Master Thesis

Analysis, Design and Implementation of a Framework for Digital Desk Applications

Henning Kolenda
Matrikelnummer: 0611040

Hochschule Mannheim

Praktischer Teil angefertigt in der
e-Business Engineering Group
Universität von Calgary, Canada

Betreuer:

Prof. Dr. E. Körner, Hochschule Mannheim

Prof. Dr. F. Maurer, Universitaet von Calgary

Statutory Declaration

Ich versichere, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit hat in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Henning Kolenda Calgary, den 31.03.2007

German Abstract

Agile Software Entwicklung beruht in einem hohen Grad auf zwischenmenschlicher Kommunikation. Auch wenn Computer für diese Kommunikation Unterstützung bieten können, können sie diese auch behindern, wenn sich mehrere Personen einen Desktop Computer teilen muessen.
Mit der Entwicklung von AgilePlanner wurde die Beschränkungen, die durch vorhandene Software verursacht werden, aufgehoben.
Die Verwendung eines digitalen Tisches könnte die zwischenmenschliche Kommunikation auch in einer Umgebung unterstützen, in der Computer zur Zusammenarbeit genutzt werden.

Abstract

Agile software development highly relies on human to human communication. Although computers can offer support for this interaction, they can also hinder this interaction in terms of people sharing a single desktop computer. With the development of AgilePlanner the limitations caused by previous existing software are lifted. Using a digital table AgilePlanner could transfer the human to human interaction based planning meeting in a computer supported cooperative environment.

Acknowledgments

First I would like to thank Prof. Dr. Eckhardt Körner and Prof. Dr. Frank Maurer for being the supervisors of my thesis.

Additionally I want to thank Prof. Dr. Frank Maurer for the possibility to participate in the research of the group also beside my thesis.

Thanks to David Fox, Robert Morgan, and Patrick Wilson for reviewing my thesis and giving hints.

I finally want to thank my parents, which made my stay in Calgary possible and simple and supported me through my whole education.

Dedication

Bianca, Emily, Micha & Simon

und alle anderen, die ich während dieser Zeit vermissen musste

Table of Contents

1 Introduction

1.1 Working Collaboratively with Computers

Since computers have moved into offices and laboratories, people have come to rely on them. With increasing in technology, calculating, memory, and storage capacity and the connecting of the computers, computer often have became more than a supporting tool for the working environment. In other words, it is fast and easy to send documents through email, download them form the network or versioning them while having them in the computing environment. To resolve these benefits of the digital supported environment, the documents have to be edited on a computer. With moving the documents from the desk into the desktop computer, the focus also moved from the desk to the computer.

But desks are designed for single person use; while most desktop computers are equipped with one monitor, one mouse, and one keyboard, there is also a possibility that a second monitor can be added to enhance the working environment. With these constraints it is difficult to work collaboratively on computers. Using a projector in a conference room can solve the space problem of sitting closely together or the limitation of visibility to the screen.

Another problem is to interact with the computer. While designed for single user input usually only one keyboard and one mouse is attached to a computer. With the spreading of keyboards and mice connected to a computer via USB it is easy to connect multiple devices of each type. But due to the fact, that Microsoft Windows does not support the input of multiple devices individually while mapping the all input events to one input queue, additional support is needed.

If the situation is based on a document and that document is in the computing environment, the focus is then on the computer and the computing environment. For example in a software development environment, the focus of pair programmers concentrates on the code. But if the digital environment should be used to support processes based in human to human interaction, the limitation of standard desktop

environments with its constraint hinders communication. This communication may be based on processes of development or simply on brainstorming meetings.

Recently researchers in the area of HCI have proposed the use of a digital table as a possible solution. The table tops of digital tables have an embedded screen or a projector and an integrated touch sensitive system that allows the user to interact with the computer. This gives the users the possibility to sit or stand around a table primarily facing each other while talking rather than spending time mainly viewing the display. This also supports an exceeding use of non verbal communication [35]. Not facing a screen gives to opportunity to recognize other peoples behavior, gestures or body language which may be valuable in the discussion.

1.2 Agile Planning

Agile software development is one method of software development currently being used instead of other famous methods such as waterfall model, spiral model, and iterative software development methods. In the Agile Manifesto [1] Beck et al. defined the principles of agile software development as following:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

These principles are reflected in various methods used in agile software development such as Extreme Programming, Scrum, Crystal and many more.

One of the basics practices of many of those methods is the replacing of documentation by face to face communication. This requires trust not only in developers, but also in customers. By reducing formal definition of requirements it is essential for customers to trust the developers in their competency and willingness to solve the problem as well as for the developers that there is a tolerance from the customers in terms of their of mistakes.

Scrum defines the interaction of the development team with the customer ion terms of the scrum meeting. In the meeting the customers defines the required functionality

and development team estimates the effort of each task. Based on this estimates the customer can now decide the priority of each feature and therefore which features are to be included in this iteration. The remaining requested tasks are moved to a backlog for the next iterations. The documentation of each task, also called user story, is reduced to a short description and note of the estimated time required to complete the story on the index card. The remaining documentation of the user story is replaced by the verbal description that the customer has with the developers. [27]

But in many projects it is not possible to have a representative of the customer on site or the development team itself can not be collocated. This requires an audio link during the planning meeting. But still the created cards can not be accessed directly through all team members. To resolve this problem several web based planning tool have been created, unfortunately these tools often hinder the team because of their slow nature in terms of interaction and thus decreases the efficiency of the planning meetings [29] [30].

1.3 Goal of Thesis Work

As Morgan et al. mentioned in [30] the use of a digital table top could lift the limitation of using computer for agile planning meetings. The application AgilePlanner described by them can solve the problem of sharing the story cards as well as the interacting with the story cards while running AgilePlanner on a digital table for collocated as well as for distributed teams.

The goal of my thesis is to provide a framework for interacting with the digital table. Thereby, I will take into account the concurrently interacting with the table. The framework itself shall provide an interface independent of neither the hardware environment nor GUI framework to be a basis for use with as many applications as possible.

My implementation of the framework will focus on the support of pen or hand input for applications running on a table. Based on these inputs, I also want to provide recognition of gestures to control the application. To provide support for multiple mouse inputs, I will try to integrate the results of the parallel run diploma thesis by

Herbig [14]. However, on basis of the fact, that it was obvious early, that it is impossible to build a framework supporting concurrent input based directly on Microsoft Windows I will therefore not be following this aspect of his thesis.

As last step I am going to integrate the input framework in AgilePlanner. In doing so I want to support the most common actions with gestures based on the pen or hand inputs.

1.4 Structure of Thesis

This thesis is organized as followed.
Chapter 1 gives an overlook about the field of the work and defines the goals of the thesis.
In chapter 2 I give a look into the related work of interacting with a digital table and gestures on the table.
The environment in which the development was done is described in chapter 3. I also describe AgilePlanner which will be the application I exemplarily integrate the resulting framework.
In chapter 4 I describe the design and architecture of the Framework and compare it to current GUI frameworks.
The implementation of the individual parts of the framework I explain in chapter 5.
And in chapter 6 I present the integration of the in chapter 5 described parts in the AgilePlanner Application.
With the review of the goals and an outlook of future work in chapter 7 I close my thesis.

Names of classes or interfaces in the text and code snippets are highlighted using the font `Courier`. Diagrams are in UML reduced to the important information.

The source code to this thesis can be found on the CD attached to his document or in the SVN repository of AgilePlanner. The project is hosted on SourceForge[1] under the

[1] sourceforge.net

name “MASE: Agile Software Engineering”. The URL of the repository is https://mase.svn.sourceforge.net/svnroot/mase. AgilePlanner is in the folder “AgilePlanner_v2.0”. To browse the repository with your internet browser visit http://mase.svn.sourceforge.net/viewvc/mase/.

2 Related Work

In this chapter I want to look at the related work on digital tabletops. I structured the chapter as followed. First I survey the different input devices. Then I look at the possibility of an input device to differentiate individual users. Next I look at the input devices with a view of the support of simultaneously operating users. The design of the applications and special behavior of applications are the following topics I explore. In the chapter "Related Work" I close with a view of gestures provided for tabletop applications.

2.1 Possible Input Devices

2.1.1 Pen

To interact with a digital table there are two different kinds of pens used in digital supported environment.
First, there are active pens, which means they are an active part of the recognition of the user interaction. For example the pen used in [4] is a multi functionality pen. In vertical display environments its laser pointer can be used to as a pointing device for a computer. However, this functionality is useless for using the pen on horizontal touch sensitive systems but the pen with its wireless emitter still provides the rest of its functionality as left and right mouse click by pressing a button or detecting a contact of the top of the pen to the surface.
The second kind of pen is a passive pen, which is a simple tool used to point on the tabletop and not providing any additional functionality by itself. For example the DViT technology from Smart can not differentiate if one of the provided pen, a finger or something else is used to touch the surface [37].

Using a pen to interact with a touch sensitive surface is a very common practice. This practice may have been made more common based on the fact that the most widely used touch sensitive device is a pen; for example to access a palm or tablet pc. A pen also allows a more natural interaction with a computer. But this also has some drawbacks. Depending on the size of the table, it can be difficult to reach a point at the far end of the table. A user may have to walk from his current position to the other

end of the table, ask another user to execute the demanded action or lean over the table to complete his task. Leaning over the table may block the use of the table for other users [12]. Another problem which occurs especially on touch sensitive systems, based on simple contacts, is the accidentally touching of the surface which is also recognized as a touch and therefore interpreted as an interaction. This could be caused by putting an elbow on the table by leaning over the table or just by pointing on an item on the display [34].

2.1.2 Hand

The direct use of a hand to interact with a digital table is very close to the use of a pen. In a general view, both have the same advantages and disadvantages; for example, difficulties to reach a point on the far end on large tables. But there are also some differences, as follows.

The main advantage of touch sensitive system accepting a hand as input device instead of a pen is that no additional device is needed. There are also some disadvantages using fingers to interact with the table. Some people prefer not to touch the surface which other people had previously touched and preferred to use a pen instead. Another issue is that people hesitated to interact simultaneously with the table. This is caused by the concerns of people to interfere with other people arms or accidentally hitting their hands. Thereby this appears more often in groups of people that don't know each other. Also cultural backgrounds played a big role ion the interaction. Especially Japanese people hesitated to interact simultaneously because of their cultural understanding of politeness.
But not only social factors play a role in handling a digital tabletop, also the conditioning of computer users to single user systems. This is also shown in the fact that many, especially adult users and user new with this technology, interact with only one finger.
Another issue is the size of the widget of applications built using standard GUI frameworks. The design of those applications is dimensioned for desktop environments. But with the larger size of a finger compared to the size of the mouse cursor and the dimension of the desktop widgets, there may be reduced proximity issues in terms of interaction interact [34]. In other words, a person's finger may be

too large to select a single small widget without interfering with another widget in close proximity. This issue could be resolved by using a pen or a GUI designed for tabletop use.

2.1.3 Mouse

While the most research about interacting with digital tabletops is in pen or hand input, the use of mouse in a digital tabletop environment has been researched very little. The uPen [4] based on a laser pointer, cannot be used as a mouse replacement in horizontal display environments, because a person leaned over the table would interfere the recognition of the laser pointer.
Although investigations have found that people prefer pen or hand input as a more natural way to interact with a digital tabletop, there are also some advantages in using a mouse. With a mouse, people have the ability to access all points of the display on the tabletop with small movements with their hand. Using a mouse also reduces the risks of physical collisions with other users [12].

2.1.4 Keyboard

Besides the general research in SDG frameworks, which was aimed at access to a single user computers with multiple users with each having access to their own input device, no digital table specific research has been done. This is caused by the fact, that the digital tabletops environment is more suited for organizational or annotation task than it is for text-entry based tasks [34]. SDG frameworks are discussed in chapter 2.4.

2.1.5 Alouds

To interact with a computer in a digital tabletop environment it is found useful to have the ability of verbal commands. Thereby studies [5] [38] have found Bluetooth headsets as a good solution for the input hardware, they give the users the possibility to walk around [38] and still have a good robustness [5]. Both implementations are based on a limited set of commands to recognize.
One of the major benefits of alouds is awareness of the command to the other members of the group. In contrast to actions based on gestures with a pen or a mouse, verbal commands are recognized by all members in the group and give them the

possibility to react to these commands immediately. It was also found out that alouds are more appropriate for abstract actions while gestures by pen, hand or mouse are for located actions. A notable fact is also, that the combination of alouds and gestures was found very useful. Thereby the gesture can refine the verbal command by adding location information [39] [40] [41]. While using only verbal commands it was found that there was a dislike by the study participants; the combination of alouds and gestures was preferred to only gestures by most of the participants. This is also shown in a higher efficiency and less errors [41].

2.1.6 Other Computers

Additional computers are used in various ways in digital tabletops environments. In [41] additional computers are used to lift the limitations of the used software recognition. Because only one instance can run on one computer multiple PCs are connected and send notifications with the recognized commands to the computer running the actual application.
Two laptops attached to a digital tabletop are used as private spaces in the study [10]. Thereby the users have the ability to drag and drop documents from the table to their private space on the laptop.
Another interesting scenario is the use of palm with pen used for text input for digital tables. This is described in [34] and already implemented in AgilePlanner [30]. Here a reduced version of AgilePlanner runs on tablet or pocket PCs to create or edit story cards.

2.2 Determination of the User

To make the reaction of an application depending on the users, it is necessary to have the possibility to determine which user interacted with the computer. Different approaches have been found to solve this problem.

One solution depends on the input devices. Some input devices provide an individual id for each device. The individual id then can then be mapped statically or dynamically to a particular user. The uPen identifies a user by emitting a different radio frequency for each user [4]. In the study [12] the used styli also provide an id. Provider of multiple mouse input based on the input of the USB port, such as

ManyMouse [11] , can provide the id of the device in the USB tree as an individual id. But not only the association of one user with an id of an input device can provide an ability to identify the active user there is another solution, DiamondTouch. With the DiamondTouch tabletop, developed by Mitsubishi Electric Research Laboratories[1], it is also possible to identify up to four users by the touching the surface with their individual fingers or hands. This technique is used, for example, in the papers [34] [40] and [47]. The recognition is based on a small current going through the body of each of the users and measured by a sensor in the chair [7].

Another solution is described by Mohamed et al. [28]. In this solution, the recognition of users is based on the gestures made with the input device. The software analyses the gestures depending on several features such as the first or the last point of trace and shape. This allows the recognition of user independent of the input device and makes it also available for the hardware which can not determine different users.

2.3 Simultaneously Operating Users

Another important property of hardware is the possibility of simultaneously interacting users. This is important to allow a fluid interaction and avoids hindering the human to human interacting. If simultaneously operating users is not supported by hardware, the interaction relies on social protocols, which forces the users to wait with until it is their turn to interact with the digital tabletop [32].

The DiamondTouch table top supports up to four concurrently with their finger interacting users [7]. Simultaneously interacting with a pen is supported by uPen [4]. There are also several providers for mouse input, for example ManyMouse [11] which supports not only identification, but also supports concurrent input.
The input devices mentioned above supporting concurrent input of users are also providing individual ids for the input devices or identify the user directly. But simultaneous input is not only supported by those kind hardware devices. The Smart DViT table is able to provide the position of two objects on the table [37]. This could

[1] http://www.merl.com/

be two fingers of one person, but also one finger each of two users. But as mentioned in the previous chapter “Determination of the User” requires this kind of hardware an additional solution to differentiate between the users.

2.4 Design of Applications

In the two previous chapters I described different kind of hardware in terms of the ability to determine users and possibility to interact simultaneously with the input device. Now I want describe the different ways that applications can interact with this multi input event. The term Single Display Groupware describes applications which support interaction of multiple users interacting with the application each with their own input device [16].
In all applications supporting simultaneously input of multiple users, the application has to be considered, that actions of one user can destroy work of other users. Problems can be caused by orienting documents to the own position of a user which can result in trashing the arrangement of the document of another user [32].

The Transparent Input Device Layer described in [5] and [16] is a Mixed Present Groupware. This means that the framework not only supports multiple inputs for one computer, but also supports interacting with multiple input devices in a distributed environment. In the context of my thesis, I focus on the aspect of the multi user support only in a local manner.
The Transparent Input Device Layer is based on a reimplementation of the Java AWT framework. The events are still sent to the listeners defined in the AWT API. But instead of sending instances of the AWT event classes, the Transparent Input Device Layer passes subclasses of the event classes to the listeners. These derived events provide additional information about the input device, where they were generated. Using the standard listeners to send events makes it possible to use the Transparent Input Device Layer in legacy applications without the requirement of changing code. For an application which should reacts individual to the input events of different devices, the developer simply has to use the subclasses instead of the standard AWT event classes. But this realization is limited to simple actions as clicking buttons and does not support actions as drag and drop [16].

The SDGToolkit is a Single Display Groupware realized in the Interactions Lab of the University of Calgary for Microsoft .NET. Applications using the SDGToolkit to provide input for multiple users use widgets provided by the framework. Those widgets are subclasses of the widgets of the .NET framework itself and provide additional events to implement the multi user support [19]. Several applications based on the SDGToolkit show the potential and the intuitive use of the framework [42].

2.5 Behavior of Applications

With the new digital environment it makes it also reasonable to think about different behavior of applications running on this new digital environment.
Users typically arrange items on desktop computers. With the use of a digital table and the analogy of conventional desks, this becomes more important. In the paper, [18], it is talked about algorithms to calculate the relationship of objects depending on the distance and positions to each other. Thereby, the above algorithms try to recognize structures in the arrangement of objects. This could be interesting for an application such as AgilePlanner, where such algorithms automatically calculate dependencies and priorities of story cards.
Another adoption from real desks to virtual desktops is BumpTop described in [1]. The application is actually written for the use on a tablet pc, but shows interesting features. BumpTop simulates a 3D desktop, where the user has the possibility to organize his documents in piles. The structuring of the documents is supported by gestures. Another interesting feature is the support of physics-based behavior. In BumpTop it is possible to use the pen to virtually throw documents from one end of the virtual desktop to the other end. Documents in the way of the "flying" document get pushed away as on a real desk.
In Flatland[17] [33], an application for digital whiteboards gives the user a set of different behaviors, such as map drawing or writing a to-do list, to make it easier to interact with the whiteboard while supporting the user with behavior dependent functionality. The to-do list, for example, supports the reordering of items in the list. The encouragement of lightweight interaction with the whiteboard in Flatland is not only focused on the input, but also in the style of the output. The style of the output is adapted to the handwriting strokes of the input. For example the calculator behavior prints the results similar to the input handwriting and not in a font of the system.

An important need of digital table top applications is the possibility to move and orient objects on the table [13]. This is required to make it easier to read documents also other sides of the table, by not have to read them up-side-down. Also documents on the far end of the table are difficult to read, because of the dimensions of large table tops.

2.6 Gestures

The chapter "Related Work" of my thesis I want to close with a look a gestures provided for digital tables.

With a digital table top supporting the interaction between users, gestures provide an easy way to release complex actions. The studies on gestures on table tops thereby focus on the attempt to provide natural gestures which can be used intuitively by the group members.

In [40] Tse et al. describe the implementation of a wrapper for standard desktop games to be used in a digital tabletop environment. Provided gestures are, for example, the use of a fist to place objects or marking a region in bordering it with two hands. Another gesture is the recognition of a grabbing, moving and releasing gesture as a cut and paste command. For all this gestures a multi touch sensitive hardware is required. In [40] as well as in [41] Tse et al. use a simple pointing gestures to refine verbal commands as already described in 2.1.5.
They also described a wrapper for Google Earth[1] in [41]. For this map based application the gestures focus on navigation, such as moving the map or zooming. Spreading fingers apart alternatively together zooms in or out on the map.
Other intuitive gestures are described in [47]. A wiping hand gesture erases writing text on the table. Or two hands pulled together cause the application to pile the documents under the hands.

[1] http://earth.google.com/

In [47] Wu et al. also talk about the reuse of gestures in a different context. Concretely, they talk about the reuse of primitive gestures, which allows a larger set of commands by the same amount of gestures. This makes it easier for the users, which only have to memorize a smaller number of gestures. But it makes it also simpler to write the gesture recognition, because it has to recognize a smaller number of gestures.

3 Environment of Implementation

After I took a look at the related work of working with digital tables, I describe in this chapter the environment in which I did the development. Thereby, I refine the available hardware and the software which was specific for me to use, each with their technical details, potentials and limitations.

3.1 Smart DViT Table

The digital table of the e-business engineering group is a based on a 6 feet long 4 feet wide and 4 feet high (183 cm x 122 cm x 122 cm) stand. With those measurements it is easily possible for up to 6 people to stand around and interact with the table. Each side has a 1 foot wide border to place notes, palm or other office tools as well as keyboard or mouse [30]. Within these borders are eight screens (two rows of each four screens) inserted. Each screen has a maximum resolution of 1280 x 1024 pixels [44]. Compared with commonly used projectors, which have an output resolution of about one screen, this large resolution allows to display significantly more objects. The screens are controlled by two Matrox QID Pro graphics cards in a standard desktop PC running Microsoft Windows XP placed below the table. To access the PC with conventional input devices such as mice and keyboards, they can be simply attached on the USB ports.

Figure 1: Picture of the digital table in the e-Business Engineering group

In addition to those common input devices, the digital table provides a touch based input functionality, specifically the DViT technology provided by Smart Technology. Contrary to others implementations of touch based systems, the DViT technology doesn't need a special sensitive surface or special pointing device. With one camera in each corner, it can detect the positions of up to two objects on the table at the same time; for example a finger or one of the four pens or two erasers inserted to the table.

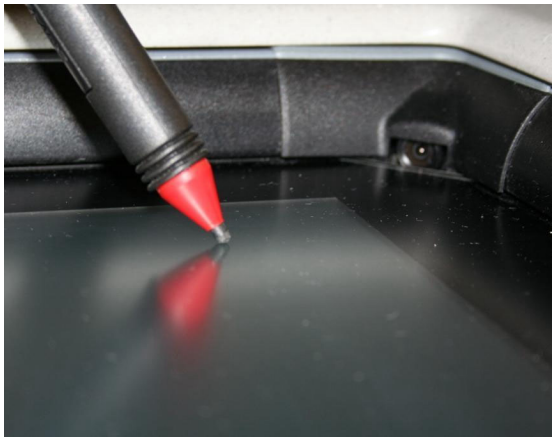


Figure 2: Picture of a pen on table in front of one of four cameras

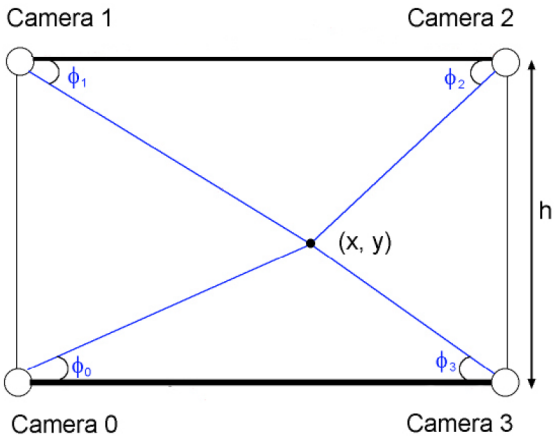


Figure 3: Camera identification of contact point; Source[37]

The driver of the table handles touch events in two ways, depending if a pen is removed or not. If no pen or eraser has been removed from their holding slot in the table boarder, mouse move events for the current position of the object touching the surface are created and sent to Windows.
In addition to the mouse events, the Smart software in this mode can detect two simple gestures based on how fingers are put on the table

- if the second finger is put right of the first finger, a click with the right mouse button is simulated
- if two fingers are set simultaneously side by side, a click with the scroll wheel (middle button) is simulated [36]

If a pen or an eraser is removed from their holding place, no mouse events are created. Instead pen specific events are sent either to the standard application or if one application registered for the events to that one. Whereby the standard application

draws in the color of the current pen on top of the window, where the object has touched the surface.

Smart Technology provides an API for Microsoft's .NET framework as well as for AWT and SWT in Java. Unfortunately Smart provides extremely limited documentation of their API, so I needed a lot of trials and errors to find out how to interact properly with the table framework

```
SWTSBSDK swtsbsdk = new SWTSBSDK();
swtsbsdk.attach(control, false);
swtsbsdk.registerControl(control, sbsdklistener);
swtsbsdk.startToolCacheSending();
```

1. Create an instance of SWTSBSDK
2. Attach the control for which the events should be received
3. Register a listener to receive the events
4. Setting that events are sent out immediately, without caching them

Especially the fourth line of code caused some difficulties to find out, because events are already sent without calling this function. But the events are seemingly cached and not sent out continuously, without calling this function.

While using the API I observed that events triggered from the first object that touches the surfaces is "1". A second object recognized by the DViT technology has the id "257" which stays as long as the object is recognized, even though the first object is removed.

3.2 ManyMouse

ManyMouse is a library to access multiple mice attached to a PC. Even though ManyMouse can in many cases provide data for serial mice as well as touch pads in laptop, for this project good support of USB mice is important. ManyMouse is available for different platforms and under the zlib[1] license [1].
ManyMouse was chosen among other frameworks because of the previous experience from another project [14] in the group.
I now wish to give a short overview of how ManyMouse works with Java under Windows XP. ManyMouse uses the User32.dll to get the entry points of the system, reads the data and fills up a queue. To read from this queue provided in C from an application written in Java, you have to use JNI.

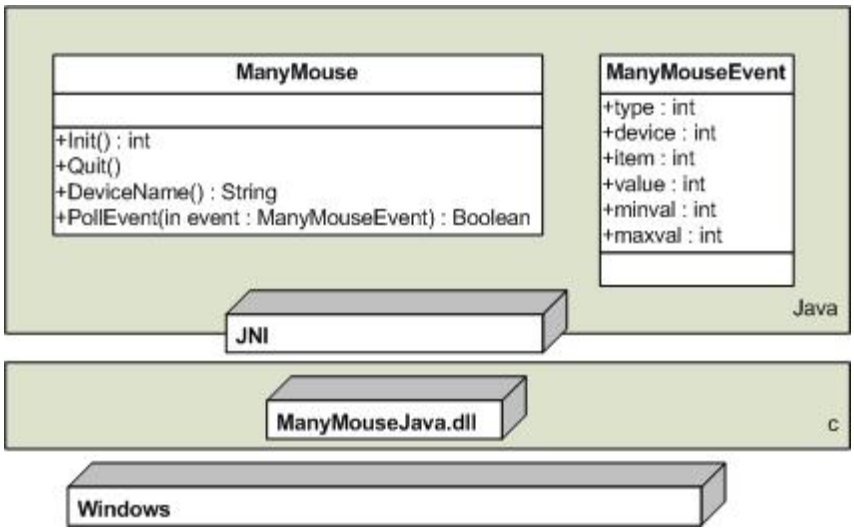


Figure 4: Architecture of ManyMouse used in a Java Application running in Windows XP

The ManyMouse library is able to provide events for moving, pressing buttons and using a scroll wheel. The movements of a mouse are fragmented into changes in horizontal and vertical changes.

[1] The zlib license is an accepted open source license by the Open Source Initiative. To view the license text of the project visit http://svn.icculus.org/manymouse/trunk/LICENSE?rev=3

3.3 AgilePlanner

AgilePlanner is an application developed in the e-Business Engineering Group of the University of Calgary to support teams in agile planning meetings. It bases on the researches of Morgan et al [29][30] who extended the basic results of Liu [21][22][23] by a distributed aspect.

In contrary to web based planning tools as [6][43][48] which are designed for vertical displays in a single user environment and on visual aspect focused applications as [46] AgilePlanner tries to use the advantages of table based environments in collocated and distributed environments[30].

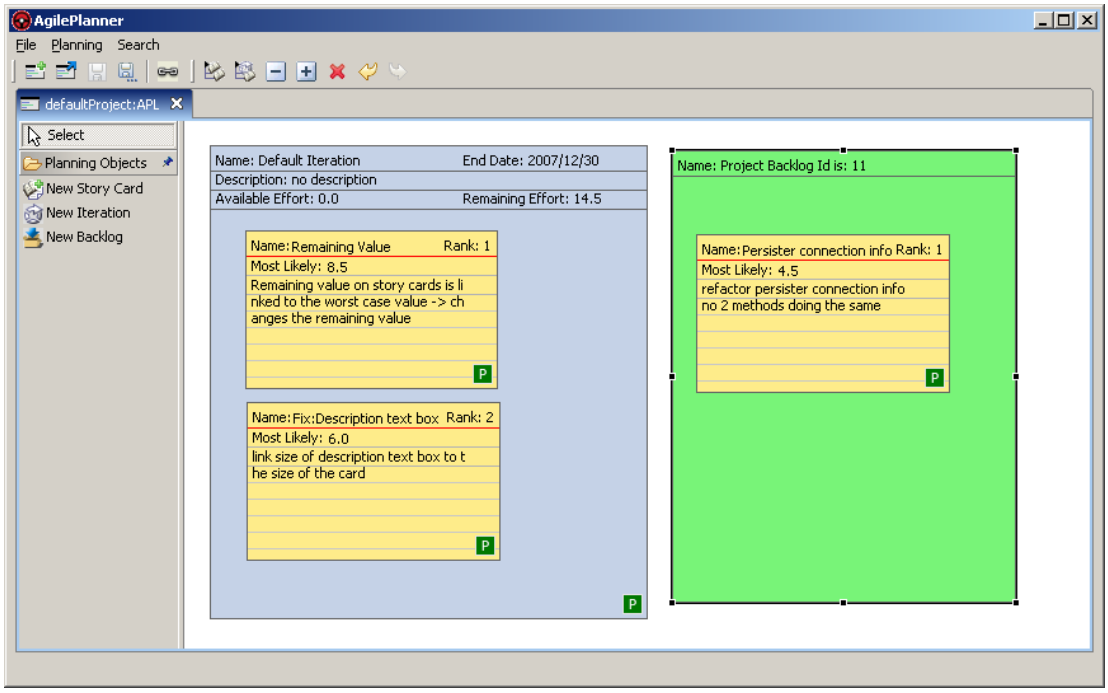


Figure 5: Screenshot of AgilePlanner with Backlog, Iteration and Story cards

AgilePlanner supports planning meetings in different ways. During the planning meeting it gives the possibility to:

- create, edit or delete story cards, iteration or backlog
- move story cards from one iteration to another or to the backlog or vice versa
- automatic calculation of effort and remaining effort for one iteration
- dynamic calculation of the rank of a user story by the location of the story cards
- and many more

With the support for distributed teams multiple instances of AgilePlanner can be connected together and each part of the distributed team can see the results of the members in other locations in real-time. But not only changes of the view are transferred, other clients also see a representations of mouse cursors of each other's client, so those can used to show highlight parts of the view and support the communication.

AgilePlanner is realized as an Eclipse plug-in using the GEF. GEF is a framework that allows developers to easily create a rich graphical editor for existing application models. As GEF is application neutral it can be used to build almost any application. The homepage of the framework provides examples for a logic diagram editor or an editor to layout activity diagrams [8].

GEF realizes the model-view-controller architecture. The model part is the data model of the application. The viewer and controller are realized in GEF, each separated in an own package. The `draw2d` package is responsible for all visual aspects of the framework. The classes in `draw2d` are used draw various kind of figures, cursor an tooltip support, printing, and many more. The controller is a subclass of `EditPart` where listeners to the model can be added [8] [45].

4 Architecture and Design of the Input Framework

In the previous chapters I gave an overview of the related work and the environment of my thesis work. Now I want to present the architecture and design of the input framework in this chapter.

4.1 Overview

While targeting the goal of my thesis, which was providing input and gesture support for digital tables, I wanted to make sure that as much functionality as possible is also available for desktop systems. I don't see the input framework as a replacement for the standard single inputs provided by all frameworks, but a set of input providers which can be used to extend those frameworks and provide accessibility to beyond this current standard.

My first goal was not to create a framework based on any GUI framework such as AWT[1], Swing[2] or SWT[3], which would reduce the reusability by limiting the application to the selected GUI framework. I also want to avoid introducing dependencies between the different input devices. To emphasize this, each files of a provider are placed in an own package.

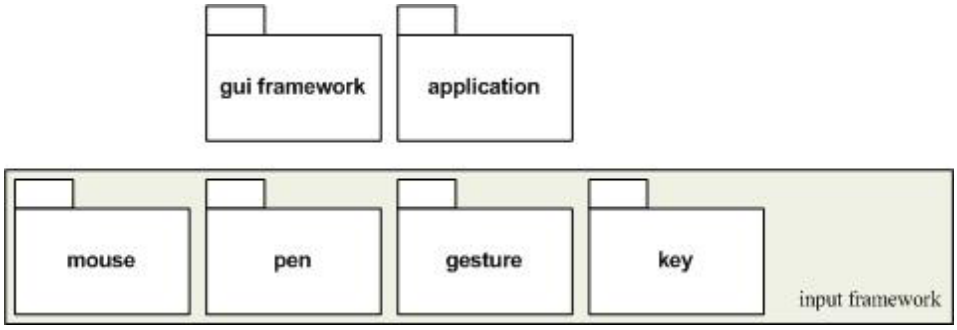


Figure 6: Package diagram of input framework

[1] Documentation and JavaDoc of version 5 under
http://java.sun.com/j2se/1.5.0/docs/guide/awt/index.html

[2] Documentation and JavaDoc of version 5 under
http://java.sun.com/j2se/1.5.0/docs/guide/swing/index.html

[3] Documentation and JavaDoc under http://www.eclipse.org/swt/

To support different environments and needs of application I first defined the interfaces which connect the application with the framework[1]. By using this abstraction it is possible to port an application from one environment to another by just replacing the implementation of the interfaces. Using a different digital table simply results in the need for writing a suitable implementation of the input framework. This allows applications to reduce their dependence on the input framework to achieve those needs. Due of the large amount of applications which provide a large variety of functions, there is also a need support for a variety of input devices. Desktop applications which are to be ported to a digital table environment probably just want to add the table input without the need of rewriting the rest of the user interaction code. In the case of an IDE supporting pair programming, there is a demand for support of multiple keyboard, so the developers can change the code simultaneously.

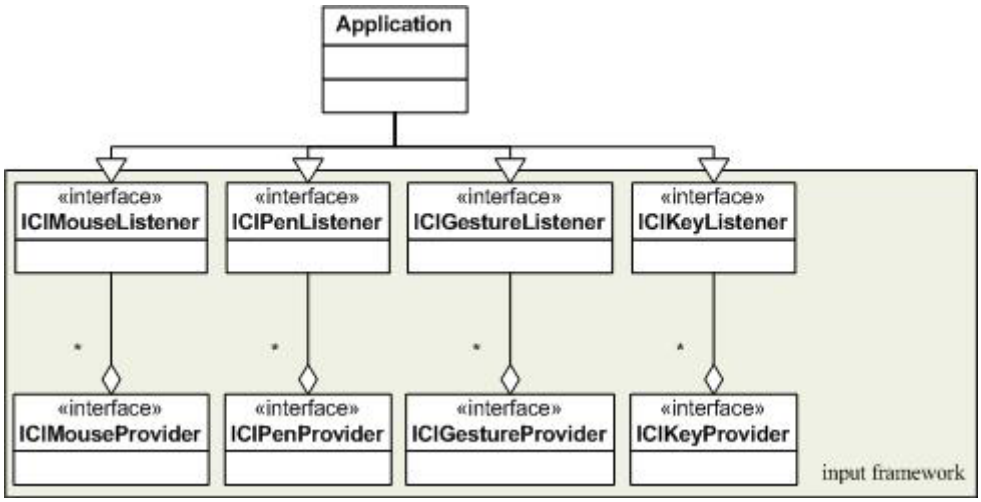


Figure 7: Class diagram of connection between application and framework

The details of the individual interface are described in the following chapters.

There are two general interfaces used for each input device:

[1] An earlier version of the interfaces to access multiple mouse input defined by me were already used in [14] to achieve a usability of the resulting implementation

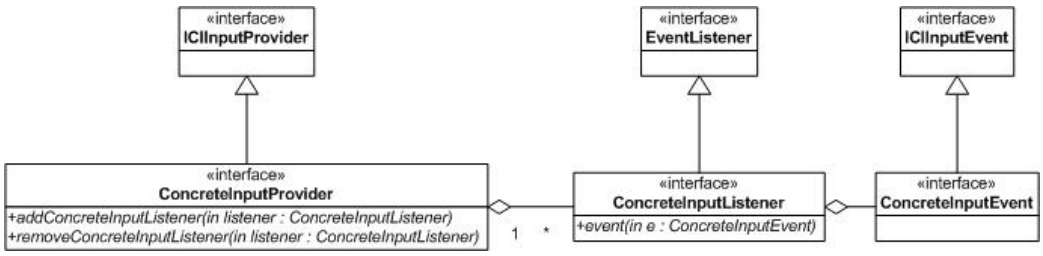


Figure 8: Use of ICIInputProvider and ICIInputEvent

ICIInputProvider

By having different implementation of input providers, each application will have its own hard coded function creating and setting up the implementations of the interfaces or will get this information by reading from a file. To reduce the individual code and give the class the ability to clean up, the function `stop()` is defined in the interface `ICIInputProvider` and all interfaces for the input provider are extensions of this. I considered introducing functions to start, pause and resuming events, however I did not find an advantage compared to just not to the sent events in the applications, because I observe no performance problems.

ICIInputEvent

Due to the fact that the use of the digital table should support human interaction, it is also useful to have the possibility to detect which user is interacting. Because I decided to keep the input framework close to the input devices the interface `ICIInputEvent` defines a getter `getID()` which represents a unique id for input devices. With this id the application has the possibility to link this id to a fixed user or dynamically depending on the application.

Positions

The advantage of being able to use a computer from more than one orientation causes difficulties when interacting with the computer. As shown in the figure below effects a different position a different view of the screen and complicates reading text. With knowing the position of a user, the application has the potential to arrange the output according to his position; for example to rotate a text so he doesn't have to read upside down.

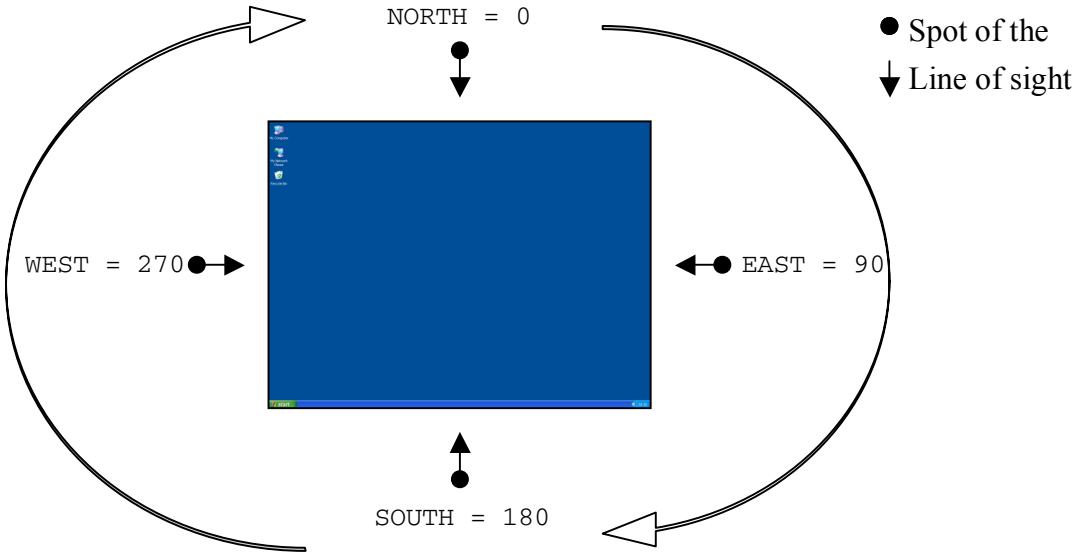


Figure 9: Positions around a table

Output is not the only issue that needed to be handled differently. Some input devices also needed to be handled differently. So when users wants to make the same gesture independent if there are located at the bottom, at the right side, or somewhere else around the digital table. They should not have to translate the gesture depending on their current position to match the computers orientation. Even though the shape of the table is not limited to a rectangle, standard computer display devices as screens or projectors are rectangles. For this reason I defined constants in the class `Positions` for each side of the table. To not limit users to those positions and allow a generic way the values of the constants are angles. The angle represents how many degrees somebody had to turn clockwise, if he would stand in the middle of the table beginning with facing to the top part of the screen. This allows it to use the input framework also on desktop computers while using the position `SOUTH` for the input devices.

ICIPoint

To have a handy way to represent a point, the framework has its own definition of a point. Compared to AWT or SWT, the definition only provides elementary functions. However, the use of one class out of those frameworks would cause a dependency and

limit the portability. I decided to define an interface, so that implementations could use the point classes of their frameworks and only have to write a wrapper around it.

Dynamic Behavior

I refrained from defining interfaces for a dynamic aspect, because I could not find examples of use that legitimize the introduction of additional interfaces or functions. This would lead to an increase in the complexity of the input framework.

4.2 Mouse Provider

Although I tried to keep as close as possible to the common GUI frameworks, such as AWT, Swing or SWT, to keep it easy for developers to get familiar with the input framework. I decided not to take over all defined interfaces.

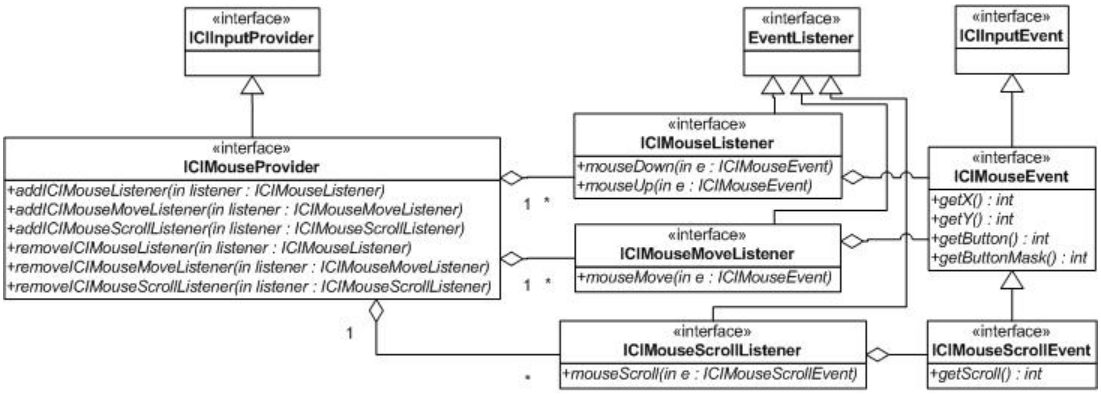


Figure 10: Class diagram for mouse provider

ICIMouseProvider

This interface defines the functions to register the listeners to receive events from the mice. It defines one add and one remove function for each type of listener and extends the `ICIInputProvider` interface.

ICIMouseListener

This listener defines the events that can be received for the buttons of a mouse. On the basis of this interface it is easy to see, that all input events are close to the hardware. Compared to the `MouseListener` interface of the SWT framework, there is no

function for handling double clicks. I also resigned to define functions related to GUI elements, as a cursors entering or leaving. Recognition of those were tasks for a GUI framework based on this input framework.

ICIMouseMoveListener

Move events are sent to this type of listener. The class implementing this interface has to be registered to the `ICIMouseProvider` to receive events. Compared to the AWT framework, there is no function support dragging events to separate the input from the GUI components.

ICIMouseScrollListener

Although it is not mandatory that mice have a scroll wheel, it is very common. For this reason I decided to provide an interface to receive scroll events. A scroll wheel event is not necessarily sent for each click of the wheel. This decision belongs to the implementation of the mouse provider.

ICIMouseEvent

Instances of this interface are used to provide detailed data on events sent to `ICIMouseListener` and `ICIMouseMoveListener`. This event class provides information about the coordinates of the current position of the cursor and if a button was pressed or released, and which one was used. In addition it knows which other buttons are currently pressed in a XOR value. To keep the input devices independent there is no function providing information if a modifier such as “Ctrl” or “Shift” is pressed on a keyboard. Having multiple keyboards defining this functionality would cause problems in an implementation, because different keyboards probably would have different state of the modifiers.

To be independent from any GUI framework and any implementation of the `ICIMouseProvider` interface, it was also impossible to use constants from one of those. `ICIMouseEvent` contains constants, for example for buttons for the input framework.

ICIMouseScrollEvent

This interface is used to define the information provided for an event caused by the use of a mouse scroll wheel and received by an instance of `ICIMouseScrollListener`. It extends `ICIMouseEvent` by a getter for the amount of clicks the scroll wheel was turned.

4.3 Key Provider

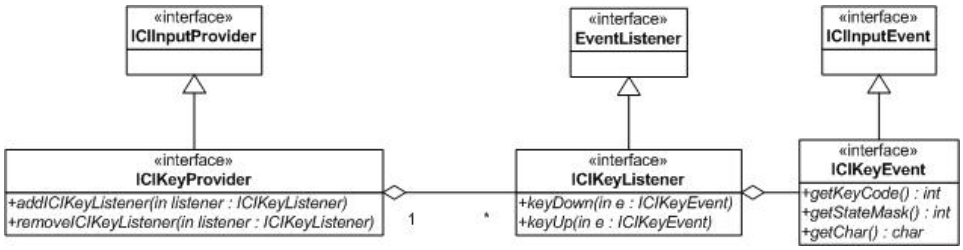


Figure 11: Class diagram of key provider

ICIKeyProvider

The `ICIKeyProvider` interface describes classes which provide for key based input. This could be a standard keyboard, only a number pad or an onscreen keyboard. The interface defines a function to add and to remove an `ICIKeyListener` to receive events from this instance.

ICIKeyListener

Events that can be received by instance of `ICIKeyProvider` are defined in this interface. The events that can be received are fired when a key is pressed or released. This is close to the SWT framework besides the naming of the methods and does not define a function similar to `keyTyped` in the `KeyListener` of AWT. I decided not to adopt the naming of those frameworks to maintain a naming convention within the framework. I did not take over the key typed event from the SWT to avoid a binding between the input and a user interface, because the focus of the GUI element could have been change between when the key was pressed and released.

ICIKeyEvent

Events sent by an instance of `ICIKeyProvider` are defined by an `ICIKeyEvent`. The instance of the event provides information on which key was pressed by giving

the Unicode value of the key. It also provides the current state of modifiers as “Ctrl” or “Shift” and with the `getChar` function the possibility exists to use the character directly by getting an “a” or “A” depending on the state of the “Shift” key.

4.4 Pen Provider

By contrast to [14] I did not achieve a general mapping of pen input to mouse events because it would result in a loss of a varying reaction of each input. If an application needs a handling beyond the standard processing of the Smart DViT table or other hardware is used which does not provide the mapping of pen to mouse input it is still possible to write a mouse provider up on the pen input.

Compared to the providers for mouse or keyboard input there are no widely-used GUI frameworks with pen support. So I was not forced to adapt methods.

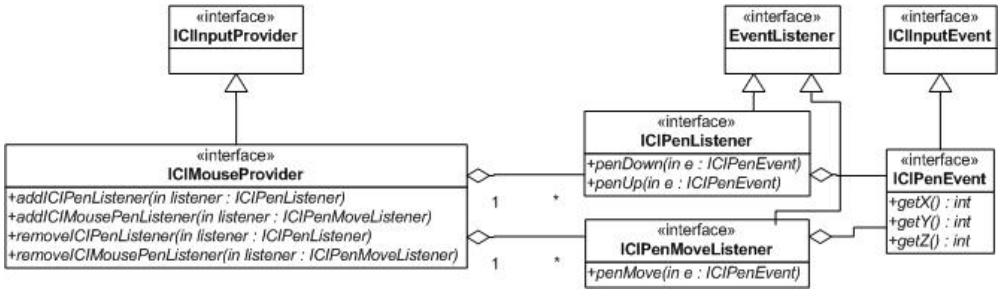


Figure 12: Class diagram for pen provider

ICIPenProvider

The `ICIPenProvider` extends the `ICIInputProvider` and defines an interface for pen input provider. It defines functions to add and remove for `ICIPenListener` and `ICIPenMoveListener`.

ICIPenListener

This interface defines events to be received when a pen or another object touches the surface or is removed, whereby it doesn’t matter if the hardware can recognize a difference between hovering over the surface and touching the surface. The value of `getZ()` from the received `ICIPenEvent` from this definition of pen down or up

has to be 0. For details of the function `getZ()` of the `ICIPenEvent` interface see the description of this interface later in the current chapter.

ICIPenMoveListener

The events caused by a movement either on the surface or above, it when recognized, are sent to the registered instances of this interface. Events are also sent for changing pressure of the surface whether the hardware can recognize this.

ICIPenEvent

`ICIPenEvent` defines the details which are provided for each event sent by an instance of `ICIPenProvider`. As there are x and y coordinates according to the `ICIMouseEvent`, the `ICIPenEvent` interface define an additional axis. If an implementation can recognize an object above the surface it will cause an event with a negative value for `getZ()`. If the hardware can detect different pressure changes, are shown in a positive value for `getZ()`. The value representing the distance to the table should be the distance in mm, the value for pressure a percentage to the maximum recognizable pressure.

4.5 Gesture Provider

The providing of gestures is the only higher level input in the framework, but gives an easy way to trigger more complex actions.

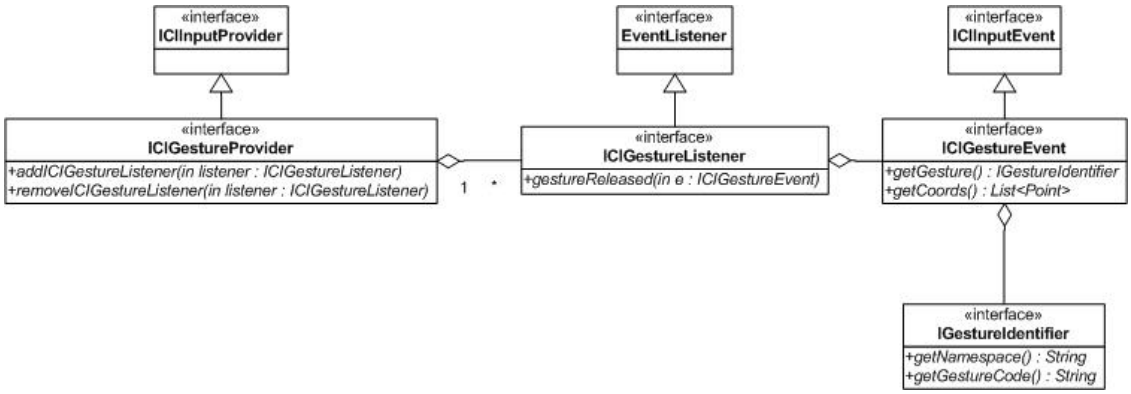


Figure 13: Class diagram of gesture provider

ICIGestureProvider

This interface provides the functions to add or remove listener to receive events based on gestures. Like all other providers for input `ICIGestureProvider` is derived from `ICIInputProvider`.

ICIGestureListener

If a listener of the type `ICIGestureListener` is register to an instance of `ICIGestureProvider`, it will be called if a gesture was released. Details to the released gesture will be provided by an `ICIGestureEvent`.

ICIGestureEvent

When a gesture is released and the gesture provider calls the registered listeners to notify them, detailed information about the gesture is provided in an `ICIGestureEvent`. The object contains the information on which gesture is being released. To get a more detailed description of the definition of a gesture see the description of the `ICIGestureIdentifier`. To be able to react based on the location where the gesture was made, the object contains a list of the coordinates. The content in this list is based on the implementation, so it is not guaranteed that the list contains every point of the gesture or a specific point as the beginning or end point. To avoid the use of an existing implementation, and thereby create a dependency on this implementation, the event uses the definition of the point form the framework.

ICIGestureIdentifier

A gesture is defined by a gesture code and a namespace. This makes it easy to define gesture with the same meaning for different objects and react to these definitions without a huge effort on parsing the information coded in one string. In Microsoft Excel for example there could be two “new” gestures, one for a workbook and one for a worksheet. The gestures then would be defined as shown in the table below.

Action	Namespace	GestureCode
Create a new workbook	Workbook	New
Create a new worksheet	Worksheet	New

Table 1: Example for different gestures with equal gesture code

The existence of a namespace also could be used in terms of differentiating a global definition and an application based definition of a gesture. In other words, a gesture with a global gesture code “new” might have the gesture code “reset” in a namespace of an application.

5 Implementation

After I introduced the interface to the individual parts of the input framework in chapter 4 I, now describe my implementation of that framework. My primarily goals were to support pen input and gestures with the integration of ManyMouse as a secondary goal.

5.1 Mouse Provider

5.1.1 Common

AbstractCIMouseProvider

This class implements all function defined by the `ICIMouseProvider` interface, while either storing the listeners in, or removing them from an instance of `EventListenerList` of the Java API [20]. To call the listeners from the subclasses, functions are provided.

CIMouseEvent

`CIMouseEvent` is a simple implementation of `ICIMouseEvent` which provides standard getter and setter for each variable but not for the variable `button`. The setter for `button` also updates the variable `buttonMask`. With the function `removeButton`, which is also updating `buttonMask`, this class can be used to store information of the state of a mouse pointer.

5.1.2 ManyMouseWrapper

Unlike Herbig [14] I chose a different approach to add ManyMouse into my implementation. To increase the reusability of the implementation I decided to separate the part which is responsible for reading and processing the mouse data from the part which displays cursor or releases events.
My implementation of `ManyMouseWrapper` consists of two threads and one list of instances of `MousePointerData`, used to store the current state for each mouse.

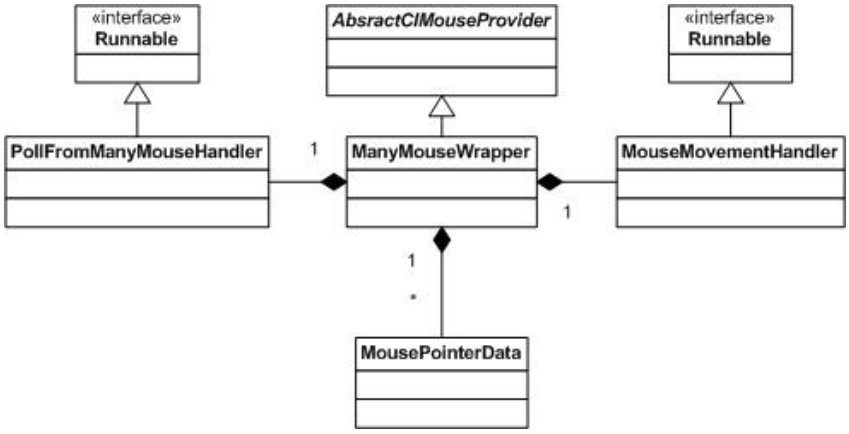


Figure 14: Class diagram of ManyMouseWrapper

The first thread gets the events from ManyMouse by calling the `PollEvent` function. The handling of the event depends on its type as shown in the diagram below.

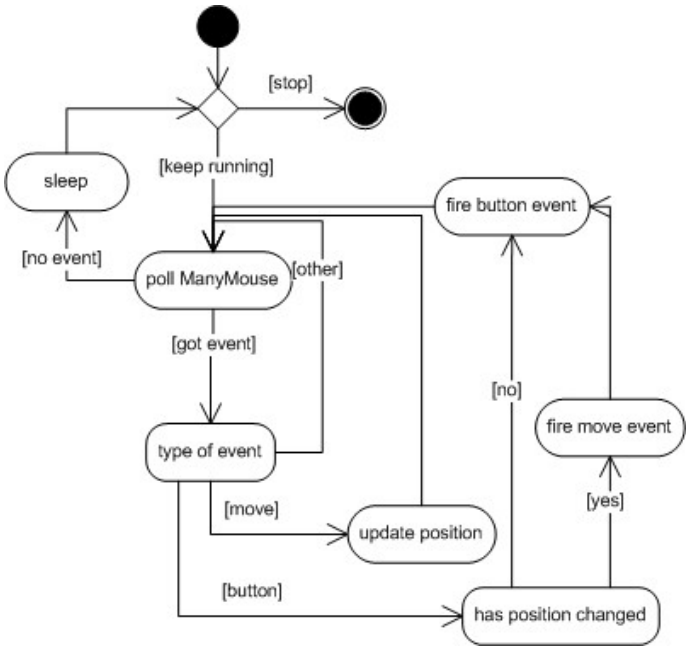


Figure 15: Activity diagram of the thread polling ManyMouse

As long as there are events in the queue of ManyMouse the threads keeps on processing them. If the event is representing a move event, the location in the corresponding entry of the list is updated. This is necessary because of the fact that ManyMouse only provides events with movements in x-Axis or y-Axis. If the event is representing a mouse button event, it will be checked if this mouse had changed its

position after the last notification of the listener to the move events first. If so, the listeners get called. After that, the listeners to the button events get called.
To be able to send the move events independent from the polling, I found it necessary to have a second thread. This thread iterates over the list with the mouse data and sends an event for each entry which had been changed since the last check.
`ManyMouseWrapper` is implemented in a singleton pattern. Even though this does not ensure an avoidance of interference with other applications, at least problems caused by the destructive reading within one application are excluded.

5.1.3 SwtManyMouseProvider

In terms of adding listeners to `ManyMouseWrapper`, applications now already have an opportunity to receive mouse events from the framework. However, for most applications, those events are still useless. The majority of applications expect coordinates relative to the top left corner of the object; as from the current GUI frameworks provided. The coordinates provided by `ManyMouseWrapper` however, are absolute to the top left corner of the display. Additional there is still a visual feedback such as a cursor for the user required.
With `SwtManyMouseWrapper` I show an exemplary solution to those problems. Within this class the coordinates are translated to the top left corner of the `Shell` passed in the Constructor and limited to their size.

```
private ICIMouseEvent calcMousePoint(ICIMouseEvent e) {
    // ...
    int x = e.getX();

    if (x < 0)
        x = 0;

    if (x > shell.getBounds().width)
        x = shell.getBounds().width;
    // ...
}
```

For every different id received by `ManyMouseWrapper`, a cursor objects location is updated, whereby the initial point is 0,0. In doing so it also deals with the SWT specific needs to access GUI elements.

Although `SwtManyMouseWrapper` uses `ManyMouseWrapper` the binding to this class is very weak and bases only on the instantiating. All other called functions are defined in the `ICIMouseProvider` interface, so that `ManyMouseWrapper` easily could be exchanged by another mouse provide.

5.2 Pen Provider

5.2.1 Common

AbstractCIPenProvider

This class implements the in the interface `ICIPenProvider` defined methods to add and remove listeners. To fire events to the registered listeners, this class provides functions to fire events. For each event of each listener type there is function to fire an event.

CIPenEvent

`CIPenEvent` is an implementation of the interface `ICIPenEvent`. It implements the interface defined getters and setters for each attribute and has a default and one parameterized constructor to initialize the attributes. For the default constructor all values are 0.

5.2.2 Smart

During the use of the SDK for the Smart DViT table some problems appeared. To solve those problems I created some classes. In this section I describe the problems and the classes that solve them

SWTSBSDKWrapper

Even if the functions of the `SWTSBDK` may be suggest that it is possible to add more than a single listener to a SWT widget it didn't work, neither in the way of adding multiple listeners to one instance of the `SWTSBSDK` nor instantiating the `SWTSBSDK`

several times and adding one listener to each instance. In both ways only the last registered listener was notified. For this reason I decided to write a wrapper class to access the Smart Board. This class registers itself as a listener to an instance of `SWTSBSDK` for a given `Control` and accepts `SBSDKListener`. Those listeners are stored for every `Control` and for each event the `SWTSBSDKWrapper` receives the respective functions of the corresponding listeners are called. The `SWTSBSDKWrapper` is implemented in the singleton pattern to avoid complications known of the direct use of the `SWTSBDK`.
This class also contains constants for the ids, which are used from the Smart SDK for the first and second object touching the surface as described in chapter 3.1

SBSDKAdapter
To get the event provided by the Smart Table, you have to register a class implementing the `SBSDKListener` interface. The problem is that this interface defines 18 functions where as in my whole implementation I only use five. To reduce the lines of code only to the needed functions and thereby increase the readability of the code I wrote the abstract class `SBSDKAdapter`, which implements all the function of the `SBSDKListener` interface. A derived class of `SBSDKAdapter` has only to override the needed functions instead of adding functions with empty bodies. This is similar to the implementation of `MouseListener` in the abstract class `MouseAdapter` in the Java API.

5.2.3 DViTBoard

The biggest problem with implementing an `ICIPenProvider` based on the events of the Smart DViT technology was the poorly documented SDK. The only obstacle with working with the Smart SDK was finding out how to use it. With the implementation `CIPenEvent`, the listener handling in the `AbstractCIGestureProvider` and using the helper classes described in chapter 5.2.2 to solve the SDK issues the implementation of the `ICIPenProvider` based on the Smart SDK resulted in a straightforward implementation.
In the constructor the `DViTBoard` registers the passed control to the Smart SDK using the `SWTSBSDKWrapper` to receive pen events. Using an inner class to extend

`SBSDKAdapter` and overwriting the `onXYUp`, `onXYMove` and `onXYDown` functions to forward the events by using the methods to fire events of the `AbstractCIGestureProvider`. In the `stop()` function defined in `ICIInputProvider` it deregisters the listener from the Smart SDK to end receiving events.

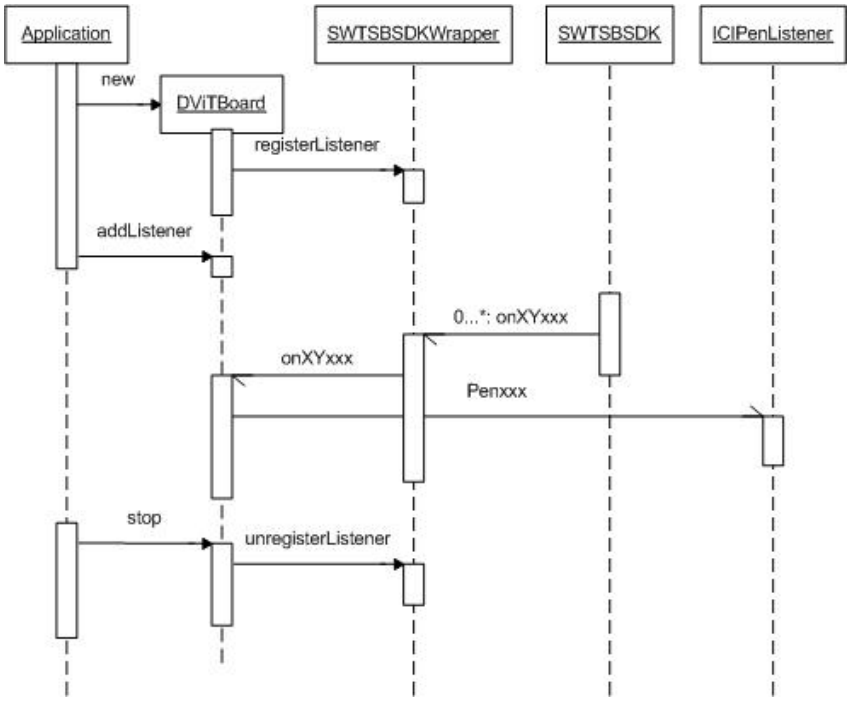


Figure 16: Sequence diagram for DViTBoard

5.3 Gesture Provider

5.3.1 Common

AbstractCIGestureProvider

The abstract class `AbstractCIGestureProvider` implements all functions defined by the `ICIGesutreProvider` interface and provides a function to fire an instance of `ICIGestureEvent` to all registered listener for the subclasses. This reduces the need to implement the listener handling in all implementing classes of the `ICIGestureProvider` interface.

CIGestureEvent

`CIGestureEvent` is a simple application of the `ICIGesutreEvent` interface. This class implements all getters defined by the interface and setters for the id and gesture attribute. For the coordinates it provides functions to add a single `Point` or an `Enumeration` of `Point` and one function to clear the list of coordinates.

Gesture

The `Gesture` class is a simple implementation of the `ICIGestureIdentifier` interface. The class implements the getters and provides getters for the namespace and gesture code attribute as well as a default constructor and one to set both attributes. Additionally `Gesture` overloads the functions `equals` and `hashCode` of the `Object` class. This makes it possible to compare the instances or store them in a hash table. The `toString` method gives a string back which consists of namespace and gesture code separated by a colon and surrounded by brackets.

DefaultGestures

This class defines constants for standard actions. Standard actions are actions that appear in most applications. For example `DefaultGesture` defines instances of `ICIGestureIdentifier` for:

- cut, copy and paste
- undo and redo
- new
- save

The namespace of all actions is “default”.

DefaultGestureAdapter

`DefaultGestureAdapter` is an abstract class implementing the `ICIGestureListener` interface. The class contains functions corresponding to each gesture defined in the class `DefaultGestures`. For example the gesture `DefaultGestures.COPY` has a corresponding function `Copy(ICIGestureEvent e)` the gesture `DefaultGesture.CUT` a corresponding function `Cut(ICIGestureEvent e)` whereupon the body of

every function is empty. In the `gestureReleased` method, which receives the gesture events from the gesture provider, the class checks the released gesture and calls the corresponding with passing the received event. A subclass now has only to overwrite the needed functions. In other words, if appropriate an application does not have to handle the gesture event itself.

5.3.2 Architecture

To get a reusable implementation of the `ICIGestureProvider` interface I found it necessary to divide the implementation in two parts. The first one is for wrapping the input and the second one for the recognition of the gesture. For this reason I defined interfaces to interact with those parts.

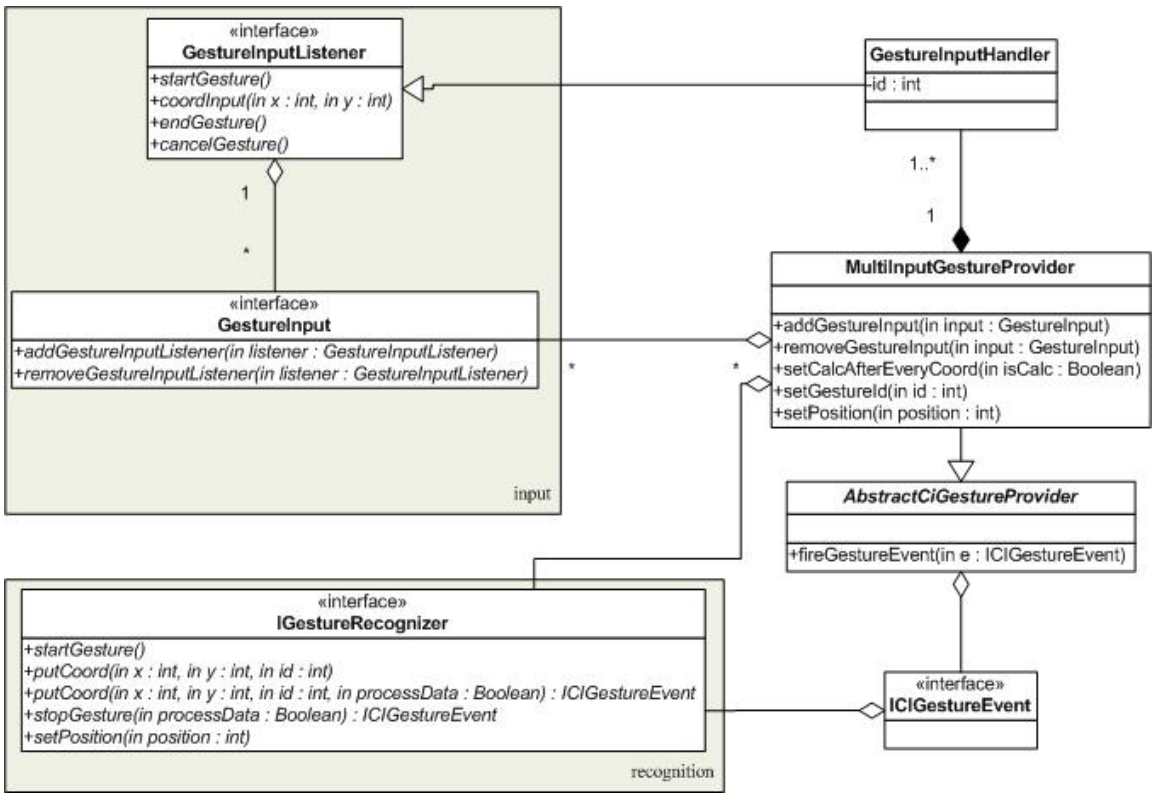


Figure 17: Class diagram for gesture provider implementation

MultiInputGestureProvider and GestureInputHandler

The class `MultiInputGestureProvider` implements the interface `ICIGestureProvider` by extending the abstract class `AbstractCIGestureProvider`. This class manages receiving of input,

forwarding of the input to the gesture recognizer and firing gesture event to the registered listeners over the functions provided by the `AbstractCIGestureProvider` class. For every gesture input a new instance of `GestureInputHandler` will be created, which registers itself as a `GestureInputListener` at the gesture input. Every instance of `GestureInputHandler` gets a unique id is which is used to differentiate the inputs from each other. With `setCalcAfterEveryCoord()` it is possible to tell the gesture recognizer to process the all already given data and try to detect a gesture. The consideration of the position belongs to the gesture recognizer and for this reason `setPosition()` just forwards the information to the recognizer. It is also possible to set an id which will be set in the fired gesture events.

GestureInput

`GestureInput` defines functions to add and remove listener. Those listeners get called, when an input event occurred. To see a detailed description of the event see the specification of `GestureInputListener` below. It also defines a function which returns if a gesture is currently running. I thought about managing this information in the `MultiInputGestureProvider` but as the implementations of `GestureInput` should send only events with new coordinates, when a gesture is running, this information has to already be there.

GestureInputListener

The `GestureInputListener` defines four types of events which an instance of `GestureInput` can send to the registered listeners. First, an event tells, the conditions to start a gesture for this gesture input are apply, second, the passing of the coordinates. A gesture ends with calling `endGesture()` of each registered `GestureInputListener`. If, for example, a time out or pressing another button causes an invalid conditions to appear, `cancelGesture()` is called. The implementation of `GestureInput` has to ensure, that events with coordinates are only sent when a gesture is currently being made. In other words, calling `coordInput()` is only allowed after a start event and not after an end or cancel event.

IGestureRecognizer

If the conditions to start a gesture apply for all inputs `MultiInputGestureProvider` calls `startGesture()` of `IGestureRecognizer` interface. After the start command each instance of `GestureInputHandler` will forward the received events with its unique id. If the overload `putCoord(x,y,true)` or `stopGesture(true)` is called the gesture recognizer will process the data and if applicable return an instance of `ICIGestureEvent` which contains the information about the released gesture. A call of `stopGesture()` will tell the gesture recognizer that it can clean up the data independent of processing the data. Due to the reason that there is only one attribute position for the gesture recognizer and not for each input the recognition of gesture is limited to gestures only from one side of the table. However I did not find useful gestures made by more than one person which would legitimate the additional effort.

5.3.3 Providers of Input

The class diagram below shows all implementations of the gesture input. It highlights the actual source of the input so that is easier to recognize the separation between the input and the processing of the data. It also shows the flexibility of the input. The additional level of abstraction allows a higher flexibility based on more simple implementations and an easier integration with current GUI frameworks by preventing the need of implementing a version of `ICIMouseProvider`.

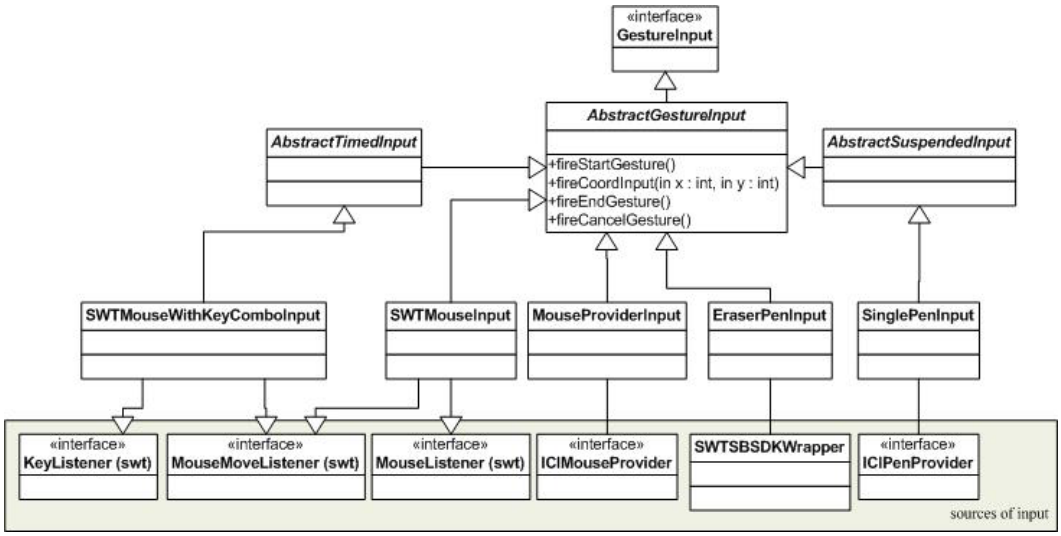


Figure 18: Class diagram for gesture input

AbstractGestureInput

To avoid the need of implementing the listener handling in every subclass of `GestureInput`, the abstract class `AbstractGestureInput` implements the in the interface defined functions. To offer the subclasses a possibility to send events to the registered listeners `AbstractGestureInput` supplies function for each function defined in the `GestureInputListener` interface. The function `isGestureRunning` is also implemented by this class.

AbstractTimedGestureInput and AbstractSuspendedInput

These two classes implement time based functionality to the extending classes. Both classes override the methods for firing events to the listeners in `AbstractGestureInput`. While `AbstractTimedInput` sends an event to cancel the gesture after a certain amount of time without any input, the `AbstractSuspendedInput` class guarantees, that after the start event from the subclass, no movement outside a tolerance is made or the start event will never be forwarded. Both implementations will not forward events send by a subclass, if a cancel event caused by the timing was sent.

MouseProviderInput

This class provides input for the `MultiInputGestureProvider` based on the definition of a provider of mouse input from the framework. It registers itself as a

listener at the passed mouse provider and reacts only for events matching the passed id. A gesture starts with pressing a configurable mouse button and ends releasing it.

SWTMouseInput

Beside the fact that `SWTMouseInput` is derived from `AbstractTimedInput` its functionality is quite close to `MouseProviderInput`. As source of the input uses this input provider the instance of `Control` which is passed in the constructor by registering itself as a `MouseListener` and `MouseMoveListener`. Thereby is to consider that the selected button from the constants in SWT and not of the input framework.

SWTMouseWithKeyComboInput

As the `SWTIMouseInput` this class uses the passed instance of Control to receive events, however, gestures start not while pressing a mouse button but using key board events. It is possible to set an individual or a combination of modifier and character which causes a gesture to start. Is any events received of another key pressed or released an event to cancel the gesture will be sent.

SinglePenInput

As `MouseProviderInput` this gesture input provider is based on the input framework. In the constructor it registers itself to the passed pen provider. A gesture starts with a pen down event and ends with a pen up event, whereby the incoming events are filtered by their id. As a subclass of `AbstractSuspendedInput` it is possible to set a time, which the pen must not move before the gesture starts.

EraserPenInput

The `EraserPenInput` uses the Smart board framework to receive events. It reacts on which pen is removed from the table. Event to start or stop the gesture and coordinates of the gesture are only sent if the eraser was the last pen removed from the table. If the other pen is removed while making a gesture, a cancel event is fired.

5.3.4 Recognition of Gesture

5.3.4.1 DirectionRecognizer

It is a very common approach to detect gestures based on the movement of mice. Due to the fact that I had a look into the gesture plug-in for Mozilla Firefox developed by optimoz[1] the classes

- `Directions`
- `AbsoluteCoordReconizer`

are licensed under the LGPL[2].

Directions

First I defined constants of the individual directions. For that I used a related analogy as the Mozilla Firefox plug-in. Starting from the key "5" on the number pad of a keyboard a direction is defined to the number which is reached. So the value of the constant for the direction up is "8", the value of constant for the direction diagonal to the up left is "9" and so on. This useful while having a number pad in front of you, during coding or debugging your application and have the possibility to peek.

Figure 19: Definition of directions analog to keypad

[1] http://optimoz.mozdev.org/

[2] http://www.gnu.org/licenses/lgpl.html

DirectionRecognizer

The class `DirectionsRecognizer` is an implementation of the interface `IGestureRecognizer`. This class makes a separation of the recognition of the direction and the interpretation of the list of collected directions. This gives this class potential for multiple reuse. In the current implementation the class `DirectionRecognizer` uses the class `AbsoluteCoordsRecongition` to recognize the direction. The `AbsoluteCoordsRecongition` compares the actual coordinates with the last one. To be processed the diagonal distance between the two positions has to be bigger as a settable value. This tolerance turned out useful to remove jitter, especially from pens causing a movement of one pixel resulting in a movement in one direction. The direction itself is recognized by the angle of the line between the two points. Between each two directions there is a settable angle where no direction is recognized. This was important for movements along the line where the recognized direction oscillated.

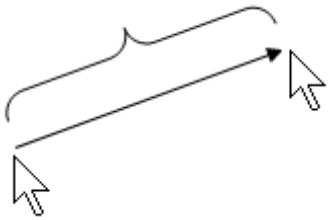


Figure 20: Minimum of distance for movement

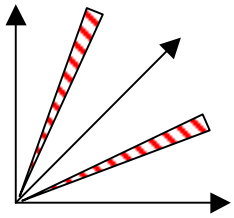


Figure 21: Angle where no direction is recognized

The recognized directions, once stored, are processed to remove duplicated entries removed. For example “77388886” will result in”7386”. If the call from the using class desires, `DirectionRecognizer` will pass the list of directions to an instance of the `IGestureTable` interface which will try to find a matching gesture and returning an instance of `ICIGestureIdentifier` if found or `null`.
This separation makes it easier to change the behavior of the recognizer. First by changing the `AbsoluteCoordRecognizer` into a recognizer not only based on the last two coordinates but based on the last n coordinates or by having different ways to assign gestures to directions. At which the simplest solution would be a hard

coded version of `IGestureTable` and a more complex but also more flexible solution reading the directions and assigned gesture from a file.

5.3.4.2 PositionRecognizer

Another implementation of the interface `IGestureRecognizer` is the class `PositionRecognizer`. This recognizer basically compares the coordinate of the last point of the gesture with the coordinates of the beginning point. Depending on the position of the last point, a gesture event is fired. For that the recognizer differentiates between four quadrants outgoing from the starting point. For each quadrant an individual gesture can be set. It is possible to define the size of a rectangle around the starting point, to end the gesture without releasing an event. This recognizer could be used to assign one out of four states to an object. For AgilePlanner a use could be setting the state of a story card to “Defined”, “In Progress” or “Done”.

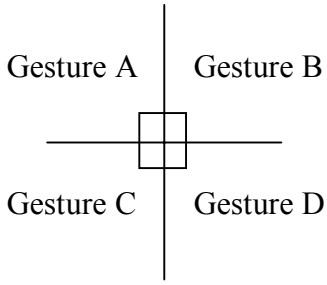


Figure 22: Releasing gesture based on position relative to start point

6 Integration in Agile Planner

In this chapter I describe the exemplary integration of the input framework in AgilePlanner. This bases on the implementation described in chapter 5 completed by a couple of specific classes. First I describe additional classes and the setup of the framework for AgilePlanner. In 6.2 I list the gestures supported in AgilePlanner by the framework. This chapter ends with a section about the linking of framework and AgilePlanner.

6.1 Input Framework for AgilePlanner

APInputFrameworkConfig

Currently the source code of the input framework is in the same project as the application AgilePlanner itself. To guarantee that the application itself is not affected by the development process of the input framework, this class contains a Boolean value if the framework should be loaded or not. With a further attribute it is possible to configure, if the framework for the table should be used. This was needful because I tried to code as much as possible at my desktop pc and the launching of AgilePlanner was suspended by a long time out of the Smart framework while trying to connect to the table. An additional attribute is to enable the mouse provider. The last attribute turns the output of debugging information to the standard output on or off. In the version of this in the head of the repository all three attributes have the value false, whereby I changed them on my development PCs and not committed them. Another option had been reading from a configuration file, which also would have resulted in not committing the changed file with a higher effort in realizing.

AgilePlannerGestures

This class defines parallel to the `DefaultGestures` from chapter 5.3.1 gestures for the use in AgilePlanner. The namespace for all gesture is “AgilePlanner”. Defined gestures for example are

- create a new iteration or story card
- set a story card as completed
- collapse or expand story cards

AgilePlannerGesturesAdapter

The abstract class `AgilePlanenrGesturesAdapter` extends the `DefaultGestureAdapter`. The functionality is the according to the `DefaultGestureAdapter` by implementing the `gestureReleased` method and forwarding the received event to functions with a name corresponding to the received event. `AgilePlannerGestureAdapter` overrides the implementation of the super class and checks after calling the `gestureRelease` of the `DefaultGestureInput` if the gesture matches one defined in `AgilePlannergestures`. It was not mandatory to derive the class from `DefaultGesturesAdapter`. It had been also possible to derive `AgilePlannerGestureAdapter` as well from the `ICIGestureListener` and only check for the gestures defined in `AgilePlannerGestures`. But this would cause the need of two gesture listeners in AgilePlanner, one checking for the default gestures and one checking for the AgilePlanner specific gestures.

APInputFramework

This class creates the input framework used in AgilePlanner which consists of a pen and gesture provider. As there is no implementation for a keyboard provider, this feature is unaccounted for. The use of ManyMouse as a provider for multiple mice support is limited to input for the gesture provider and not provided for other interactions with AgilePlanner. This was inevitable because of the fact, that there is currently no stable implementation of a mouse provider supporting this. `APInputFramework` creates one instance of `DViTBoard` as pen provider and several instances of `MultiInputGestureProvider` with variable inputs as gesture provider. To give access from outside to the events released by the created providers, `APInputFramework` implements the interface `ICIPenProvider` and `ICIGestureProvider`. The implementations of the add and the remove function for the listeners forward the listener to the corresponding functions of the pen provider or to each gesture provider.

AgilePlannerGestureTable

`AgilePlannerGestureTable` is a subclass of `AbstractGestureTable` and defines gestures based on the direction of mouse or pen movements. For details on the direction based recognition in the framework see chapter 5.3.4.1. The individual defined gestures are described with all other gestures provided for AgilePlanner in chapter 6.2.

BoardGestureIDSetter

The possibility of handling gestures depending on the position of the table the user stands is a nice feature but without any association of pens to user or positions there is no usage for this feature. To work around this disadvantage I defined one color and one id for each side of the table. The class `BoardGestureIDSetter` uses the fact, that the Smart DViT table sends event when a pen is removed and the current color for a Smart board can be retrieved. Using this functionality, the class reacts on the event of a removed pen and sets the position and id for an instance of `MultiInputGestureProvider` according to the color of the removed pen.

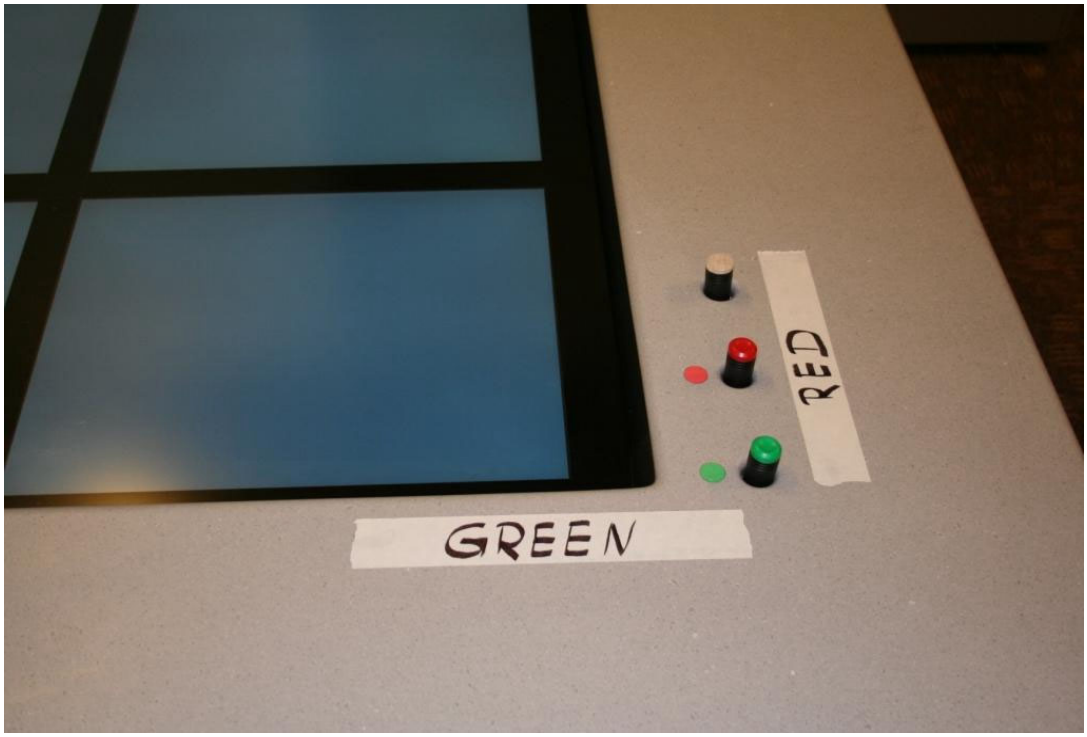


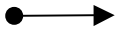
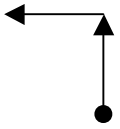
Figure 23: Colors of pen associated to positions around the table

6.2 Gestures implemented in AgilePlanner

The recognition of most gestures realized in AgilePlanner are based on the `DirectionRecognizer` described in chapter 5.3.4.1 and therefore on the direction of the movements with the input device.

Direction based gestures

Gestures with a single device are possible either with one mouse by pressing the "Ctrl" key or using a pen of the table. While using the table you have to consider, that the colored pens are each assigned to one side of the table and by using the `BoardGestureIDSetter` the gestures are translated to the appropriate orientation.

Action	**Gesture** (*namespace*:*gestureCode*)	**Directions** From the users view
Undo	(Default:undo)	
Redo	(Default:redo)	
Collapse story cards	(AgilePlanner:storycard-collapse-all)	

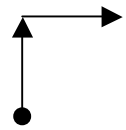
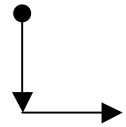
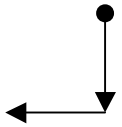
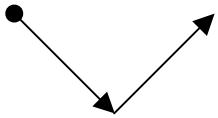
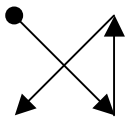
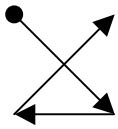
Expand story cards	(AgilePlanner:storycard-expand-all)	
New story card	(AgilePlanner:storycard-create)	
New iteration	(AgilePlanner:iteration-create)	
Set complete story card	(AgilePlanner:storycard-set-completed)	
Delete story card	(Default:delete)	
Delete story card (second gesture)	(Default:delete)	

Table 2: Direction based gestures made with one mouse or pen

Rotating, resizing and moving story cards

Especially for using AgilePlanner from different positions of the table creates the need of rotating cards. The gesture based on two pens allows the user to orient the

card to their own position, but limits thereby the possibilities of simple direction based gestures with one input devices. It is also an intuitive way of resizing card by putting two fingers on it and enlarging it by pulling or shrinking it by moving the fingers together. This gesture can be made from any side of the table as long one pen is removed to enable the pen input. The rotation angle and resize factor are relative to the points where the card was initially grabbed. Concerning the use of the provided command stack which manages the executed commands and enables undo and redo two different gesture events are sent. The first one indicates that the story card is currently rotated, the second one, that the changing has ended. This will allow AgilePlanner to show a preview of the current story card and provide an easy way to undo the changes by only adding one command to the command stack.
Due to the fact, that the feature of rotating story cards is not implemented in AgilePlanner I draw rectangle of the computed size, location and rotation to simulate the reaction to this gesture as showed in the picture below.

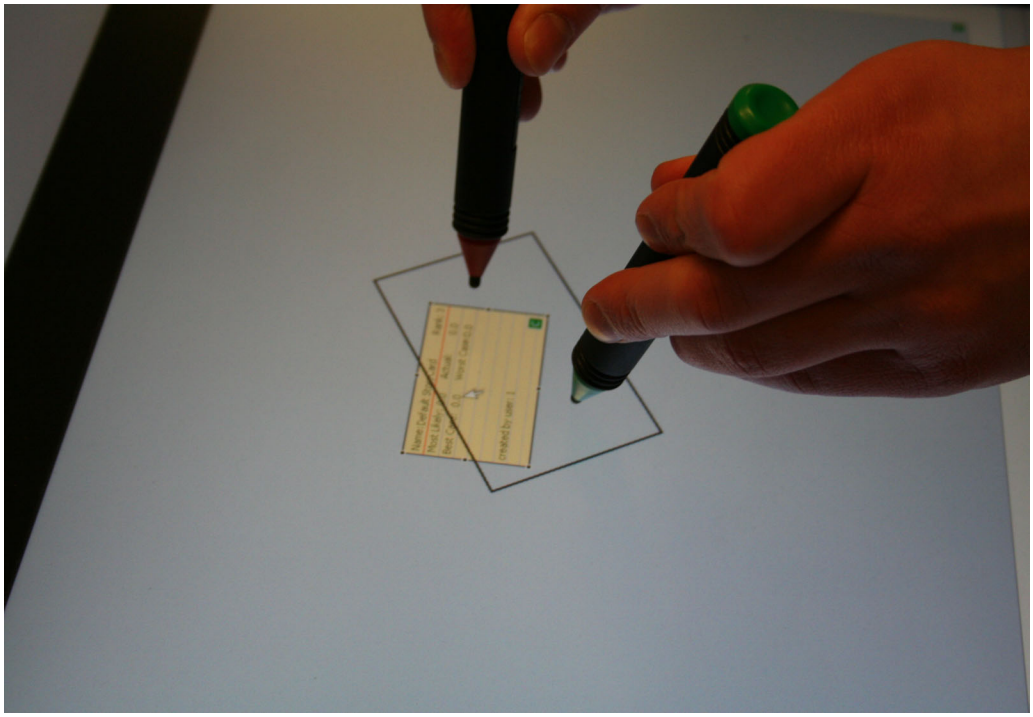


Figure 24: Simulating the rotation, resizing and moving of a story card

Deleting story cards with eraser

Taking the eraser of the table and moving over a story card to erase it is an even more intuitive way than crossing it out with a pen. But with only two erasers it is not

possible to assign one eraser to one side of the table. With that there is no generic way to log the user who has deleted the story card. To negotiate this, a dialog window appears when an eraser is removed where the user can select his id by clicking on the color.

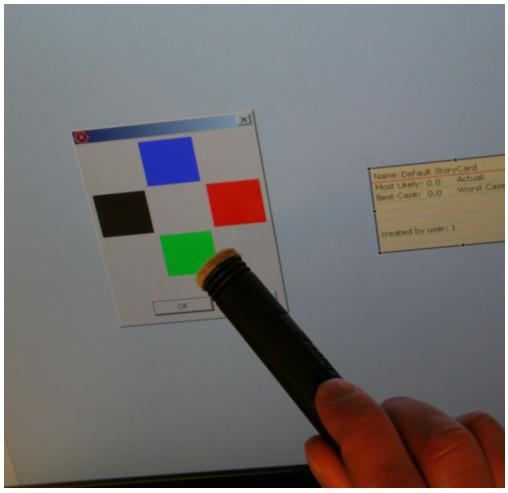


Figure 25: Dialog to select user for deleting a story card with the eraser

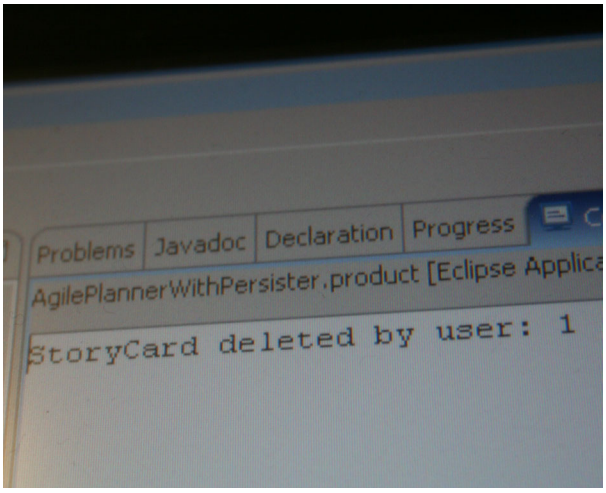


Figure 26: Recognition which user has deleted the story card

Setting of best case value

It is possible so set the value for the best case of a story card by using a mouse. The gesture is activated by pressing “Ctrl +B” while the mouse cursor is over a story card. As long the keys are pressed, every movement updates the best case value now. The value depends on the distance between the story card which was under the mouse cursor and the current position of the mouse cursor. This gesture is not implemented for the use with pens because of the problem with interferences of the several gesture providers and unhandy combination of keyboard and pen use.

6.3 Accessing the Framework in AgilePlanner

To run AgilePlanner with the described input framework there are two libraries needed

- “ManyMouse.dll” in the project root folder to access ManyMouse
- “sbsdk.jar” in the build path and the “plugin.xml” and “RegistrationUtils.dll” in the project root folder to get pen support

As the input framework and especially the gesture input is planned to be mainly used in the planning area of AgilePlanner, to interact with story cards , the setup of the input framework is done during the creation of the graphical objects of the planning area. Concretely in AgilePlanner this is the class `Editor`. To easily interact with the graphical objects in this area, the listener which reacts to the gesture events sent by the framework is in this class. But to separate the code reacting to the gestures and the rest of the code in the class `Editor`, which is mainly used to set up the planning area, the event handler is realized as an inner class.

For the reactions to the gestures I tried to use as much of the existing functionality of AgilePlanner as possible. This should show that the framework is easy to integrate in applications by using functions which are independent to the GUI of the applications. In AgilePlanner this code is realized as implementations of the abstract class `Command`. While using these commands, it allows also the use of undo and redo over GEF, if the commands implement these features. The remaining code is basically responsible for generating the information needed to execute those commands. For example calculating the point where a story card is created or which story card should be deleted based on the coordinates provided by the gesture event.

7 Conclusion and Future Work

In this chapter, I summarize my thesis. First I will describe the problems which appeared during the work on my thesis. Then I will review the goals of my thesis and compare them to my results. I close with an outlook and suggestions for future work.

7.1 Problems

Code Repository

As mentioned in chapter 3.3 AgilePlanner is realized as an Eclipse plug-in. But unfortunately plug-in projects in Eclipse can only include other projects which are plug-in projects themselves or JAR files. To neither get the overhead of a Eclipse plug-in project nor the necessity to rebuild the jar file and update the AgilePlanner project I moved the sources of the input framework in a subfolder of the AgilePlanner project.

Although the code for pen input and gesture support reached a stable point after the work during my thesis, in my opinion it does not make sense to separate the code bases yet. Through the persisting need of work on other parts of the framework it is still easier to develop and test with AgilePlanner. But on a long term view the separation of the two code bases should be a goal.

Smart DViT

Beside the poor documentation of the API of the Smart DViT touch system, there also appeared to be problems with the precision of the table. While trying to use the table it turned out, that the precision does not increase continuously with increasing the amount of calibration points. From the possibilities of 4, 9 20 or 80 calibration points per screen, I observed that the best result was achieved with 9 calibration point per screen. Besides the huge time effort to calibrate it with 20 and 80 points per screen, it was found to be less stable in terms of recognition of pens.

Concurrent Input

I tried to be concerned with the possibility of concurrent input from users as much as possible. My first goal was to realize this based on concurrent input from the digital table. But with only the digital table based on DViT technology available, there was

no possibility to realize that. The above mentioned caused for a need to have another device supporting this particular feature. The difficulty of this I describe in the next section.

Existing implementation of mouse provider

Unfortunately I could not use the results of [14] directly. I could, however, use the knowledge of how to interact with ManyMouse and hence had to rewrite the entire mouse provider. This was due to the fact that the available implementation as a prototype led to the discovery of some bugs. It is also based on interaction with Microsoft Windows, which I can not see as a possible solution for a framework supporting concurrent input as long as it is based on adding events in the Windows message queue.

AgilePlanner code stability

AgilePlanner was subject to a major refactoring during the time of my thesis. This refactoring was done in order to separate the application and data layer. This was to provide greater flexibility for AgilePlanner in terms of future maintainability. But this unfortunately resulted in instability of the most parts of the application during the time of the refactoring. Also the work on the feature supporting the visibility of remote cursors, which needs to listen to mouse event in the planning area, caused times of unpredictable behavior of AgilePlanner. Especially during the time where I integrated the input framework in the AgilePlanner code to call commands were pervaded by troubles while trying to find out if the reason for error was caused by code of the input framework or AgilePlanner itself.

7.2 Thesis Contributions

In chapter 1.3 I described the goals of my thesis. At this point I want to review my accomplishments and compare them with the goals.

Definition of framework

In the first step I defined the interface of the framework. With the defined interfaces for the various input devices, as well as gesture input, it is possible to develop applications independent from hardware.

Pen input for table

One of my primary goals was to implement the support of pen input for the framework. After a great deal of difficulty, caused by the inferior documentation of the touch sensitive hardware and the calibration, I developed a clear version of a pen provider based on the SDK for the Smart DViT technology used in the present digital table.

Gesture support

The next step was implementing recognition of gestures based on this pen input. The drawback of the Smart DViT technology which is not supporting the recognition of a user, I implemented a work around for that drawback, by assigning a different color representation to each one side of the table in terms of pen colors. This information I used to set the id and the independence of the same gesture from the position of the table. With the realization of providing gestures for applications running on the table and hence I achieved the second major goal of my thesis

Additional support of gesture for mice

While trying to implement a reusable gesture provider, my implementation can also be used based on mouse input. This was possible by separating the input from the recognition of the gesture. The process of making gestures also available for mice was also driven by circumstances encountered during the development process. The problems caused by the touch sensitive hardware of the table and the fact that it is not comfortable to code in front of a 10mega pixel display with bezels, forced me to investigate other avenues in search of other possibility to test my gesture recognition.

Rewriting of the mouse provider based on ManyMouse

With the need to test the gesture provider using concurrent input and the impracticality of using the existing implementation, I rewrote the mouse provider based on ManyMouse. Even though I did not implement all the features shown by the earlier prototype, I think I provided a stable version that will allow the implementation of those features at a later date.

7.3 Future work

The missing implementation of a keyboard provider and the mouse provider reduced to basic functionality shows the input framework is not completed yet. In this chapter I want to describe my ideas to complete the above and the next steps in this way.

Additional input provider

To provide a larger amount of possible environments to run applications using this framework, additional implementations for input provider are needed. This would allow the use of AgilePlanner or other applications using this framework also in digital table environments different from the development environment. It also would allow extending AgilePlanner to replace conventional white boards used as a story boards. A large plasma television with touch sensitive hardware could be mounted instead of the white board and developer could move story cards from one column to the next while using their fingers touching the display. This would negotiate the need to transcribe the story cards from AgilePlanner to index cards out of paper.

Dynamic aspect

The interface I defined for the framework concentrates on a static framework and does not take explicitly into account the adding or removing of input devices. This would be useful for visual representations of input devices. As in the current implementation of the mouse provider for SWT, creates an instance of a cursor with the first event for each id, this could be replaced by an event for adding a device and the cursor could be destroyed receiving an event of a removed device. I think this feature is not of high priority, because the most applications should not rely on it, but it would be nice to have this feature later.

User management

With the continuing use of the framework, it would be also nice to have a user management feature that would reduce the need to implement this in each application. Under user management I understand the possibility to configure user specific values as name, color and so on and to make those values persistence beyond the runtime of the application. The data object representing a user could be assigned to one or more input devices, which visual representation automatically it could adapt the settings

from the user defined in the data object. Concretely, the color and name shown in a mouse cursor would be defined in XML file. During the start up of the application the user management would ask the users to assign their configuration objects with the id of this mouse and all necessary configurations will be done from the user management. In a scenario, with the dynamic aspect of the input framework mentioned above, a dialog to choose one out of the configured users could pop up when a new mouse is attached. The user management also could support different roles for the user, which have individual settings. For the example in AgilePlanner this could be the roles of developer and customer, where the developers have gestures to set the effort and customers have gestures to set the priority of user stories.

Visualization of Gestures

During first experiments with the recognition of gestures, I used the mouse events provided by SWT framework. In the test application I also implemented a visualization of the gesture. More precisely, I drew a colored line based on all points the mouse cursor hovered during the gesture. With changing from the desktop environment to the digital table I found the visualization more distracting in general, and more specifically, the use of gestures based on more than a single input. With refactoring the code to the current architecture, I discarded this feature focusing to the recognition of gestures for pen input. But with using the same code also to provide gestures for desktop systems, the above discarded feature would be nice to have again. However the visualization can easily integrated in the current architecture.

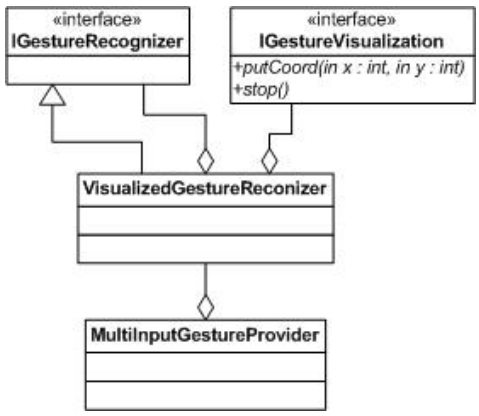


Figure 27: Visualization of gestures in the current architecture

A class `VisualizedGestureRecognizer` has to be created which will be registered as the gesture recognizer to the `MultiInputGestureProvider`. The

class will forward the calls to the actual gesture recognizer set in this class and to an instance of the `IGestureVisualizer`. This interface will probably need only one function to receive the coordinates and one function to signal that the visualization can be cleared. This also allows the separation of the code interacting with the visual elements.

Visualization pen/hand input

One of the main advantages of using a digital table is the support of the human to human interaction. It can be specified which person is talking only by pointing on the card, without the need of interacting with any other input device as a mouse. While interacting with the table, no visual representation like a mouse cursor is needed, because all group members gathered around the table can view the position of both the pen and the finger. However, a visual representation, such as a mouse cursor, may not be visible if it is covered by the pen or finger.

But in terms of running distributed applications running on a digital table such as AgilePlanner, it would also be useful to have a feature supporting the position of a pen on the other clients table, to provide this part of the human to human interaction for distributed teams. Therefore strict attention should paid to the fact that an unintentional pointing gesture may be recognize as gestures and may cause undesirable actions in applications.

GUI components for input framework and Concurrent Input

A final step after realizing all different input providers would be to integrate the input framework with a GUI framework, as Herbig [14] and Hoemig [15] mention the combination of their results in their future work section. While my goal was to provide as much table relevant input as possible, some day it would be nice to have a GUI framework supporting the table relevant functionality for widgets. Useful GUI functionality would be the support of rotating widgets to orient them to a user position or zoom, to make it easier to read on high resolution screen from the far end of the table.

As a first step the integration could be done as described in their thesis. But as the support of real concurrent input will be sooner or later, an inevitable feature, in my

opinion, the final implementation can not be based on the message queue of Microsoft Windows. As mentioned in [14] and as the architecture used in the SDGToolkit [42] the only solution, besides an operating system supporting concurrent input directly, is to rewrite all widgets that should be possible to be used concurrently. I can not see the second solution, only overwriting a single widget which acts as a dispatcher, as a feasible implementation, because this would be an application specific workaround which could not be reused in a framework.

Gesture repertoire for AgilePlanner

The gestures currently supported in AgilePlanner are not the result of a research based analysis. It is a mixture of own experience using AgilePlanner, casual conversations with colleagues within the e-Business Engineering Group about their wishes of gestures and the need to present the results of my work with the various kind of gestures. To get a well-founded set of gestures it would be appropriate to have a user study to ask the users about which actions they want to have supported by gestures and which gesture they prefer for each action.

List of Abbreviations

AWT – Abstract Window Toolkit

DViT – Digital Vision Touch

IDE – Integrated Development Environment

GEF – Graphical Editing Framework

GUI – Graphical User Interface

HCI – Human Computer Interaction

JAR – Java Archive

JNI – Java Native Interface

LGPL – Lesser General Public License

SDG – Single Display Groupware

SDK – Software Development Kit

SVN – Subversion

SWT – Standard Widget Toolkit

UML – Unified Modeling Language

URL – Uniform Resource Locator

USB – Universal Serial Bus

References

[1] Agarawala, Anand et al.; Keepin' It Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen; 2006

[2] Beck, Kent et al.; Manifesto for Agile Software Development (Online, http://agilemanifesto.org/) ; Viewed March 19th 2007

[3] Beck, Kent et al.; Extreme Programming Explained; 2005

[4] Bi, Xiaojun et al.; uPen: A Smart Pen-liked Device for Facilitating Interaction on Large Displays; 2006

[5] Chen, Fang et al.; ViCAT: Visualisation and Interaction on a Collaborative Access Table; 2006

[6] Danube Technologies Inc.; ScrumWorks – Product Overview (Online, http://danube.com/scrumworks); Viewed January 12th 2007

[7] Dietz, Paul et al.; DiamondTouch: A Multi-User Touch Technology; 2001

[8] Eclipse Foundation Inc.; Eclipse Graphical Editing Framework (GEF) (Online, http://www.eclipse.org/gef/); Viewed March 20th 2007

[9] Everitt, Katherine et al.; MultiSpace: Enabling Electronic Document Micro-mobility in Table-Centric, Multi-Device Environments; 2006

[10] Everitt, Katherine et al.; Observations of a Shared Tabletop User Study; 2006

[11] Gordon, Ryan; ManyMouse (Online, http://icculus.org/manymouse/); Viewed November 3rd 2006

[12] Ha, Vicki et al.; Direct Intentions: The Effects of Input Devices around a Tabletop Display; 2006

[13] Hancock, Mark s. et al.; Rotation and Translation Mechanisms for Tabletop Interaction; 2006

[14] Herbig, Andreas; Diploma Thesis: Digital Tabletop Concurrent Input Support Framework; Department of Computer Science, Hochschule Mannheim; 2007

[15] Hoemig, Sascha; Diploma Thesis: Feasibility Study and Prototypical Implementation of a Window Framework for Digital Tables; Department of Computer Science, Hochschule Mannheim; 2007

[16] Hutterer, Peter et al.; Supporting Mixed Presence Groupware in Tabletop Applications; 2006

[17] Igarashi, Takeo et al.; An Architecture for Pen-based Interaction on Electronic Whiteboards; 2000

[18] Igarashi, Takeo et al.; Adaptive Recognition of Implicit Structures in Human-Organized Layouts; 1995

[19] Interactions Lab, University of Calgary; Documentation of the SDGToolkit (Online, http://grouplab.cpsc.ucalgary.ca/cookbook); Viewed March 29th 2007

[20] Sintes, Tony; Event listeners – How do you create a custom event? (Online, http://www.javaworld.com/javaworld/javaqa/2000-08/01-qa-0804-events.html); Viewed October 27th 2006

[21] Liu, Lawrence; Master of Science Thesis: An Environment for Collaborative Agile Planning, Department of Computer Science, University of Calgary; 2006

[22] Liu, Lawrence et al; An Environment for Collaborative Iteration Planning; 2005

[23] Liu, Lawrence et al; Support Agile Project Planning; 2005

[24] Kakehi, Yasuaki et al.; Transparent Tabletop Interface for Multiple Users on Lumisight Table; 2006

[25] Matrox; User Guide for matrox QID Series (Online; http://www.matrox.com/graphics/media/common/user_manuals/qid/qid_en.pdf); Viewed March 6th 2007

[26] Matsuda, Masafumi et al.; Behavioral analysis of asymmetric information sharing on Lumisight Table; 2006

[27] Maurer, Frank; Presentations for class CPSC 601.93 W2007 (Online, http://ebe.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=Courses.CPSC60193W2007.Schedule); March 19th 2007

[28] Mohamed, K.A. et al; Disoriented Pen-Gestures for Identifying User around the tabletop without Cameras and Motion Sensors; 2006

[29] Morgan, Robert at al.; MasePlanner: A Card-Based Distributed Planning Tool for Agile Teams; 2006

[30] Morgan, Robert et al.; Using Horizontal Displays for Distributed & Collocated Agile Planning; 2007 to appear

[31] Morris, Meredith Ringel; Supporting Effective Interaction with Tabletop Groupware; 2006

[32] Morris, Meredith Ringel et al.; Beyond "Social Protcols": Multi-User Coordination Policies for Co-located Groupware; 2004

[33] Mynatt, Elizabeth D. et al.; Flatland: New Dimensions in Office Whiteboards; 1999

[34] Ryall, Kathy et al; Experiences with and Observations of Direct-Touch Tabletops; 2006

[35] Shen, Chia; Multi-User Interactions on Direct-Touch Horizontal Surfaces: Collaborative Tabletop Research at MERL; 2006

[36] Smart Tech; Documentation of Control Panel

[37] Smart Tech; DViT Digital Vision Touch Technology White Paper (Online, http://smarttech.com/dvit/DViT_white_paper.pdf); Viewed October 13th 2006

[38] Tse, Edward et al.; GSI DEMO: Multiuser Gesture/Speech Interaction over Digital Tables by Wrapping Single User Applications; 2006

[39] Tse, Edward et al.; Motivation Multimodal Interaction Around Digital Tabletops; 2006

[40] Tse, Edward et al; Multimodal Multiplayer Tabletop Gaming; 2006

[41] Tse, Edward et al; Enabling Interaction with Single User Applications through Speech and Gesture on a Multi-User Tabletop; 2006

[42] Tse, Edward et al; SDGToolkit: A Toolkit for Rapidly Prototyping Single Display Groupware; 2002

[43] Version One LLC.; Agile Project Management Tools (Online, http://versionone.com/products.asp) ; Viewed January 10th 2007

[44] ViewSonic; Specifications of VP930b (Online, http://www.viewsonic.com/pdf/us_eng/products/VP930b-3_Mar_2007.pdf), Viewed March 6th 2007

[45] Wikimedia Foundation; Graphical Editing Framework (Online, http://en.wikipedia.org/wiki/Graphical_Editing_Framework); Viewed March 20th 2007

[46] Woldrich, David; CardMeeting (Online, http://cardmeeting.com/); Viewed January 12th 2007

[47] Wu, Mike et al.; Gesture Rgistration, Relaxation , and Reuse for Multi-Point Direct-Touch Surfaces; 2006

[48] XPlanner; XPlanner Overview (Online, http://xplanner.org/); Viewed January 12th 2007