, this topic is quite important.
- Matching Methods: Similar to syntactic matching, it is important to think about different matching methods in semantic matching. One critical issue in this area are the above mentioned correlations.

3.5 Realization in the DWARF Framework

For the first step, I used only syntactic matching. If more than one connection was possible, the DWARF selector displayed a menu to the user with all possible entries. For each leaf node in the syntactic hierarchy, I created a separate CORBA IDL. The common structure of this IDLs is:

```
struct InputDataXXX {
  InputDataEventType eventType;

  xxxType xxxValue;
};
```

InputDataXXX can either be InputDataBool, InputDataString, InputDataAnalogLimited or InputDataAnalogUnlimited. The eventType is an enum with two possible values: InputDataState and InputDataChange. InputDataState means, that the value in xxxValue represents the current state. With InputDataChange, the value in xxxValue represents the current state, too, but this indicates, that this state has changed into this one. An example for this is a button click: A button click can be represented by a button release action. The state would be boolValue=false (the button is not pressed anymore). The eventType would be InputDataChange because the button state changed into this state. The id of the input item (e.g. slider1, buttonOK,...) has to be placed in the event header field event.header.fixed_header.event_name.

In detail, the four different IDLs are:

- InputDataBool Used for discrete input data like a button click. xxxType=bool, xxxValue=boolValue.
- InputDataString Used for string input data. xxxType=string, xxxValue=stringValue.
- InputDataAnalogLimited Used for limited analog input data like slider data. All values have to be normalized into the range 0 - 1. xxxType=double,xxxValue=limitedAnalogValue.

- InputDataAnalogUnlimited Used for unlimited analog input data like coordinates. xxxType=double, xxxValue=analogValue.

An important question was how to model the needs and abilities for the DWARF framework[1]. There are different ways to model this and I will discuss three different approaches: One approach is to create only one ability for all kinds of input data. This turned out to be unusable in some situations. An example is a service which needs InputDataBool and InputDataAnalogLimited but wants to take this data depending on user wishes either from different sources (e.g. a gyro and a TouchGlove button click) or from one source (e.g. TouchGlove slider and button click). With only one need, this scenario produces lots of problems. Another point is: Sending events with different event structures (e.g. InputDataBool, InputDataAnalogLimited) over one event channel is not a good coding style.

A second approach is to create an ability for each entity of the input device. For example, a TouchGlove configuration with three sliders and two buttons would cause five abilities. This approach brings maximum flexibility. On the other hand, this maximum flexibility causes lots of collisions when the service manager tries to satisfy needs for input data. Imaging, for example, the TouchGlove provides four sliders. A service has a need for only one slider. But with which ability should the service manager satisfy this need ? This would cause lots of manual user action in the system. Another problem with this approach is the configuration of some services which use templates and generate their input data abilities dynamically. An example is the TouchGlove interpreter, which needs to load a profile before generating the input data abilities. But loading the profile from the configuration services requires existing abilities for the template mechanism.
The third approach is a trade-off between the first and second one. There is one ability for each input data idl. For instance, if an input device provides three sliders and two buttons, it has two abilities. One for InputDataBool and one for InputDataAnalogLimited. Each ability has an attribute called "numOfInstances" which holds the number of different input data items which will be handled over this ability. This number can be zero or greater. The advantages of this approach are: services, which should be used either with input data of different type from different sources or with input data from one source can be connected very easily. An example here is a pie menu, which needs a bool and a limited analog value. In one use case, both is provided by the TouchGlove, in another use case, the limited analog value is provided by a gyro and only the bool by the TouchGlove. A disadvantage is that interconnecting different input devices to one virtual input device is only possible with an extra service. Another disadvantage is: All services with an input data need have to decide with the input data item id whether this event is interesting for them or not.

[1] A special thanks goes to Asa MacWilliams and Christian Sandor for a great discussion about this point

4 TouchGlove DWARF Services

4.1 Introduction

The TouchGlove input device (figure 4.1) is a new input device developed by the Columbia University [7]. It consists of a half-glove and touch-sensitive surface device which is mounted on the palm portion of the half-glove. The physical design and the interaction method make it possible for the user to interact simultaneously with a wearable computer, as well as with objects and machines in the environment. Input is performed using both tapping and dragging motions of the fingertips on the surface.
I implemented a suite of services to connect the TouchGlove input device to the DWARF framework:

- The Raw Data Provider, which connects to the hardware and provides raw data events
- The Raw Data Interpreter, which transforms the raw data events into high-level events like ”button clicked” or ”slider moved” based on a so-called profile
- The Profile Creator, which is a visual tool for creating profiles

Please note, that these services are highly experimental and no standard usage has been established, yet.

4.2 Requirement Analysis

4.2.1 Problem Statement

The current problem of the TouchGlove is, that there is only a low-level driver for MS Windows available. This work should provide first a low-level driver for Linux and secondly integrate this device into the DWARF framework.
The system design has to be flexible enough for porting this driver easily to different operating systems and hardware. The design of the DWARF framework is important for the system design, too.

4.2.2 Scenarios

Here I will describe the scenarios I used when designing the TouchGlove service.

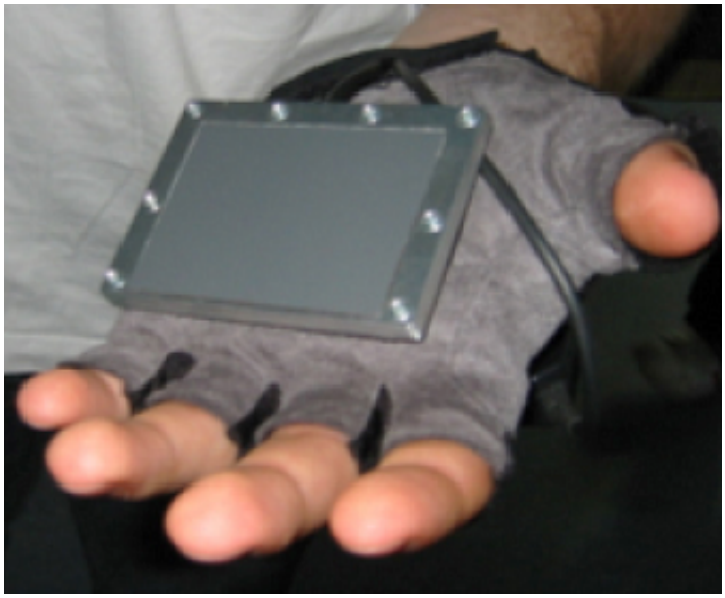


Figure 4.1: The TouchGlove input device

1. Scenario: The user wants to navigate through a menu with the TouchGlove input device. Therefore, the menu service and the TouchGlove service has to be started. The TouchGlove needs a configuration, that provides a slider-like and a button-like input (such a TouchGlove configuration is called *profile* in the future). He moves the slider up and down to choose the right menu entry and presses the button to select this entry.
2. Scenario: The user wants to modify the color of a virtual object in his HMD. For each color component (red, green and blue), he needs a slider-like input element on the TouchGlove surface.
3. Scenario: An administrator wants to create a new TouchGlove profile for an application. He uses a visual tool for creating and arranging input elements like sliders or buttons and stores the profile in a persistent data storage system.

4.2.3 Identification of Actors

Service Services, which need input data like button clicks, have to connect with the TouchGlove driver service.

User The user wears the TouchGlove and want to interact with the running DWARF application with this input device.

Profile Administrator The Profile Administrator creates the profile with is used by the driver to transform the raw data into high-level events.

4.2.4 Use Cases

Use Case: **Using TouchGlove as input device**
Initiated by: `User`
Communicates with: `Consumer Service, TouchGlove Service`
Flow of Events:
1. The *Consumer Service* and the *TouchGlove Service* is started.
2. The services are connected by the middleware.
3. A profile is loaded, which fits the input requirements of the *Consumer Service*.
4. The user performs actions on the TouchGlove input device, which are interpreted. The interpreted result is sent to the *Consumer Service*.

Use Case: **Creating a new profile**
Initiated by: `Profile Administrator`
Communicates with: `Storage Service, TouchGlove Service`
Flow of Events:
1. The profile creation tool and the *TouchGlove Service* is started.
2. The *Profile Administrator* creates a profile by arranging elements manually or with the TouchGlove device directly.
3. The new profile is stored in the *Storage Service*.

4.2.5 Functional Requirements

In this section, I describe the functional requirements for the DWARF TouchGlove framework.

Connecting with the hardware The TouchGlove driver has to connect to the hardware. The driver must be flexible enough to allow different hardware setups.

Providing different kinds of Raw Data The touchpad has different modes, which provide different kinds of data (e.g. relative or absolute coordinates). This data should be accessible for the whole DWARF system, because some services would like to use this raw data directly (e.g. a usability test system).

Converting Raw Data into High-Level Events In the DWARF system, the raw data is usually useless. Other services need events like ”button clicked” or ”slider moved”. For this, the raw data has to be interpreted and high-level events have to be sent. The driver should understand at least the elements ”button” and ”slider”. This elements are arranged on the touchpad surface. This arrangement is called ”Profile” in the following.

Allow Different Touchpad Profiles The TouchGlove driver should be able to change the arrangement of the different elements on the touchpad dynamically. Because for different applications, different elements are needed.

Creating the Profiles It should possible to create new profiles.

Storing the Profiles The profiles have to be stored in a way, that they can be accessed easily from different computers.

4.2.6 Non-Functional Requirements

Low Latency The latency between a movement of the user on the touchpad and the output of the high-level event should be low. Else, it could be difficult for the user to use this device intuitively.

Creating the Profiles Easily It should be easy and intuitive to create new profiles.

4.2.7 Pseudo Requirements

A pseudo requirement for the TouchGlove driver was our project team's agreement on using C++ as implementation language in all time-critical services. DWARF has to be used as communication framework.

4.2.8 Object Models

In this section, I will identify objects from the scenarios and use cases described above, forming the analysis object model for the TouchGlove service.
The object model is divided into "real-world" entity objects, boundary objects representing system's interface, and control objects representing interactions between the system and outside.

4.2.8.1 Entity Objects

Profile Profile objects are created in the Profile Creator and give the user the possibility to perform different input tasks with the TouchGlove input device. The profiles contain information about the number of elements and their arrangement on the touchpad surface.

TouchGlove Service This object should connect to the TouchGlove hardware and provide the input performed by the user to the input data consumers.

Raw Data Objects The data, which is provided by the touchpad hardware, is stored in Raw Data Objects. These objects are used for recognizing actions like ”slider has position xy” or ”button pressed”.

High-level Event Objects These objects contain the high-level input data like ”slider s1 has position 0.2” or ”button b1 pressed”.

Profile Creator The profile creator object class is used to create profiles. This profiles are stored in with a persistent storage service after creation.

Communication infrastructure For the communication between the services, a communication infrastructure is required. Since all DWARF services take advantages of the existing DWARF middleware, I have not modeled the communication part in detail.

4.2.8.2 Boundary Objects

Input Data Consumer and Storage The interface between all services in a DWARF system is standardized by a DWARF Service description.

Profile Creator The Profile Creator has a visual interface for user interaction. This creation tool provides the functionality ”new”, ”loading”, ”saving” and ”manipulating” profiles.

4.2.8.3 Control Objects

Raw Data Queue The TouchGlove Service needs a Raw Data Queue object, because the interpretation of raw data provided by the hardware only works with a set of raw data objects.

4.3 Related Work

[7] invented the idea of using a touchpad in a half-glove as a mobile input device. They used this device only for navigating through menus. This resulted a very limited dynamic configuration of the touchpad. The operating system they used was MS Windows.

4.4 System Design

4.4.1 Design Goals

In the design of the TouchGlove driver, I had to pursue several sometimes conflicting goals.

Creating the Profiles Easily It should be easy and intuitive to create new profiles.

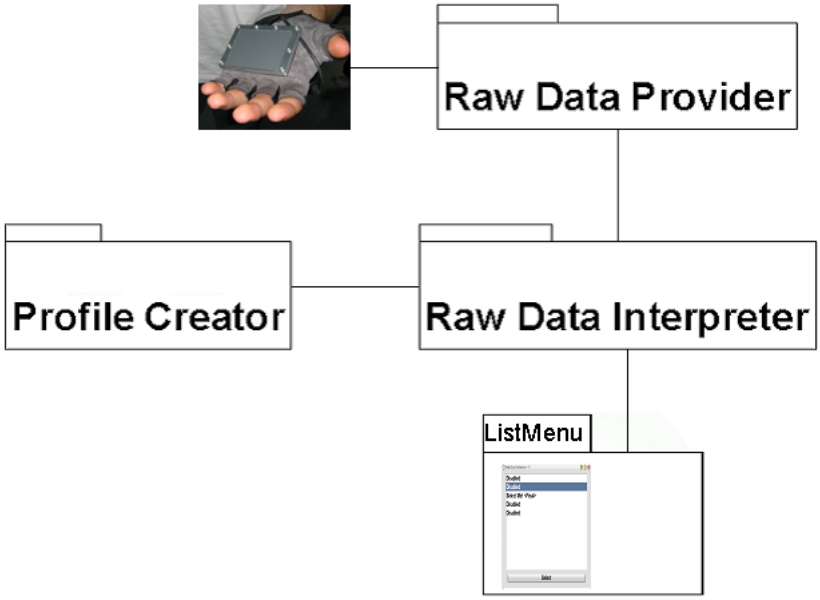


Figure 4.2: Subsystem Decomposition of TouchGlove

Low Latency The latency between a movement of the user on the touchpad and the output of the high-level event should be low.

Flexible Design To be able to port the TouchGlove driver easily to other hardware platforms and operating systems, the driver should by decomposed as much as possible. The paradigm introduced in the general driver framework chapter fits best for this design goal.

Unfortunately, the design goals ”Low Latency” and ”Flexible Design” provide conflicts. The decomposition of the driver in many small entities causes high network load and higher latency, which is caused though the communication process.

4.4.2 Subsystem Decomposition

The requirement analysis helped my to identify three subsystems (figure 4.2) :

Raw Data Provider The Raw Data Provider connects to the hardware provides the ability of sending raw data events.

Raw Data Interpreter The Raw Data Interpreter has a need for raw data and interprets this data. It sends high-level input events.

Profile Creator The profile creator is a tool for creating profiles, which are used in the Raw Data Interpreter.

This three subsystems will be implemented into three DWARF services.

4.4.2.1 Raw Data Provider Subsystem

The first component is the Raw Data Provider service. This service has to act as a kind of driver and must handle the communication between the touchpad on the TouchGlove and the software. The TouchGlove is connected to the serial port. This means for the component to establish a serial connection. The output of this component are events, which consist of the raw data the touchpad provides.

4.4.2.2 Raw Data Interpreter Subsystem

The second component is the Raw Data Interpreter Service. This service gets raw data as input and tries to identify user actions like “button pressed” or “slider moved” on the basis of a profile provided by the user. The output of this service are user action events, like “button b1 pressed” or “slider s1 moved to position 0.23”.

4.4.2.3 Profile Creation Subsystem

The third component is the Visual Profile Creation Tool Service. This visual tool is for creating the profiles the interpreter services uses. The user can either click the profiles together or use the TouchGlove to show the component directly where the buttons and sliders should be located by moving over the TouchGlove. For this, the component receives TouchGlove raw data provided by the Raw Data Provider. The profiles are stored in an XML file.

4.4.3 Hardware/Software Mapping

Figure 4.3 shows the deployment diagram of the TouchGlove services. The Raw Data Provider has to run on the computer, which is connected to the TouchGlove hardware. This is usually a wearable computer. The Raw Data Interpreter can run on a server machine, because no interaction with the user is necessary. The Profile Creator should run on a desktop PC, as the user has to interact with this visual tool extensively.

4.4.3.1 Third-Party Software Components

This section describes the off-the-shelf software components that are part of the TouchGlove services, and the hardware the services are designed to run on.

Serial Port connection For the connection to the serial port, the POSIX serial port driver of Linux is used, which is installed and configured automatically by the operating system.

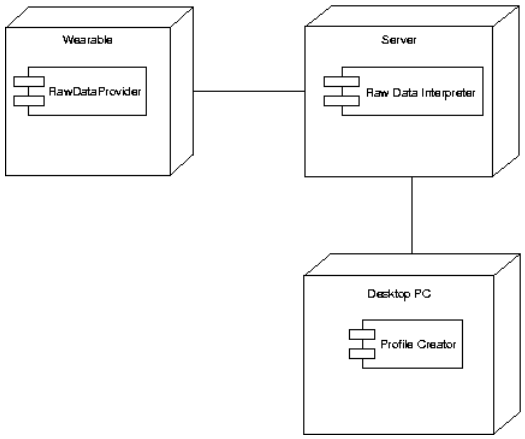


Figure 4.3: Deployment Diagram of TouchGlove

Trolltech QT For the Profile Creation Subsystem, a visual user interface was required. I used QT of Trolltech for this purpose.

XML Parser For parsing profile descriptions, which are written in XML, the TouchGlove services use the XERCESC parser.

DWARF framework All communication tasks are handled by the DWARF framework.

4.4.3.2 Hardware

The Raw Data Provider service can only run on hardware with a full serial port. My first intention was to connect the TouchGlove with the serial cable to an iPAQ to provide a mobile setup. But I had to abandon this idea due to hardware problems with the iPAQ. The reason is: For initialization, the touchpad needs a hardware handshake. But unfortunately, the serial port of the iPAQ does not allow hardware handshakes.

4.4.4 Persistent Data Management

In this section, I describe the persistent data stored by the TouchGlove services.

Profile XML The profiles are stored in an XML with the following DTD:

```
<! ELEMENT TouchGloveProfile (grid,slider*,button*) >
<! ELEMENT grid (xsize,ysize) >
<! ELEMENT xsize >
<! ELEMENT ysize >
<! ELEMENT slider (id, width, startrect, endrect) >
<! ELEMENT button (id, rects) >
```

```
<! ELEMENT rects (rect*) >
<! ELEMENT rect>
```

Here an example XML for one slider an one button:

```
<TouchGloveProfile>
  <grid>
    <xsize>16</xsize>
    <ysize>8</ysize>
  </grid>
  <slider>
    <id>slider1</id>
    <width>3</width>
    <startrect>1</startrect>
    <endrect>113</endrect>
  </slider>
  <button>
    <id>button1</id>
    <rects>
      <rect>0</rect>
      <rect>1</rect>
      <rect>2</rect>
      <rect>3</rect>
      <rect>4</rect>
    </rects>
</TouchGloveProfile>
```

4.4.5 Boundary Conditions

Startup The startup of the Raw Data Provider and the Profile Creator has to be done manually. The Raw Data Provider requires a command line argument, which tells the service the port the touchpad is connected to.
The Raw Data Interpreter service starts automatically, if a input data consumer is started.

Shutdown Shutdown is the same process as startup, only in reverse.

Error Conditions Error conditions in the communication process are handled in the DWARF middleware. Error conditions in the subsystems are handled in the subsystem itself.

4.4.6 Subsystem Functionalities

4.4.6.1 Raw Data Provider

The Raw Data Provider Service connects via serial port to the TouchGlove and waits for user input data. The touchpad sends packages with specific information like x,y, pressure and more. The type of the message packages depend on the mode the touchpad is running in. Basically, the touchpad can run in absolute or relative mode. In absolute mode, it sends absolute x and y coordinates. In the relative mode, it sends relative coordinates. Note: The interpreter can only handle absolute touchpad data.

4.4.6.2 Raw Data Interpreter

The Raw Data Interpreter interprets the incoming raw data messages. New messages are stored in a queue. Depending on the current message and the previous messages, the action is determined and the state machine changes its state. Afterwards, contingent upon the current state, the raw data is interpreted. The process is visualized in detail in figure 4.4.

The important parts of this process are:

- time limit: The time limit is the maximum time stamp difference between the current raw message and the raw message received before. This is used to determine whether the user has started a new task or not. If the time limited is exceeded, the interpreter deletes the raw message queue.
- button time limit: A button click can only be recognized by the isGesture flag, which is usually set by the touchpad hardware after the click has occurred. Because of allowing buttons superposed with sliders, the interpreter has to wait some time to check whether it was a button click or not. If the time delta between the first entry in the queue and the current message is greater than the button time limit and no isGesture occurred meanwhile, all data is interpreted as slider data. The problem with this approach is: Until the button time limit has exceeded, no slider event can be sent out. This means, that there is a delay of "button time limit", until slider events are sent.
- The Interpreter Elements handle element specific things like hit recognition.
- The Interpreter State machine is implemented as State Pattern. It holds the current state of the interpreter.

The State Machine The state machine is shown in figure 4.5.

The different states are described in the following:

- Unidentified motion: At the beginning, the state machine is in this state. This means that it is not clear, which motion occurs at the moment.
- Button clicked: If an isGesture is recognized, the state changes to "Button clicked". A button click event will be sent afterwards, if the position matches to a button in the profile. Afterwards, the state is automatically changed to "Ignore input data", because the event has sent and the rest of raw messages is not interesting anymore.

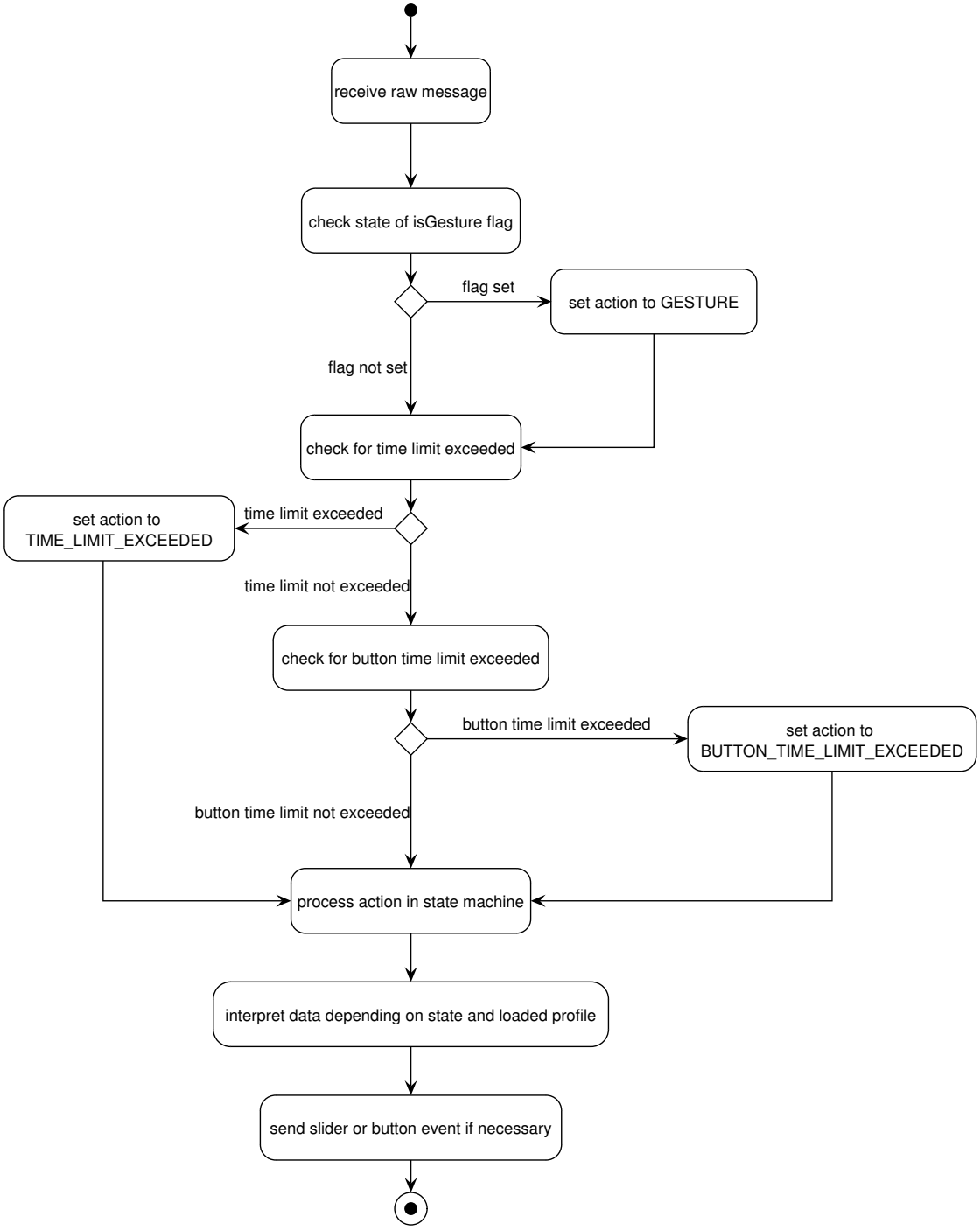


Figure 4.4: Raw Data Interpreter Activity Diagram

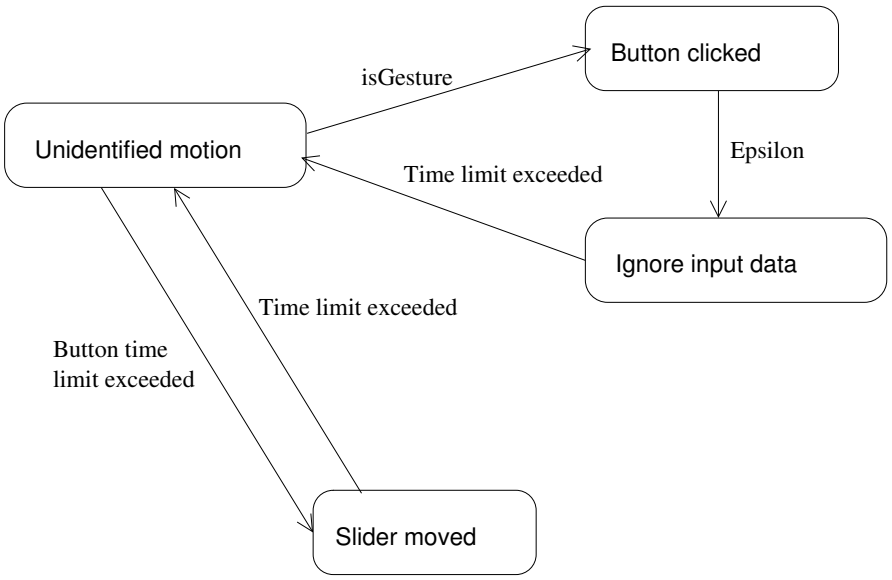


Figure 4.5: Interpreter State Machine

- Ignore input data: Nothing happens here. All incoming raw messages will be ignored.
- Slider moved: After the button time limit has exceeded, the state is changed to "Slider moved". This means, that all incoming raw messages are interpreted as slider moves.

4.4.6.3 Profile Creation

The Visual Profile Creation Tool is for creating user profiles which are needed in the Raw Data Interpreter Service. The user can visually compose the profile by adding interpreter elements like sliders and buttons. Besides the possibility to modify buttons and sliders with the mouse, it is possible to use the TouchGlove directly as input device.

4.5 Object Design

4.5.0.4 Raw Data Provider Subsystem

Architecture Figure 4.6 shows the architecture of the raw data provider service, which is highly oriented towards the previously introduced driver framework. But due to performance issues, all layers are implemented into one service.

- The RS-232 Linux Driver is developed for the communication with the serial port. It uses the POSIX interface of Linux. The class name is POSIXSerialComm.
- The RS-232 Protocol Nub is a general interface for communicating with serial ports. All specific implementations of RS-232 drivers have to implement this interface. The class name is RS232Nub.

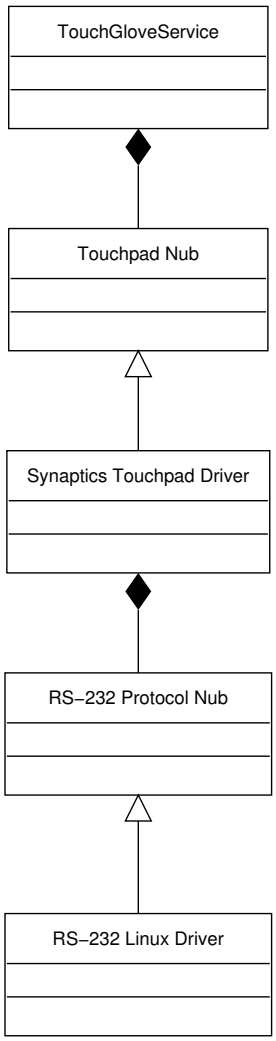


Figure 4.6: Driver layers of raw data provider service

- The Synaptics Touchpad Driver manages the touchpad communication with a Synaptics touchpad which is connected over a serial port. It uses the RS-232 Protocol Nub for the communication with the serial port. Unfortunately, Synaptics touchpads with different communication protocols (RS-232, PS/2, ...) use different commands for configuring. This is the reason, why the Touchpad Driver is specialized for Synaptics serial port touchpads. The class name is SynapticsSerialTouchpadComm.
- The Touchpad Nub is a general interface for communicating with touchpads over serial port. All touchpads connected over serial port have to implement this interface. The class name is SerialTouchpadNub.
- The TouchGloveService is a DWARF service that receives the touchpad data and propagates it into a DWARF system.

Interface For a detailed description, please refer to the Doxygen documentation.

Output Event Format The service sends regardless to the touchpad mode events in given format:

```
struct TGloveDataRaw {
    long x;
    long y;
    long pressure;
    long width;
    boolean isGesture;
    boolean isFinger;
    boolean isLeftButton;
    boolean isRightButton;
    boolean isUp;
    boolean isDown;
    Time timestamp;
  };
```

- x: the x coordinate. (either absolute or relative)
- y: the y coordinate. (either absolute or relative)
- pressure: The pressure with which the user presses his finger on the touchpad.

Z = 0	No finger contact
Z = 10	Finger hovering near pad surface.
Z = 30	Very light finger contact.
Z = 80	Normal finger contact.
Z = 110	Very heavy finger contact.
Z = 200	Finger lying flat on the surface.
Z = 255	Maximum reportable Z; whole palm flat on pad surface

- width: this value supplies extra information about the character of the contact with the touchpad. Please refer to [33] for details.
- isGesture: The touchpad is able to recognize tapping gestures. If such a tap is recognized, the flag is set to true. This flag is used to recognize button clicks in the Raw Data Interpreter Service.
- isFinger: The finger bit reports the state of the firmware's internal finger-presence check. True means, a finger is present.
- isLeftButton: It is possible to connect two physical buttons to the touchpad. This flag is true, if the left button is pressed.
- isRightButton: This flag is true, if the right button is pressed.
- isUp: Some touchpads provide two additional physical buttons called "Up" and "Down". This flag is set, if the up button is pressed.
- isDown: This flag is set, if the down button is set.

Note: In DWARF, this event description can be found in the TGloveDataRaw.idl.

4.5.0.5 Raw Data Interpreter Subsystem

Architecture The Raw Data Interpreter Service receives TouchGlove raw data from the Raw Data Provider Service and tries to recognize high-level events like button clicks or slider moves. The user can provide specific profiles which contain information about locations of different buttons and sliders he wants to use. In a profile, the whole touchpad is screened. This grid is numbered from 0 (top-left corner) to (xGridSize*yGridSize)-1 (down-right corner). For each interpreting action, the users hit on the TouchGlove is transformed into a "hit rectangle" in the grid.

The Raw Data Interpeter Service consists mainly of the classes shown in figure 4.7:

- The TouchGloveInterpreterService handles the communication over DWARF.
- The TouchGloveInterpreter interprets the incoming raw data messages. New messages are stored in a queue. Depending on the current message and the previous messages, the action is determined and the state machine changes its state. Afterwards, contingent upon the current state, the raw data is interpreted.
- The Interpreter Elements handle element specific things like hit recognition.
- The Interpreter State machine is implemented as State Pattern. It holds the current state of the interpreter.

Interfaces For a detailed description, please refer to the Doxygen documentation.

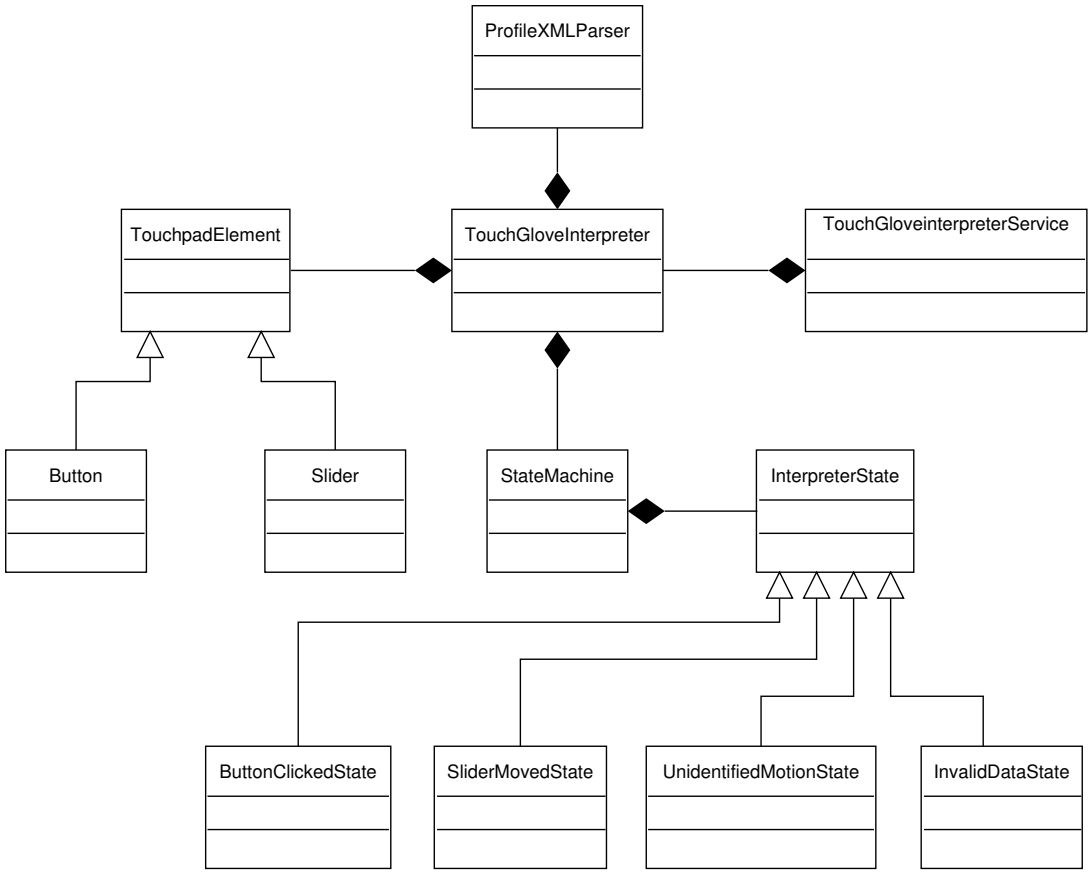


Figure 4.7: Raw Data Interpreter Class Diagram

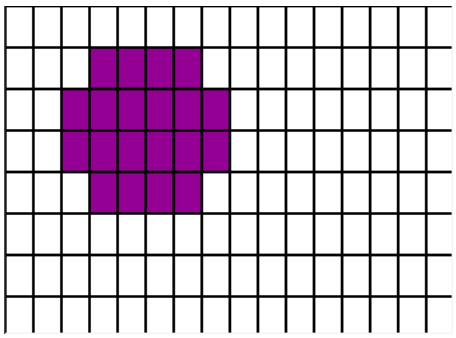


Figure 4.8: Button Example

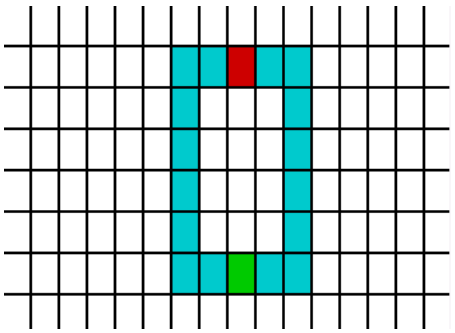


Figure 4.9: Slider Example

Interpreter Elements In the current implementation, two different interpreter elements are possible: Buttons and sliders. Interpreter elements of different types can be ordered overlapping.

Buttons Buttons are represented simply by a list of rectangles in the grid, which represent the buttons location and shape.

The hit recognition algorithm works simply by checking the coordinates against the profile grid. A problem here is: The coordinates in the raw message which has the isGesture flag set are usually 0/0, because the isGesture event is sent after the hit has occurred. To solve this problem, the coordinates from older raw messages, which are stored in the message queue, are used.

Sliders Sliders are represented by a start point, an end point, and a width. This means, that the start and end line are always orthogonal to the axis through the start and end point. The start and the end point are rectangles in the grid. This width is the width in rectangles.

The hit recognition algorithm calculates for each slider a rectangle with the given start point, end point and width. If the hit point is in this rectangle, the slider position is calculated by projecting the hit point orthogonally onto the line between start and end point.

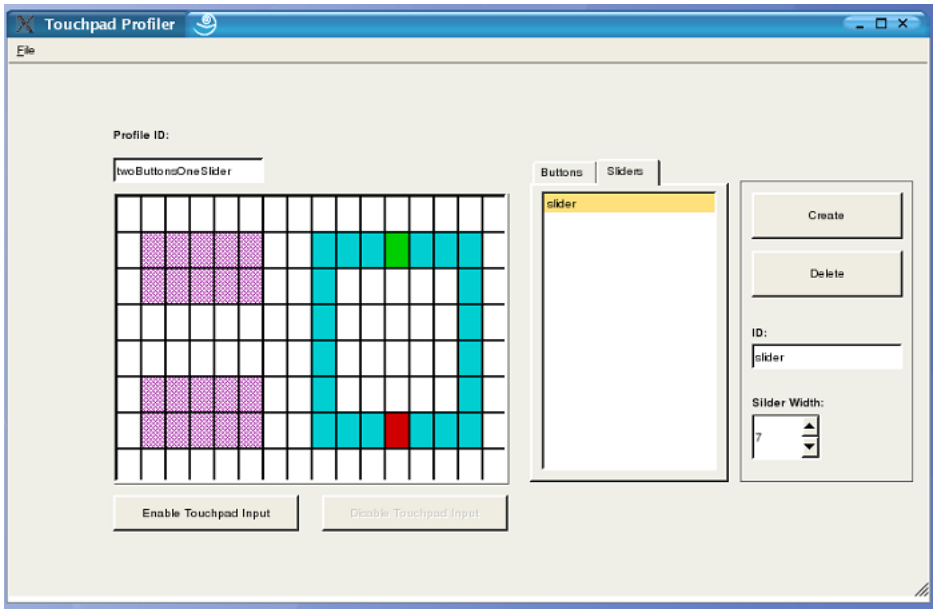


Figure 4.10: Screenshot of Profile Creation Tool

Afterwards, the position of this point on the line is calculated and normalized between 0 and 1.

4.5.0.6 Profile Creation Subsystem

The Profile Creation Service is implemented as a QT application and consists mainly of the classes shown in figure 4.11:

- The TouchGloveProfilerService class starts the QT application is responsible for communicating with the DWARF framework.
- The TouchGloveProfilerImpl class contains the functionality of the TouchGlove Profiler.
- For the visual creation of the profile, a special canvas as needed, which allows access to the mouse click coordinates. This canvas is implemented in the TGCanvasView class.
- the ProfileXMLParser implements parsing functionality for profile XMLs.

Interface For a detailed description, please refer to the Doxygen documentation.

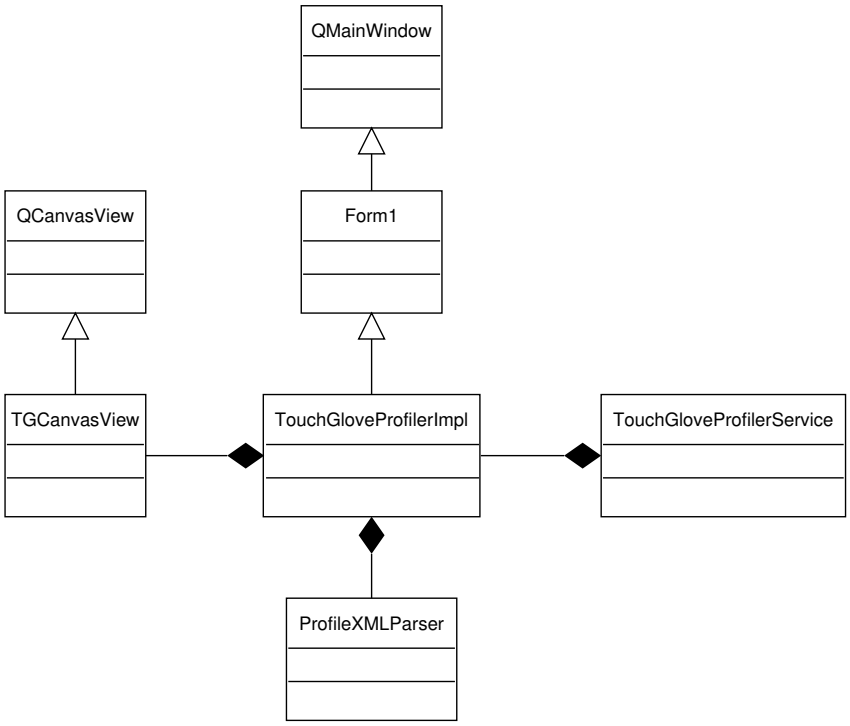


Figure 4.11: Class diagram for Profile Creation Tool

5 Conclusion

5.1 Results

The first chapter of this work was about a general driver framework for hardware drivers. It showed a solution, which is highly modularized and lightweight. This approach was proved by the development of the TouchGlove Raw Data Provider, which uses this driver framework. Due to performance reasons, it seems not to be a good idea to use a single DWARF service for each driver layer.

The second chapter discussed an approach for connecting input devices flexibly with the consumer. Although there has to be done lots of work in the second part of the matching process, the introduced ideas seems to be a good start for achieving maximum flexibility in the matching process. This concept was used in the ARCHIE scenario.

The third chapter is about the development of a driver for the TouchGlove input device. The subsystem decomposition proposed three components, the Raw Data Provider, the Raw Data Interpreter and the Profile Creator. The Raw Data Provider establishes a connection to the TouchGlove hardware over the serial port and sends the raw data using the DWARF framework to the Raw Data Interpreter and the Profile Creator. A problem in this component was the initialization of the TouchGlove over the serial port, because I wanted to connect the TouchGlove hardware to the iPAQ handheld of Compaq. Unfortunately, this handheld provides not a full serial port and the initialization was not possible.
The Raw Data Interpreter interprets the incoming raw data events and generates with the help of the currently loaded profile high-level events. A problem in this component is the distinction between a button click and a slider move, because I allowed the possibility to overlay buttons and sliders.
The Profile Creator is a visual tool for creating the profiles used in the Raw Data Interpreter. Besides using the mouse to create input elements, the TouchGlove can be used directly. The profiles are stored in an XML in the Configuration Service.

5.2 Lessons Learned

I learned, that augmented reality is an interesting research field, which requires a great portion of improvisation in developing new hardware and programming the software.
For understanding the DWARF framework deeply, a lot more time was needed than I expected before. The reason for this was mainly the complexity, which is necessary for such a flexible distributed framework.

Connecting input devices in a flexible way is more difficult than I expected initially. Many problems occurred in deep discussions with ARCHIE team members and our coaches.
Finally, working in the ARCHIE team on a common project was an excellent team work experience for me. It is a huge different to code a project for its own or to work in a group, which needs great communication efforts between the participants.

5.3 Future Work

TouchGlove hardware The usability of the TouchGlove hardware should be improved. One problem was the disturbing cable, because the device was intended to be used as a mobile input device. A solution would be a bluetooth adapter. Another problem was in the usability of the touchpad surface with the fingers. Firstly, only about half the area was reachable with the fingers. Secondly, it happens often, that the user touches the touchpad surface with other finger parts than the finger tip. This causes useless raw data. A solution would be a deeper embedding of the touchpad into the glove.

Profile Elements Allowing more profile elements like circular sliders and relative sliders would be another point for improvement. This would help to provide intuitive input possibilities for untrained users.
This improvement affects different parts of the TouchGlove services: Firstly, the Profile XML DTD has to be enhanced. Secondly, the Raw Data Interpreter needs major changes. And finally, the Profile Creator needs coding efforts to allow more input elements.

Profile Creator Unfortunately, the current implementation of the Profile Creator is a prototype. It is running, but not easy to enhance. A re-implementation, which handles the creation and design of input elements in a more generic way would be desired.

Input Device Matching Most work has to be done in the approach of connecting input devices flexibly. The semantic matching part is not developed right now, but is essential to provide a usable way in using input devices flexible. The goal is to find a solution, which reduces the user interaction in choosing an input device but does not limit the user in using input devices flexibly. Another important point is that the approach has to be easily implementable into the existing DWARF framework without major coding efforts in the DWARF middleware.

A Hardware Assembly

The TouchGlove had to be assembled at first. For this, I used a Synaptics TouchPad (TM41B-200, COMBO) [33] with a serial connector. For the glove, a common bicycle half glove was taken. The touchpad was mounted with a Velcro to the glove.
The frame around the touchpad was made out of aluminum by the ”Rechnerbetriebsgruppe” of the Technische Universität München.

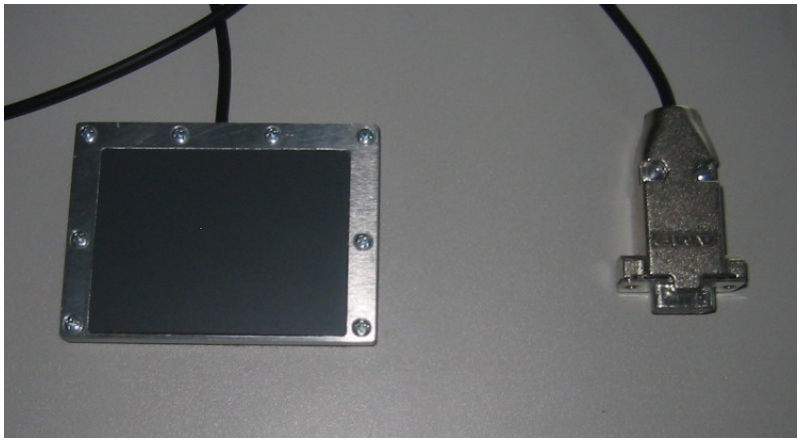


Figure A.1: Front side of the touchpad used for the TouchGlove

Figure A.2: Back side of the touchpad used for the TouchGlove

B Using the TouchGlove services

B.1 Startup

B.1.1 Raw Data Provider

The service name for starting the Raw Data Provider is called TouchGloveService. Only the parameter serialPort is required: In this parameter, the user has to specify the serial port the touchpad is connected with. ATTENTION: If the wrong port is given, strange things can happen. (e.g. if the mouse is connected to this port, the mouse will not work anymore)

B.1.2 Raw Data Interpreter

The Raw Data Interpreter is a so-called template service. It starts automatically, if there is a need for input data of a TouchGlove. The ConfigurationService of [34] has to be started before.

B.1.3 Profile Creator

The service name for starting the Profile Creator is TouchGloveProfilerService. No parameters are required.

B.2 Configuration

B.2.1 Raw Data Provider

There are several parameters which can be configured for the Raw Data Provider:

- command line parameter touchpadMode: This optional parameter is for specifying the touchpad mode.

MODE_00	Relative mode (1200 baud)
MODE_04	Relative mode with gestures disabled (1200 baud)
MODE_48	Relative mode with high packet rate (9600 baud)
MODE_88	Absolute mode (9600 baud, 6-byte packets)
MODE_8A	Absolute mode (9600 baud, 7-byte packets)
MODE_8B	Absolute mode (9600 baud, 8-byte packets with W)
MODE_C8	Absolute mode (9600 baud, high packet rate, 6-byte packets)
MODE_CA	Absolute mode (9600 baud, high packet rate, 7-byte packets)
MODE_CB	Absolute mode (9600 baud, high packet rate, 8-byte packets/W)

The default value is MODE_8B.

- Touchpad resolution: The resolution of the touchpad (min/max x/y coordinates) is stored in the four attributes RangeMinX, RangeMinY, RangeMaxX and RangeMaxY in the TouchGloveService.xml. Depending on the touchpad used, the attribute values have to be modified.

B.2.2 Raw Data Interpreter

The time limit and the button time limit is stored in the attributes timeLimit and buttonTimeLimit in the TouchGloveInterpreter.xml. It handles the distinction between button and slider events in the interpreter.

B.2.3 Profile Creator

No configuration is needed.

B.3 Creating Profiles

This is a short introduction about creating profiles visually with the Profile Creator:

B.3.1 Creating and Deleting Buttons and Sliders

Firstly, the type of the input element you want to create has to be selected. This is done by clicking on the corresponding tab (see figure B.1). After entering a unique ID into the ID field, you can press the create button.
Note: The IDs of the elements are later used for identifying the profile.
To delete an input element, simply highlight the corresponding list entry and press the Delete button.

B.3.2 Designing Element Layouts

At first, select the element you want to design from the list. You have now two possibilities to design the layout of your created elements: You can use the mouse or you can use directly the TouchGlove.

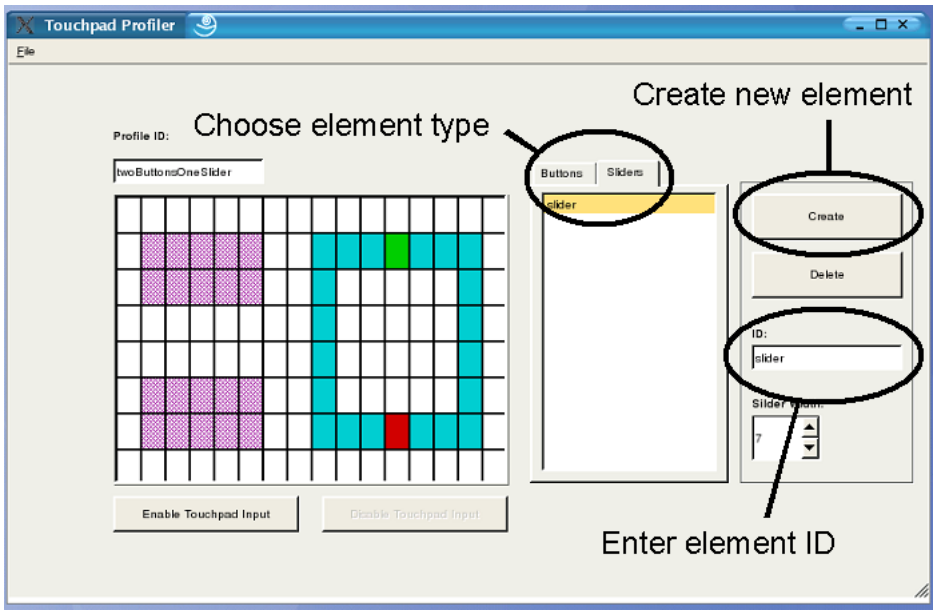


Figure B.1: Creating input elements in the Profile Creator

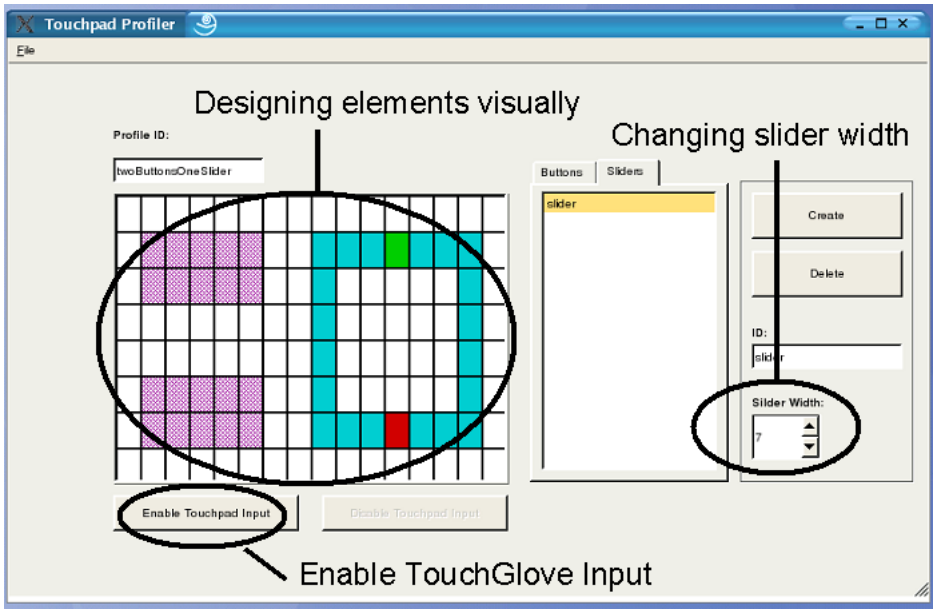


Figure B.2: Designing input elements in the Profile Creator

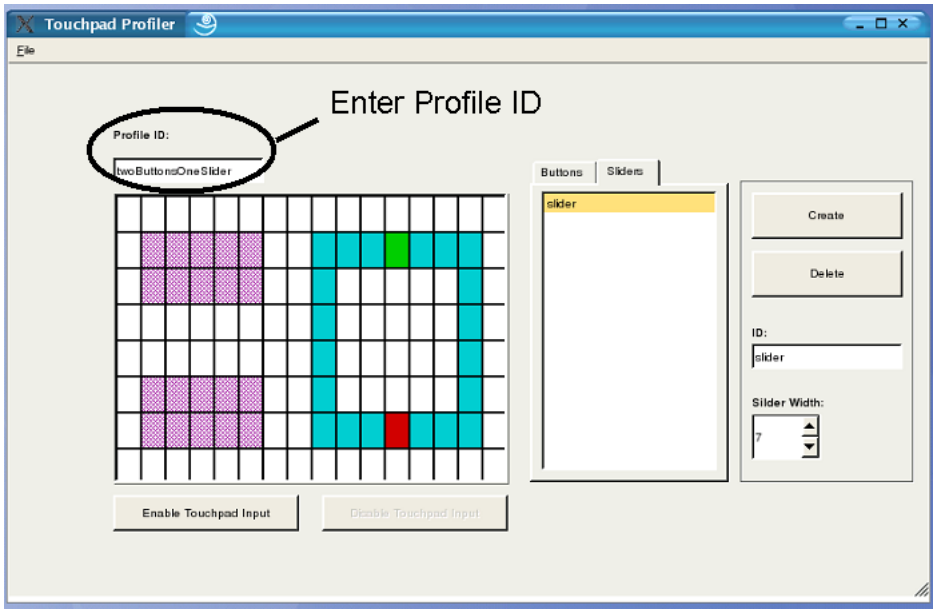


Figure B.3: Saving profiles in the Profile Creator

B.3.2.1 Using the mouse

Buttons For creating a button, simply click with the mouse on the rectangles which should be added to your button. With a second click on a selected rectangle, it will be deselected.

Sliders A slider is represented by a red dot, which is 0, and a green dot, which is 1. You can change the slider simply by dragging the red or the green dot around. Modification of the size of a slider is done in the input field on the right side marked in figure B.2.

B.3.2.2 Using the TouchGlove

Additionally, you can use the TouchGlove directly for designing the elements. Simply click on the ”Enable Touchpad Input” button to start.

Buttons By touching the TouchGlove, the corresponding rectangles are selected. Deselecting can only be done by using the mouse.

Sliders To position a slider, the first contact with the touchpad is the starting point of the slider. Now you can move your finger around on the touchpad to position the ending point of the slider. The last touched position is then the end point position.

B.3.3 Storing and Loading Profiles

After finishing the profile, it has to be stored in the Configuration Service and in the file system. Please enter a file name into the field marked in figure B.2. This name is only used for the file system storage. In the menu File, you can save and load profiles. You can only load the profiles from the file system.

B.4 Configuring TouchGlove Input Data Consumer

Each profile consists of several input elements like buttons and sliders with unique ids. Unfortunately, the service manager of DWARF does not allow multi value attributes. This causes a somehow dirty hack in the consumer XML description for selecting a profile. All ids of the required profile have to be concatenated to a string like the following way:
”-accept–create–delete–select-”
Accept, create, delete and select are the ids of the input elements. The ids have to be sorted lexically.
In the consumer XML, a need for InputDataBool and InputDataAnalogLimited has to be created. These needs require a predicate called configurationKey, which asks for the string described above. If only a subset of a profile is needed, wildcards can be used in this predicate as well. Here is an example:
”*-create-*-select-*”
This predicate would ask for all profiles, which contain at least two input elements called create and select.
In the events itself, the id of the input element is sent in the field event.header.fixed_header.event_name .

Bibliography

[1] *DWARF Project Homepage*. http://www.augmentedreality.de.

[2] *Request For Comments*. http://www.rfc.net.

[3] K. AHLERS, A. KRAMER, D. BREEN, P. CHEVALIER, C. CHRAMPTON, E. ROSE, M. TUCERYAN, R. WHITAKER, and D. GREER, *Distributed Augmented Reality for Collaborative Design Applications*, Eurographics '95 Proceedings, Maastricht, (1995).

[4] APPLE, *I/O Kit Fundamentals*, 2002.

[5] M. BAUER, *Distributed Wearable Augmented Reality Framework (DWARF) Design and Implementation of a Module for the Dynamic Combination of Different Position Tracker*, Master's thesis, Technische Universität München, 2001.

[6] M. BAUER, B. BRÜGGE, G. KLINKER, A. MACWILLIAMS, T. REICHER, S. RIß, C. SANDOR, and M. WAGNER, *Design of a Component-Based Augmented Reality Framework*, In IEEE and ACM International Symposium on Augmented Reality, (2001).

[7] G. BLASKO and S. FEINER, *A Menu Interface for Wearable Computing*, 6th International Symposium on Wearable Computers (ISWC 2002), (2002), pp. 164–165.

[8] B. BRÜGGE and A. H. DUTOIT, *Object-Oriented Software Engineering. Conquering Complex and Changing Systems*, Prentice Hall, Upper Saddle River, NJ, 2000.

[9] S. FEINER, B. MACINTYRE, and T. HÖLLERER, *Wearing It Out: First Steps Toward Mobile Augmented Reality Systems*, First International Symposium on Mixed Reality (ISMR 1999), (1999).

[10] M. FIORENTINO, R. DE AMICIS, G. MONNO, and A. STORK, *Spacedesign: A Mixed Reality Workspace for Aesthetic Industrial Desgin*, In Proceedings of the IEEE and ACM: ISMAR 2002, (2002).

[11] S. GRIBBLE, M. WELSH, J. VON BEHREN, E. BREWER, D. CULLER, N. BORISOV, S. CZERWINSKI, R. GUMMANDI, J. HILL, A. JOSEPH, R. KATZ, Z. MAO, S. ROSS, and B. ZHAO, *The Ninja Architecture for Robust Internet-Scale Systems and Services*, Computer Networks 35, (2001).

[12] C. HESS, M. ROMAN, and R. CAMPBELL, *Building Applications for Ubiquitous Computing Environments*, Pervasive 2002, (2002).

[13] O. HILLIGES, *Development of a 3D-View Component for DWARF based Applications*. Systementwicklungsprojekt, Technische Universität München, 2003.

[14] C. JUST, A. BIERBAUM, A. BAKER, and C. CRUZ-NEIRA, *VRJuggler: A Framework for Virtual Reality Development*.

[15] H. KATO, M. BILLINGHURST, and I. POUPYREV, *ARToolKit version 2.33 Manual*, 2000. Available for download at `http://www.hitl.washington.edu/research/shared_space/download/`.

[16] C. KULAS, *Usability Engineering for Ubiquitous Computing*, Master's thesis, Technische Universität München, 2003.

[17] M. KURZAK, *Architecture and Urban Planning*, Master's thesis, Universität Stuttgart, 2003.

[18] R. LANGENDIJK, *The TU-Delft research program "Ubiquitous Communications"*, 21st Symposium on Information Theory, (2000).

[19] F. LOEW, *Context-Aware Service Selection realized by the AR Toolkit*. Systementwicklungsprojekt, Technische Universität München, 2003.

[20] A. MACWILLIAMS, *Using Ad-Hoc Services for Mobile Augmented Reality Systems*, Master's thesis, Technische Universität München, 2001.

[21] A. MACWILLIAMS and T.REICHER, *Decentralized Coordination of Distributed Interdependent Services*, Technical Report, (2003).

[22] F. MICHAHELLES, *Designing an Architecture for Context-Aware Service Selection and Execution*, Master's thesis, Ludwig–Maximilians–Universität München, January 2001.

[23] A. OLWAL, *Unit - A Modular Framework for Interacton Technique Design, Development and Implementation*, PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2002.

[24] H. T. REGENBRECHT, M. T.WAGNER, and G. BARATOFF, *MagicMeeting - a Collaborative Tangible Augmented Reality System*, in Virtual Reality, vol. 6, Ulm, Germany, 2002, Springer.

[25] G. REITHMAYR and D. SCHMALSTIEG, *Open Tracker - An Open Software Architecture for Reconfigurable Tracking based on XML*, Technical Report, (2000).

[26] G. REITHMAYR and D. SCHMALSTIEG, *Mobile Collaborative Augmented Reality*, In Proceedings of the IEEE and ACM: ISAR 2001, (2001).

[27] S. RISS, *A XML based Task Flow Description Language for Augmented Reality Applications*, Master's thesis, Technische Universität München, 2000.

[28] M. ROMAN, C. HESS, R. CERQUEIRE, A. RANGANTHAN, R. CAMPBELL, and K. NAHRSTEDT, *A Middleware Infrastructure to Enable Active Spaces*, IEEE Pervasive Computing, (2002).

[29] C. SANDOR, *CUIML: A Language for the Generation of Multimodal Human-Computer Interfaces*, Master's thesis, Technische Universität München, 2000.

[30] D. SCHMALSTIEG, A. FUHRMANN, G.HESINA, Z. SZALAVARI, L. M. ENCARNACAO, M. GERVAUTZ, and W. PURGATHOFER, *The Studierstube Augmented Reality Project*, Technical Report, (2000).

[31] F. STRASSER, *Personalized Ubiquitous Computing with Handhelds in an Ad-Hoc Service Environment*. Systementwicklungsprojekt, Technische Universität München, 2003.

[32] STUDIERSTUBE, *OpenTracker: An Open Architecture for Reconfigurable Tracking based on XML*. http://www.studierstube.org/opentracker/overview.html.

[33] SYNAPTICS, *Synaptics TouchPad Interfacing Guide*, 2nd ed.

[34] M. TÖNNIS, *Data Management for AR Applications*, Master's thesis, Technische Universität München, 2003.

[35] A. TRIPATHI, *Augmented Reality Application for Architecture*, Master's thesis, University of Southern California, 2000.

[36] VARIOUS, *Proceedings of the IEEE International Workshop on Augmented Reality - IWAR 1999*, (1999).

[37] VARIOUS, *Proceedings of the IEEE and ACM International Symposium on Augmented Reality - ISAR 2000*, (2000).

[38] VARIOUS, *Proceedings of the IEEE and ACM International Symposium on Augmented Reality - ISAR 2001*, (2001).

[39] VARIOUS, *Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality - ISMAR 2002*, (2002).

[40] M. WAGNER, *Design, Prototypical Implementation and Testing of a Real-Time Optical Feature Tracker*, Master's thesis, Technische Universität München, 2000.

[41] A. WEBSTER, S. FEINER, B. MACINTYRE, W. MASSIE, and T. KRUEGER, *Augmented Reality in Architectural Construction*, 1996.

[42] M. WEISER, *The Computer for the 21st Century. Scientific American*, Scientific American, (1991), pp. 94–104.

[43] B. ZAUN, *A Bluetooth Communications Service for DWARF*. Systementwicklungsprojekt, Technische Universität München, 2000.

[44] B. ZAUN, *Calibration of Virtual Cameras for AR*, Master's thesis, Technische Universität München, 2003.