A Personal Digital Assistant interface for Immersive Projection Technology visualisation

Master’s Thesis

Einat Robinsson

TRITA-NA-Eyyhn
Abstract

User interfaces (UIs) are crucial components in computer applications. User interfaces consist of UI devices such as mice, keyboards, etc. and Graphical User Interface (GUI) elements such as buttons and menus.

For interaction in Virtual Reality (VR), number of UI devices for 3D and 2D are used. However, there are some difficulties when interacting with 2D GUI elements, such as menus, using 3D interaction devices in the 3D environments.

One of the suggested solutions for that problem is to combine 2D and 3D interaction devices in the 3D environment, is meaning that 2D interaction devices will interact with 2D GUI elements and 3D interaction devices will interact with 3D GUI elements. The 2D GUI elements will still be used in the 3D environment, but interacted with a 2D interaction device, such as a Personal Digital Assistant (PDA), which is more natural to use for interaction with 2D GUI elements.

This thesis has tried to accomplish this solution by using a PDA (3Com’s Palm Pilot Vx) as a 2D interaction device for a 3D environment (CAVE). The system was developed using Java™ 2 Micro Edition (J2ME) and was tested in the VR-Cube located in the Royal Institute of Technology in Stockholm, Sweden.

The developed system is running on a Palm Vx and interacts with a 3D environment using GUI elements that are defined by the running VR application.

Royal
Contents

1 Introduction 3

2 Background 7
   2.1 Virtual Reality (VR) 7
   2.2 Basic Terminology 7
      2.2.1 Immersion 7
      2.2.2 Interaction 8
      2.2.3 Tactile and Haptic Feedback 9
      2.2.4 Acoustic Feedback 10
   2.3 VR Systems 10
      2.3.1 Immersive VR Systems 10
      2.3.2 Non-immersive VR Systems 11
      2.3.3 Hybrid VR Systems 13
   2.4 Interaction Techniques 13
      2.4.1 Navigation 13
      2.4.2 Selection and Manipulation 14
   2.5 The Personal Digital Assistant (PDA) 17
      2.5.1 Introduction 17
   2.6 Previous Work 17
      2.6.1 The Bamboo Project 17
      2.6.2 The JAIVE Project 19
      2.6.3 Virtual Harlem 19
      2.6.4 Peephole Displays: Handheld Devices as Virtual Windows 19
      2.6.5 Tweek 21
      2.6.6 The Palmist 21
      2.6.7 Cellular Phones As Interaction Tools 21

3 General Design 25
   3.1 General 25
   3.2 User Requirements 25
      3.2.1 System Limitations 26
   3.3 Use Cases 26
      3.3.1 General 26
      3.3.2 Use Case 1: Navigate in the 3D world 27
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3 Use Case 2: Interact with the VR application</td>
<td>28</td>
</tr>
<tr>
<td>3.3.4 Use Case 3: Add a GUI item</td>
<td>29</td>
</tr>
<tr>
<td>3.3.5 Use Case 4: Delete a GUI item</td>
<td>31</td>
</tr>
<tr>
<td>3.4 System Requirements</td>
<td>33</td>
</tr>
<tr>
<td>3.4.1 System architecture of the CAVE</td>
<td>33</td>
</tr>
<tr>
<td>3.5 General Design of the system</td>
<td>33</td>
</tr>
<tr>
<td>4 Construction</td>
<td>35</td>
</tr>
<tr>
<td>4.1 General</td>
<td>35</td>
</tr>
<tr>
<td>4.2 Design</td>
<td>35</td>
</tr>
<tr>
<td>4.2.1 System architecture</td>
<td>35</td>
</tr>
<tr>
<td>4.2.2 System Components</td>
<td>36</td>
</tr>
<tr>
<td>4.3 Implementation</td>
<td>36</td>
</tr>
<tr>
<td>4.3.1 The Java™ 2 Platform Micro Edition (J2ME)</td>
<td>36</td>
</tr>
<tr>
<td>4.3.2 The Connected Limited Device Configuration (CLDC)</td>
<td>38</td>
</tr>
<tr>
<td>4.3.3 The Mobile Information Device Profile (MIDP)</td>
<td>39</td>
</tr>
<tr>
<td>4.3.4 Mobile Programming Conclusions</td>
<td>40</td>
</tr>
<tr>
<td>4.3.5 Class Diagram</td>
<td>41</td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>45</td>
</tr>
<tr>
<td>5.1 The Implemented System</td>
<td>45</td>
</tr>
<tr>
<td>5.2 Test</td>
<td>45</td>
</tr>
<tr>
<td>5.2.1 The communication mechanism for the Joystick</td>
<td>45</td>
</tr>
<tr>
<td>5.2.2 The Palm test program</td>
<td>47</td>
</tr>
<tr>
<td>5.2.3 Uncompleted Test</td>
<td>48</td>
</tr>
<tr>
<td>5.3 Future Work</td>
<td>48</td>
</tr>
<tr>
<td>6 Appendix</td>
<td>51</td>
</tr>
<tr>
<td>6.1 Class libraries</td>
<td>51</td>
</tr>
<tr>
<td>6.2 Compilation and Running tools</td>
<td>51</td>
</tr>
<tr>
<td>6.2.1 Compilation</td>
<td>52</td>
</tr>
<tr>
<td>6.2.2 Conversion to a .PRC file</td>
<td>52</td>
</tr>
<tr>
<td>6.2.3 Installation of a .PRC file</td>
<td>53</td>
</tr>
<tr>
<td>6.2.4 Running a .PRC file</td>
<td>53</td>
</tr>
<tr>
<td>6.2.5 Communicating over the Serial Port with Telnet</td>
<td>53</td>
</tr>
<tr>
<td>6.3 Protocol</td>
<td>54</td>
</tr>
<tr>
<td>6.3.1 General</td>
<td>54</td>
</tr>
<tr>
<td>6.3.2 ASCII protocol</td>
<td>54</td>
</tr>
<tr>
<td>6.3.3 Button</td>
<td>54</td>
</tr>
<tr>
<td>6.3.4 Buttons</td>
<td>54</td>
</tr>
<tr>
<td>6.3.5 Menu</td>
<td>54</td>
</tr>
<tr>
<td>6.3.6 MenuItem</td>
<td>55</td>
</tr>
<tr>
<td>6.3.7 Label</td>
<td>55</td>
</tr>
<tr>
<td>6.3.8 Textfield</td>
<td>55</td>
</tr>
</tbody>
</table>
6.3.9 Slider ........................................ 55
6.3.10 Location ................................... 55
6.3.11 Return value .............................. 56
6.3.12 Error codes .............................. 56
6.4 Program Notes .............................. 56

References .......................... 57
Acknowledgements

I would like to thank my husband, Michael and my daughter Noa who supported me the whole way.

Thanks to Kai-Mikael Jää-Aro, my supervisor at PDC, for all the big help in this thesis work.
Chapter 1

Introduction

One of the basic problems in VR is the interaction with a 3D environment.

Interaction with Virtual Reality is done by using number of User Interface (UI) devices for 3D and 2D. However, there are some difficulties when interacting with 2D GUI elements such as menus, using 3D interaction devices, in 3D environments.

An example for that is illustrated in Figure 1.1. It can be seen that the menu shown on the (lower) figure is covering a part of the 3D view. Furthermore, because the environment used is an Immersive Projection Technology (IPT)\textsuperscript{1} environment, such as a CAVE, the menu displayed there, is not static, but floating. That fact makes it difficult to interact with such menus, because they can move anywhere on the space, even on the floor or on the ceiling of the CAVE.

Because 2D GUI elements can be interacted most naturally and conveniently, with 2D interaction devices, one of the suggested solutions for that problem was to combine 2D and 3D interaction devices in the 3D environment, is meaning that 2D interaction devices will interact with 2D GUI elements and 3D interaction devices will interact with 3D GUI elements.

The 2D GUI elements will still be used in the 3D environment, but with a 2D interaction device, such as a Personal Digital Assistant (PDA), which is more natural to use.

This thesis will try to accomplish this solution by using a PDA (3Com’s Palm Pilot Vx) as a 2D interaction device for a 3D environment (CAVE).

This interaction device will serve the VR application running in the Virtual Environment (VE) by providing the VR techniques needed. Furthermore, it will provide an independent GUI for the VR application according to its requirements. This way the VR application will be able to create its own GUI elements on the PDA display by, for example, adding buttons, menus, labels, text fields and sliders.

This thesis work is meant to be used in the VR-Cube environment in the Royal Institute of Technology in Stockholm, Sweden.

The following topics are described in this thesis report:

\textsuperscript{1}An Immersive Projection Technology environment is a general name for CAVE, workbenches etc..
**Figure 1.1.** Interaction in a 3D environment. One of the basic problems in Virtual Reality is interacting with 2D GUI elements using 3D User Interface devices, in a 3D environment. This situation is illustrated in the lower figure, where the 3D environment is interacting with menus (2D GUI element) that cover part of the environment. The upper figure shows a 3D environment that does not interact with 2D GUI elements. Figures courtesy of Kai-Mikael Jää-Aro, Royal Institute of Technology.
• Background. Introduces VR terminology, VR systems and previous work that was done in the field.

• General design. Describes the analysis part including user and system requirements.

• Construction. Describes the design and implementation parts. Includes some technical information about the system and the Java language. This part can be skipped by those who are not interested in the technical part.

• Conclusions. Describes the test part, conclusions and future work in the same field.

• Appendix. Some additional technical documentation for running and using the system.
Chapter 2

Background

2.1 Virtual Reality (VR)

Virtual Reality is a broad term which was described differently by a large number of publications, TV shows and conferences. This led Burdea Grigore and Coiffet Philippe in their book, *Virtual Reality Technology* [2] to define the term Virtual Reality (VR) that according to them, until then was confusing:

Virtual Reality is a high-end user interface that involves real-time simulation and interaction through multiple sensorial channels. These sensorial modalities are visual, auditory, tactile, smell, taste, etc.

VR in terms of functionality is a simulation in which computer graphics are used to create a realistic looking world. Furthermore, this world responds to the user inputs instantly, in the form of gestures, verbal commands, etc. This immediately response defines the real-time\(^1\) interaction.

Interactivity contributes to the feeling of Immersion, which means, being part of the action on the screen, that the user experiences. Furthermore, VR can use all human sensorial channels, in sense that users cannot just see and manipulate graphic objects on the screen, they can also touch and feel them [2].

2.2 Basic Terminology

2.2.1 Immersion

*Immersion* is the feeling of being part of the action on the screen. The sensation of immersion within the Virtual Environment (VE), is influenced by the user integration with the virtual world.

The VR system used provides the sensation of immersion in the virtual world for example there are VR systems which provide the user with a view of the virtual

\(^1\) Real-time means that the computer is able to detect user inputs and modify the virtual world immediately.
world using a Head Mounted Display (HMD)\(^2\), as shown in Figure 2.1 and give the opportunity to feel the virtual world also with touch (tactile feedback) and sound (acoustic feedback) [25].

2.2.2 Interaction

John Vince in his book *Virtual Reality Systems* [25] explains that when the user is immersed within the virtual world, there is a natural requirement to interact with the virtual objects in it, such as to touch them. Obviously this is impossible, as there is nothing to touch. The user's sense of immersion is increased, when including a part of a 'virtual body' such as a hand, in the virtual world.

The interaction occurs in such a way that when moving for example, the head (equipped with for example, an HMD), the computer responds with the new views of the virtual world. When stretching out the hand (equipped with an interaction glove, for example) towards a virtual object, the virtual hand in the scene will move in the same direction [25].

**VR Interaction Devices**

The fact that VR interaction devices are addressing several human sensorial channels, makes them different from each other in functionality and purpose. Different

\(^2\)A device which supplies the user's eyes with a view of the virtual world [25]. This device consists of a screen which shows different pictures of the 3D world for the right and left eyes, in order to simulate the 3D world.
tracks are needed for body, hand and head tracking.

For navigation in the VR world, one needs to receive information about the real-time position and orientation of moving objects. Moving objects in 3D have three translations as well as three rotations called "yaw", "pitch" and "roll", which define a data set of six numbers that need to be measured as fast as the object moves [2]. For that purpose, tools such as trackballs were developed.

For the visual feedback, tools such as Head Mounted Display (HMD) and stereo glasses were developed.

For the tactile and force feedback, VR interaction devices such as a joystick is available.

2.2.3 Tactile and Haptic Feedback

Tactile feedback is a touch feedback and haptic feedback is a force feedback. When the hand is touching an object, the fingers get to feel that object, and when pushing on the same object, the hand muscles will be activated. However, providing the same level of tactile feedback as in the real world is currently impossible, and may remain so, unless a technology that communicates with the brain is be developed [25].

In the meantime, to provide the tactile feedback, tactile gloves which activate small pads along the fingers are available. Touch sensors used for tactile feedback, give information on contact-surface geometry such as its smoothness, temperature or slippage due to gravity.

If it is necessary to transmit forces from the virtual domain, then it is needed to involve some hardware that the user can grasp such as force effectors. Force sensors, give information on the total contact force, on contact surface compliance or grasped object weight [2].

Force feedback is used for example on the flight controls in a flight simulator so that the pilot experiences the real physical forces that act on real planes. [25].
2.2.4 Acoustic Feedback

Hearing supplies a strong feeling of immersion in VR. Some VR systems use sound in order to complement the interaction with the visual and tactile domains. For example, when watching waves breaking on the shore, the sound of it is also heard in order to strengthen the feeling of standing on a beach. The sound has also to come from the same location as the event [25].

2.3 VR Systems

Virtual Reality systems are the devices used in VR for presenting the 3D environment in order to be able to interact with it. However, VR systems can be divided into three VR system groups: immersive, non-immersive and hybrid [25].

2.3.1 Immersive VR Systems

An immersive VR system replaces the user’s view of the real world with computer-generated images that react to the position and orientation of the user’s head [25]. By that, it gives the feeling of being a part of the virtual world and therefore to be able, to hear and see in the virtual environment.

Such an environment is the Cave Automatic Virtual Environment (CAVE) which is a full immersive environment (will be explained in detail the next section).

The interaction in these environments is done usually with the help of, for example, a Head Mounted Display (HMD) which supplies the user’s eyes with the view of the computer generated world, and a 3D hand tracker such as a 3D glove (shown in Figure 2.2) or joystick.

The Cave Automatic Virtual Environment (CAVE)

The first CAVE was introduced at SIGGRAPH 92 (see Figure 2.4), by the Electronic Visualisation Laboratory at the University of Illinois at Chicago. This CAVE was a three sided room with three rear projection screens for walls (3 × 2.75 meters) and a down-projection screen for the floor. High-resolution (1280 × 512) video projectors displayed stereo images and computer-controlled amplifiers relayed sampled sound through a network of speakers [25].

For interaction with a CAVE, tracking sensors are attached to the user’s head or to a visual device such as shutter glasses (shown in Figure 2.5), as well as to a pointing device or a glove, to be held by or worn on the hand. In this way, the projected images can be updated in real time according to changes in the user’s position and viewing angle [20].

The CAVE is large enough to let viewers watch the world. It is fully immersive, and the users have the possibility to feel and explore the environment while receiving a real-time visual response [25].

The disadvantage in that CAVE was that the down-projector for the floor caused a shadow to appear on the floor.
This problem was taken into consideration when the Royal Institute of Technology in Stockholm, Sweden, built its CAVE, 1998 (shown in Figure 2.3). In order to avoid that problem, the CAVE was built with six walls (including ceiling and floor) instead (compare to three walls in the old version), and uses six projectors (one projector to every wall). This CAVE is a $3 \times 3 \times 2.5$ meters environment [21].

### 2.3.2 Non-immersive VR Systems

Non-immersive VR systems provide the user with the ability to watch the 3D world but without the feeling of being part of it. The user observes the 3D world through some display device such as a graphic workstation and interacts with it using devices such as a Space Mouse. A non-immersive VR system is shown in Figure 2.6 and a Space Mouse is shown in Figure 2.7.
Figure 2.5. Shutter glasses used in the CAVE. Picture is courtesy of SARA.

Figure 2.6. Non-immersive VR System. The user observes the 3D world through a workstation and interacts it using a geoball. Picture courtesy of Kai-Mikael Jää-Aro, Royal Institute of Technology.
2.3.3 Hybrid VR Systems

Hybrid VR systems allow the user to observe the real world with virtual images superimposed over this view, also known as 'augmented reality' systems.

2.4 Interaction Techniques

There is a difference between input devices, mentioned earlier in VR Interaction devices and interaction techniques. Input devices are the VR tools which are used to implement various interaction techniques.

The interaction tasks in the VE can be divided into: navigation, selection and manipulation.

This section summarizes the main ideas mentioned in the article An Introduction to 3-D User Interface Design [1].

2.4.1 Navigation

Bauman, Kruijff, LaViola and Poupyrev [1], describe Navigation as the following:

The task of navigation is the most prevalent user action in most large-scale 3D environments, and it presents challenges such as supporting spatial awareness, providing efficient and comfortable movement between distant locations, and making navigation lightweight so that users can focus on more important tasks.

They classify navigation tasks into three categories:

- **Exploration.** Navigation with no explicit target: the user just investigates the environment.
- **Search.** Moving to a particular target location.
• **Maneuvering.** Characterized by short-range, high-precision movements that are used to place the viewpoint at a more advantageous location for performing a particular task.

Navigation can be divided into a motor component called *travel*, which is defined as the movement of the viewpoint from one location to another, and a cognitive component called *wayfinding*, which is defined as the cognitive process of defining a path through an environment.

Five common metaphors for travel interaction techniques are exist:

- Physical movement. The use of the user’s body motion to travel through the environment.
- Manual viewpoint manipulation. The use of the user’s hand motions to effect travel.
- Steering. It is the specification of the direction of motions and includes techniques such as gaze-directed steering (determined by the user’s head orientation) or pointing (used by hand orientation).
- Target-based travel. The system handles the actual movement according to the user’s destination specifications.
- Route planning. The system handles the actual movement according to the user’s path (that should be taken through the environment) specifications.

In Wayfinding, the cognitive map of an environment, is built up by using spatial knowledge, which consists of landmark, procedural and survey knowledge.

Furtherhead trackbe considered

Simulation of Navigation can be accomplished, for example with a help of a joystick device that does it by tracking the *relative* coordinates.

### 2.4.2 Selection and Manipulation

*Object selection, object positioning and object rotation* are three basic tasks, where at least one of them should be accomplished by interaction techniques for 3D manipulation.

There are two classical approaches for designing selection and manipulation techniques:

- To provide the user with a “Virtual” hand (a 3D cursor, often shaped as a human hand), whose movements correspond to the movements of the hand tracker. Selection and manipulation involve touching an object, then positioning and orienting this virtual hand within the VE. This virtual hand technique

---

3Relative coordinates are the position coordinates, normalised to the range of $[-1, 1]$ 2D coordinates of the position chosen by the user. The 2D coordinates (x and y) are then interpreted to 3D coordinates in the virtual world.
is intuitive because it simulates a real-world interaction with objects. Only those objects within the area of reach can be picked up.

- To point at objects in the VE using virtual ray casting, (as shown in Figure 2.8) from the virtual hand. When the virtual ray intersects an object, it can be picked up and manipulated.

**System Control**

System Control refers to a task in which a command is applied to change the state of the system or the mode of interaction. Issuing a command includes the selection of an object from a set. Therefore some similarities can be seen between system control and object selection techniques.

In desktop applications, the use of commands is very common, unfortunately, the use of commands via 2D interaction styles such as pulldown menus and command-line input, are not always usable within a VE.

One of the problems of VE system control is that normally one- or two-dimensional task becomes three-dimensional, which reduces the effectiveness of traditional techniques. For example, touching a floating menu item in space is much more difficult than selecting a menu item on the desktop, not only because the task has become 3-D, but also because the important limitation of the physical desk on which the mouse rests is missing.

This situation is illustrated in Figure 1.1, where the VE is shown with and without the 2D menu. When the 2D menu is shown in the virtual environment, it is not static, but is moving in the VE (can be seen on one of the walls, on the floor or on the ceiling), which makes it difficult to capture it for selection of a menu item.
Two-Dimensional interaction in the Three-Dimensional Environments

As mentioned before, the two dimensional interaction, such as 2D menus, in the 3D world is one of the most basic problems known today in the VE.

One of the suggested solutions for that was to use only 3D interaction for the 3D world, because the 3D applications usually contain 3D worlds in which users can create, select and manipulate 3D objects. The problem with that solution is, that in reality, 2D interaction offers a number of advantages over the 3D interaction techniques for certain tasks.

As mentioned before, most efficient selection techniques are essentially 2D, although manipulation may require 3D interaction techniques.

The suggested solution is then to take advantage of both the 2D and 3D interaction techniques in order to get the maximum benefits of them. This way the interfaces for 3D applications will be easier to use and more intuitive for the user. This can be done by creating interfaces for the 3D application that integrate the two techniques.

The combined 2D/3D interfaces can be classified into three categories where all of them have a physical surface but they differ in how the 2D physical surfaces are utilized.

1. Applications which use fully immersive displays—these applications use for example an HMD, where the user is unable to see physically the 2D surface, which is usually a piece of tracked plastic or pen-based tablet. The user then must be supplied with a graphical representation of the surface in order to interact with it. An example of such tools is Virtual Notepad which is used for writing and annotating in VR [22].

2. Applications which use semi-immersive displays—these applications use for example workbenches as display. The physical 2D interaction surface is either on top of the workbench display so that users can interact directly with the display, or on a tracked, transparent tablet that users can hold in their hands, and see the graphics virtually on the surface of the tablet. Example for this is the ErgoDesk system [4], which is a modeling application using the physical display for a 2D interface.

3. Applications which use a separate 2D display surface—these applications use for example a handheld computer and pen-based LCD (Liquid Crystal Display) tablets. This thesis is an example of this type of interface, using a Palm Pilot in a CAVE, for environment and geometry controls. The system control interface is shown on the Palm Pilot display and effects on selected objects in the VR application.

---

4 Workbench is a workspace where one can interact with a 3D world. A Responsive Workbench (see Figure 2.9) is an example of that.
Figure 2.9. Responsive Workbench. This workbench is a 3D interactive workspace, where computer generated images are projected onto it and viewed through shutter glasses. Picture courtesy of Stanford University.

2.5 The Personal Digital Assistant (PDA)

2.5.1 Introduction

The main vision for developing a PDA (shown in Figure 2.10) was to implement a computer that can conveniently be stored in a pocket (of sufficient size) and used while one is holding it [10].

Two of the main goals when developing a PDA were to be of small size (compared to a desktop) and fast. In order to achieve these goals, one had first to eliminate all superfluous services, features and capabilities and second, to move all heavy-duty processing to a desktop computer, led to two disadvantages regarding the PDA [24].:

1. It has a limited memory area (about 500k ROM), which is problematic if one need to download big applications.

2. It has a small screen size which requires a small size of GUI (Graphical User Interface) or the use of scrollbars.

These two disadvantages have been brought into consideration in this thesis work and it will be explained later how.

The PDA used in this thesis work, the Palm Pilot Vx, which has serial and Infra Red (IR) ports, where only the serial port was used here. New versions of PDAs provide also wireless communication.

2.6 Previous Work

2.6.1 The Bamboo Project

The idea of using a PDA as an interaction device in a VE, is not new. It was first introduced by Watsen, Darken and Capps [26] with the Bamboo Project, 1999, in
Figure 2.10. A Personal Digital Assistant (PDA) device, Palm Pilot Vx. Picture courtesy of Panwebi Electronics.
order to solve the earlier mentioned problem of the interaction tools in 2-D and 3-D.

The Bamboo project (see Figure 2.11) offers the user an interface which includes camera, environment and geometry functionalities. Every functionality was implemented as an applet, and there is a possibility to download some new applets as well. Bamboo interacts with the CAVE-like environment via the PDA’s serial port.

After that research, a lot of projects started to follow up this technique, some of them are listed below. This thesis is following up the same technique but improves the GUI’s independance, where a VR application is able to define its own graphical interface on the PDA display.

2.6.2 The JAIVE Project

The JAIVE (Java based Interface to the Virtual Environment) project has also developed an interaction tool which provided the user with integration of common interaction methods such as selecting colors, performing push-button operations etc. into their IPT applications [8]. JAIVE is shown in Figure 2.12.

2.6.3 Virtual Harlem

This project implemented a virtual reconstruction of Harlem in the 1920s, where the interaction was done by a PDA. This system is used for education and includes a number of capabilities in VR such as audio, playback, avatars, navigation, controls and triggers that detect when a user enters an area [19]. The use in Virtual Harlem is shown in Figure 2.13.
Figure 2.12. The Java based Interface to the Virtual Environment (JAIVE) Project [8].
2.6.4 Peephole Displays: Handheld Devices as Virtual Windows

The Peephole Displays project has developed the handheld device as a small virtual window on a large workspace, where one has to move the handheld in order to see different parts of the workspace. This project (see Figure 2.14) proposes a new scrolling mechanism by moving the whole handheld instead of using a scrollbar for that [11].

2.6.5 Tweek

Tweek is a Java-based middleware tool which can be used by developers in order to create a GUI that provides all the capabilities users need for VE interaction. This tool enables the control of a VR application and has been implemented to be general so that it gives programmers the freedom to design the required GUI in their VE interaction design [6]. Tweek can run on desktop computers or palmtop computers in projection-based virtual reality systems [7].

2.6.6 The Palmist

The Palmist is a computer, whose capabilities can be expanded and achieve functionality by a software framework. It offers the following features for the user:

- wireless mobility for the user
- better presentation of data like values and long texts.

The Palmist is the interface device for the ARSBox which is a projection-based (Cave-like), PC-based VR system [9]. The Palmist is shown in Figure 2.15.

2.6.7 Cellular Phones As Interaction Tools

This sketch which was presented by Larsen, Bærentzen and Christensen 2.16, uses cellular phones to interact with the VE, the same reason this thesis is using the
Figure 2.14. Peephole Displays. Scrolling large workspaces with a help of a handheld device [11].
Palm. Cellular phones don’t have hardware support for 3D graphics, the screen is much smaller (176×208 or 208×320 pixels) and the only input is a limited number of pushbuttons. The aim of this sketch was to enable users to build LEGO™ structures in collaboration using either a traditional workstation or a cellular phone as shown in Figure 2.16. This project was implemented in J2ME-Java 2 Micro-Edition.
Figure 2.16. Cellular Phones As Interaction Tools: The upper figure shows the LEGO™ structure as it is seen on the cellular phone, and the lower figure shows how the same structure is seen on the workstation. Pictures courtesy of the Technical University of Denmark [3].
Chapter 3

General Design

3.1 General

The modeling notation used in this thesis work, is the Unified Modeling Language (UML) [23]. This method was used for the different parts of the application development — analysis, design, implementation and test.

This chapter describes the analysis part of the system – from the user requirements to a general description of the system’s different platforms, which effect the design and implementation later.

3.2 User Requirements

The system should implement a graphical user interface on a PDA-device for interaction in the CAVE-like environment, the VR-Cube, at the Royal Institute of Technology in Stockholm, Sweden. In order that the user would be able to interact the CAVE environment using the PDA-device, the following requirements have to be accomplished:

- The user should be able to navigate in the 3D world presented by the VR application in the VR-Cube.
- The user should be able to select items in the 3D world.
- The user should be able to manipulate selected items in the 3D world.
- The VR application interacting with the PDA device should be able to define its own graphical user interface. The GUI in the system will be portable and flexible, in a sense that the user (in that case, the developer of the VR application) will be able to define the required GUI for the wanted VR application. This means that, the GUI generated on the PDA device will be different for different VR applications.
- The interaction with the VR application and the user should be easy, friendly and fast (real-time environment requires high performance).
• The generated GUI will be familiar to the user, with menus, push-buttons, textfields, labels and sliders.

3.2.1 System Limitations

The fact that the system will work on a PDA device, which is small and has limited resources, requires some limitations for the user requirements.

• The system will provide only primitive GUI features, supported by the PDA device. That means that no advanced GUI features such as special sliders can be supported. Primitive GUI features are for example push-buttons, menus, textfields and labels.

• The system uses a serial connection to communicate with the PDA device. The wireless communication isn’t used because there is no IR connection resources in the VR-Cube, that is necessery for wireless communication.

3.3 Use Cases

3.3.1 General

The user requirements mentioned earlier will be divided into number of use cases. A Use Case Overview is a view which captures the behavior of a system, subsystem or class as it appears to a user. It partitions the system functionality into transactions called Use Cases which are meaningful to actors-usually the users of the system.

A Use Case describes an interaction with actors as a sequence of messages between the system and one or more actors.

Actors can be humans, other computer systems and processes [23].

The use case overview diagram is shown in Figure 3.1 and it can be seen that the user requirements were divided into four use cases.

The main requirements, to perform the navigation, selection and manipulation tasks, will be adapted to the PDA device’s features. In order to provide navigation, the system will simulate joystick movements with the PDA. The joystick can move in two dimensions, which are interpreted as acceleration forward/back and rotation right/left in a 3D environment. The user will use the PDA stylus to indicate the direction of the movements in two dimensions.

The selection/manipulation will be accomplished by the VR application and through the GUI defined by it.

The navigation technique will be implemented in use case 1: Navigate in the 3D world and the selection and manipulation techniques will be accessed via use case 2: Interact with the VR application.

The selection and manipulation techniques will be implemented by the VR application itself and will be interacted with the generated GUI. This GUI is defined by adding and deleting GUI elements to the PDA display according to the VR application’s requirements, and is implemented in use case 3: Add a GUI item and use case 4: Delete a GUI item.
The system has two groups of users:

- users of the VR application, which use the PDA device as an interaction tool in the VR-Cube.

- developers of the VR application, which define the GUI according to the VR application’s requirements.

### 3.3.2 Use Case 1: Navigate in the 3D world

The system provides the user with the possibility to navigate in the 3D environment using the PDA stylus. Once the PDA stylus is moved by the user on the PDA display (two dimensional environment), the $x$ and $y$ coordinates of the marked positions on the PDA, are generated\(^1\) by the system and are sent forward to the VR application. The VR application then, interprets these 2D coordinates as acceleration.

The user in this use case is the user of the VR application. The interaction between this use case, other use cases and the actors is shown in Figure 3.2.

---

\(^1\)The position $(x, y)$ is normalised to be in the range of $[-1, 1]$
**The Process Workflow**

This use case will be accomplished by the following steps:

1. The system is started when the user taps the system's icon on the PDA display.

2. The user starts to navigate by moving the stylus on the PDA display.

3. The system sends the x and y values (floating point format) of the current position to the corresponding VR application via the serial connection.

4. The VR application interprets the 2D coordinates as 3D coordinates. The control of the process is passed to the VR-application, which is responsible also to show the user the required position on the 3D world.

5. The user can then choose to repeat the following steps:
   - continue navigating by using the stylus on the display, and the process continues in point 3
   - stop navigating by pulling the stylus out of the display, and the process continues in point 6

6. The system finishes the work flow process, by sending the initial position (0,0) to the VR application.

**Alternative process, following an error in serial connection**

1. This process starts in point 3, when the system cannot be connected via the serial port.

2. The system shows an error message to the user as a dialog box with an ok button.

3. The user chooses ok.

4. The user can choose one of the following:
   - try to continue navigating, and the process continues in point 3 in the work flow process.
   - Stop navigating, and the process continues in point 6 in the work flow process.

**3.3.3 Use Case 2: Interact with the VR application**

The system provides the user with the possibility to interact with the VR application via the defined GUI.

The user of this use case is the user of the VR application. The interaction between the use case and the actors is shown in Figure 3.3.
Use Case 2: Interact with the VR application. Use case diagram that shows the relationships between this use case and the actors of the system.

**The Process Workflow**

This use case will be accomplished by the following steps:

1. The user chooses to perform an action by interacting with the GUI.
2. The system sends an event which consists of the definition (name) of the GUI element to the VR application, via a serial connection.
3. The control of the process is passed to the VR application, that is responsible for the implementation of the functionality.
4. The system finishes the work flow process.

**Alternative process, following an error in serial connection**

1. This process starts in point 2, when the serial connection can not be established.
2. The system shows an error message to the user, as a dialog box with an ok button
3. The user chooses ok.
4. The user can choose one of the following:
   - try to continue interacting the GUI, and the process continues in point 1 in the work flow process.
   - Stop interacting with the GUI, and the process continues in point 4 in the work flow process.

**3.3.4 Use Case 3: Add a GUI item**

The system provides the user with the possibility to add GUI items such as menus, menuitems, push-buttons, sliders, text-fields and labels to the PDA-interface according to the VR application’s requirements.
**UseCase3:** Add a GUI item

The appearance and functionality of the GUI item will be defined by the VR application and will be used by the it in the VR-Cube for selection and/or manipulation.

The user of this use case is the VR application.

The interaction between this use case and the actors is shown in Figure 3.4 below.

*Figure 3.4.* UseCase3: Add a GUI item. Use case diagram that shows the relationships between this use case and the actors of the system.

**The Process Workflow**

This use case will be accomplished by the following steps:

1. The system is started when the user taps the system’s icon on the PDA display.
2. The VR application defines one or more required GUI item to be added (according to a specified protocol contained in the Appendix).
3. The VR application sends the definition in a message, forward to the system via a serial connection.
4. The system validates the required definition.
5. The system sends an ok signal back to the VR application.
6. The VR application can then choose to repeat the following steps:
   - adding more items, and the process continues in point 2.
   - deleting items, and the process continues in use case 4: Delete a GUI item.
7. The system finishes the work flow process.

**Alternative process, following an error in validation**

1. This process starts in point 4, when the system discovers an error in the sent message.
2. The system shows an error message to the user, as a dialog box with an ok button.
3. The user chooses ok.
4. The user chooses ok.
5. The system continues in point 2 in the work flow process.

**Alternative process, following an error in serial connection**

1. This process starts in point 3, when the serial connection cannot be established.
2. The system shows an error message to the user, as a dialog box with an ok button.
3. The user chooses ok.
4. The user chooses ok.
5. The user can choose one of the following:
   - try to continue adding the GUI, and the process continues in point 2 in the work flow process.
   - Stop adding the GUI, and the process continues in point 7 in the work flow process.

### 3.3.5 Use Case 4: Delete a GUI item

The system provides the user with the possibility to delete existing GUI items such as menus, menu items, push-buttons, sliders, text fields and labels from the PDA display.

The user of this use case is the VR application. The interaction between this use case and the actors is shown in Figure 3.5 below.

**The Process Workflow**

This use case will be accomplished by the following steps:
1. The VR application defines one or more required GUI items to be deleted (according to a specified protocol contained in the Appendix).

2. The VR application sends forward the definition in a message to the system via a serial connection.

3. The system validates the required definition.

4. The system sends an ok signal back to the VR application.

5. The VR application can then choose to repeat the following steps:
   - deleting more items, and the process continues in point 1.
   - adding new items, and the process continues in use case 3: Add a GUI item.

6. The system finishes the work flow process.

**Alternative process, following an error in validation**

1. This process starts in point 3, when the system discovers an error in the sent message.

2. The system shows an error message to the user as a dialog box with an ok button.

3. The user chooses ok.

4. The system continues in point 1 in the work flow process.

**Alternative process, following an error in serial connection**

1. This process starts in point 2, when the serial connection can not be established.

2. The system shows an error message to the user, as a dialog box with an ok button.

3. The user chooses ok.

4. The user can choose one of the following:
   - try to continue deleting the GUI, and the process continues in point 1 in the work flow process.
   - Stop deleting the GUI, and the process continues in point 6 in the work flow process.

         functionality or the appearance of an existed GUI item such as menus, menu-items, push-buttons, sliders, text-fields and labels to the PDA. The user of this use case is the developer of the VR-application.
3.4 System Requirements

The system was developed on Windows NT and tested on 3Com’s PalmPilot Vx, but is meant to work on every handheld computer. The communication to and from the PDA is done via a serial port through a USB adapter.

3.4.1 System architecture of the CAVE

The VR-Cube, at the Royal Institute of Technology in Stockholm, Sweden has a special architecture which has to be taken into consideration when developing a new system which is meant to interact with it.

The VR-Cube system is driven by a two-rack SGI Onyx2 computer called Boye, which has 12 R10000 CPUs, 4GB memory and 100GB disk and three InfiniteReality graphics pipes with two 64MB raster managers each [21].

Snipe is the computer which interprets the joystick (interaction device used in the VR-Cube) signals sent via an USB connection to a TCP/IP connection that Boye is familiar with. Snipe is simply configured with the communication between the VR-Cube and the joystick-device, where every signal that is sent from the joystick-device is sent forward to Boye, as shown in Figure 3.6.

The joystick device is a read-only device and therefore the dataflow is going in one direction, i.e. from the Joystick to Snipe and Boye.

3.5 General Design of the system

The system is a 3 Tier (T) client/server application where the client program is running on Boye and the server program is running on the PDA device. 3T system means that the system can be divided into three different layers where every layer is running on a different device. For example Boye, Snipe and PDA in our case, where the client program is running on Boye and defines its GUI items to be added to the PDA, where the server program is running (as shown in Figure 3.7).
Figure 3.7. System architecture. The communication between Boye, Snipe and the Palm.

The PDA device is a read-write device and therefore the dataflow will go in two directions, i.e. from the PDA to Snipe and Boye when an action is performed, and to the PDA from Boye when the GUI is defined for a specific application.
Chapter 4

Construction

4.1 General

This chapter describes the design and implementation parts of the thesis work. The system was designed using the UML method [23], as mentioned before, and was implemented using Java™ language and its special platform for the mobile devices, Java™ 2 Platform Micro Edition (J2ME).

The system is Object Oriented (OO).

4.2 Design

4.2.1 System architecture

The system architecture for the thesis work is influenced by the existing system architecture in the VR-Cube, as shown before in Figure 3.6. The system is therefore a 3 Tier (T) Client/Server application, meaning that the system is running on three different environments, and has the ability to communicate between those environments. This technique makes the system more secure, scalable, faster and flexible.

In this way, different devices can be used, according to the program requirements. For example, the client program consists of the UI towards the user– which has to be friendly, efficient and fast. The server program, on the other hand, is used usually for communication with a database.

In our case, the system is running on three different devices, as illustrated in Figure 3.7:

- PDA. The server program, where it generates the User Interface required by the client (VR application).

- Cockoo (used instead of Snipe) where the communication between the PDA and Boye is running. Both client and server.

- Boye. The client, where the VR application (client program) is running.
Figure 4.1. System architecture. The communication between Boye, Snipe and the Palm with the corresponding files.

Figure 4.1 illustrates the system running on the different devices, with different implementations.

4.2.2 System Components

The system is object oriented and is therefore divided into a number of classes, where every class is responsible for the definition of a certain object. Generally, the system is divided into three “components”, where every component corresponds to a different tier in the 3T architecture.

1. Interface component—responsible for the interface toward the user and communication between the user and the program.

2. Communication component—responsible for the communication with the serial port, and handling (send/receive) of the message.

3. Business component—responsible for the implementation of the logic in the usecases.

4.3 Implementation

4.3.1 The Java™ 2 Platform Micro Edition (J2ME)

The program running on the PDA was implemented in Java, in order for that to be working both on WindowsNT and UNIX. The software used here is the Sun’s Java 2 platform Micro Edition (J2ME) which is a very small java application environment.

This software is a framework for the deployment and use of Java technology for small devices, which include for example PDAs and cellular phones [14]. In order to support a flexible deployment, the J2ME architecture is modular and scalable by providing virtual machine technologies optimized for the different processor types and memory constraints.
For resource-limited products, J2ME™ supports minimal configurations for the Java virtual machine and Java Application Programming Interfaces (APIs) that supply only the essential features of each kind of device. These minimal configurations can be expanded with additional APIs or a wider implementation of Java virtual machine. In order to support that expanding, two concepts are defined by J2ME:

1. **Configuration.** A configuration defines a minimum platform for a group of devices with similar requirements on total memory and processing power. It consists of a virtual machine, and a minimal set of core libraries and APIs.

2. **Profile.** A profile is layered on top of a configuration and extends it with the specific demands of a certain device family, such as the mobile device family. The main goal of it is to define a standard Java platform for a certain device family or domain and therefore include more domain-specific class libraries than the class libraries provided in a configuration [17]. Currently, there is only one profile defined, the Mobile Information Device Profile (MIDP) which is designed for PDAs, cellphones and related devices.

**J2ME™ Configurations**

In J2ME, an application is written for a particular profile and a profile extends a particular configuration. J2ME can support more than one configuration where each configuration specifies the Java virtual machine features and a set of APIs used by the profile implementers and others.

A configuration is simply a *contract* between a profile implementer and a device’s Java virtual machine, where the virtual machines of the devices agree to implement all the features defined in the configuration.

Currently, there are two J2ME configurations: the Connected Limited Device (CLDC) and the Connected Device Configuration (CDC).

1. **CLDC** is designed for small devices with constrained CPU and memory resources. Those devices run on either a 16- or 32-bit CPU and have 512 Kbytes or less memory available for the Java platform and applications. Such devices are PDAs and cellular phones.

2. **CDC** is designed for devices with more robust resources, which run on a 32-bit CPU and have 2 Mbytes or more memory available for the Java platform and applications. Such devices are TV settop boxes, Internet TVs, Internet-enabled screenphones, high-end communicators, and automobile entertainment/navigation systems [17].

The majority of functionality in CLDC and CDC has been inherited from Java Standard Edition (J2SE), where each class inherited must be exactly the same or a subset of the corresponding class in J2SE. Furthermore, CLDC and CDC may introduce new features designed specifically to fit the small devices, which are not inherited from J2SE.
Configurations and Java virtual machines are very closely related in the sense that making small differences in the configuration’s specification will cause a large number of modifications in the design of the virtual machine.

**The K-Virtual Machine (KVM) Technology**

The KVM is a compact, portable Java virtual machine designed especially for small, resource-limited devices. The goal for the KVM technology was to create the smallest possible Java virtual machine that would preserve all the central parts of the Java programming language, and would be able to run in a resource-constrained device with a few hundred kilobytes total memory.

The KVM is designed to be:

1. small, with a static memory in the range of 40 kilobytes to 80 kilobytes.
2. clean, commented and portable.
3. modular and customizable
4. as complete and fast as possible.

The “K” in the KVM stands for “kilo”, because its memory budget is measured in kilobytes. KVM is suitable for 16 or 32-bit RISC/CISC microprocessors with a total memory budget of a few hundred kilobytes. The minimum total memory required by the KVM implementation is about 128 kB, including the virtual machine, the Java class libraries and some heap space for running Java applications.

The role of the KVM varies for different devices. In some implementations, the KVM is used on top on an existing software in order to give the device the ability to download and run Java content. In other implementations, the KVM technology is used at the lower level in order to be able to implement low-level system software and application of the device in the Java programming language.

Currently, the KVM and CLDC technologies are closely related in the sense that CLDC only runs on top of KVM, and CLDC is the only configuration supported by the KVM [17].

**4.3.2 The Connected Limited Device Configuration (CLDC)**

CLDC is the specification for a class of Java virtual machines that can run on the devices targeted by CLDC and support the profiles which extend the CLDC. The KVM is the implementation of a Java virtual machine that satisfies the CLDC specifications.

The goals of the CLDC are to define a standard Java platform for small and resource-limited devices, to allow dynamic delivery of Java applications and to enable an easy 3rd party application development.

The CLDC requirements are:

1. To run on various small devices with limited resource such as cellphones and PDAs.
2. To make minimal assumptions about the native system software available in CLDC devices.

3. To define a minimum complement of Java™ technology which will be applicant to a various of mobile devices.

4. To provide portability of profile-level code between the kinds of mobile devices.

The CLDC implementation should fit in less than 128 kilobytes and CLDC specification assumes that applications can run in 32 kilobytes of Java™ heap space.

**CLDC Scope**

The CLDC configuration does not support user interfaces. That means that in order to implement a GUI in the PDA (according to the user requirement), one has to find another resource that is compatible with CLDC.

**Limitations of Java™ language and Java™ Virtual Machine in CLDC**

CLDC supports the Java™ language and Java™ Virtual Machine specifications as possible within the strict memory constraints of the small devices. Therefore, some constraints must be performed in order to achieve the minimal requirements. One of the Java™ features that are not supported in CLDC technology is the floating point operation that is needed in our system according to use case 1: navigate in the 3D world.

**Classfile Verification**

CLDC requires that the Java virtual machine will be able to identify and reject invalid classfiles. However, CLDC defines an alternative mechanism to the standard classfile verification defined by J2SE, because it is too memory-consuming for small devices. In this alternative, each method in a Java classfile contains a “stackmap” attribute which is newly defined in CLDC and is added to standard classfiles by a “pre-verification” tool that analyzes each method in the classfile. Pre-verification is performed on a server or desktop system before the classfile is downloaded to the device.

This attribute increases the size of the classfile by approximately 5 % and makes the verification process much faster than in the standard J2SE.

For further information about CLDC, the reader is referred to *Connected, Limited Device Configuration, Specification Version 1.0a* [16].

**4.3.3 The Mobile Information Device Profile (MIDP)**

MIDP, is a set of Java™ APIs which, together with the CLDC provides a complete (because they complete each other) J2ME™ application runtime environment targeted at mobile information devices, such as cellphones and PDAs [15].
MIDP technology consists of API features that do not exist in CLDC, such as user interfaces, which is suitable for the use of our system, but on the other hand it does not support serial communication which is necessary here.

MIDP was produced by the Mobile Information Device Profile Expert Group (MIDPEG) that consists of number of companies such as Ericsson, America Online and Palm that develop and deliver small devices with limited resources.

The main goal of the MIDPEG was to establish an open, third-party application development environment for Mobile Information Devices (MID).

For further information about MIDP, the reader is referred to *Mobile Information Device Profile, Specification 1.0a* [18].

### 4.3.4 Mobile Programming Conclusions

The programming platform used in this system is J2ME including only the CLDC and KVM techniques. MIDP was found unsuitable because of the no support for serial communication.

In order to implement user interfaces which are not supported in CLDC, the KAWT library was used.

In order to support floating point operations for the navigation in the 3D world, the MathFP library was used.

**kAWT**

As mentioned earlier, CLDC and KVM don’t support the user interface functionality and features supported in the J2SE. Therefore, in order to implement a GUI on the Palm display, the use of a UI package was required and kAWT was chosen.

kAWT [12] is an implementation of a class library which provides a simplified version of Abstract Window Toolkit (AWT) for the KVM. The AWT is the Java UI standard class library.

The user interface functionality was provided earlier by the com.sun.kjava classes in J2ME CLDC Beta1 release, but moved to the MIDP instead. The MIDP was evaluated, during this thesis work, but found not suitable, because it does not support the communication from the target device to the serial port that was part of the system requirements.

kAWT supports GUI features similar to those in AWT such as buttons, textfields, sliders etc.

**MathFP**

As mentioned earlier, CLDC and KVM don’t support the floating point data type. One of the user requirements for this thesis work was to be able to navigate the VE using the Palm stylus. In order to implement the navigation operation, a normalisation of the x and y coordinates to floating point is required. Therefore, the need for a tool which provides the floating point data type was required for this system.
MathFixedPoint (MathFP) is a class library which simulates floating point computations on the KVM with no support for floats or doubles [13].

### 4.3.5 Class Diagram

The system was divided into three components, where every component is responsible for a different “tier”. Every component will be implemented by classes derived from J2ME with CLDC and kAWT class library. The class diagram is shown in Figure 4.2.

#### Interface Component

The interface component consists of one class (PDAKawtlet) derived from the kAWT class library, called Kawtlet, which is responsible for the interface and the application life-cycle (start, delete). This class is the only entrance to the application (is called when the application is started) and connects the user with the kernel part of the application. It also defines the initial graphical user interface that is shown when the user has started the application.
**Communication Component**

The communication component consists of one class `MessageHandler` which is responsible for message handling and contains another class `PortCommunication` which is responsible for the physical connection to the serial port and is a thread which is waiting for reading comming messages to the port.

`PortCommunication` implements the `Runnable` interface (for thread handling) and use `InputStreamReader` for reading from the serial port (in `run` function), and `PrintStream` for writing to the serial port (in `write` function). Its `run` function waits for messages to be read and send them forward to the `MessageHandler`’s function `process`. This thread is started up and is stopped by `MessageHandler`, however, the `stop` function doesn’t actually stop the thread and it remains for forward investigation.

`MessageHandler` is responsible to parse the comming message with the help of `StreamTokenizer`\(^2\) and send a message via `PortCommunication`.

The message is read into an array, which is sent to the `ObjectCreator` class.

The methods used here are from J2ME and CLDC.

**ObjectCreator**

This class is responsible for creating the object defined in the message. The creation commands for the objects are stored in a `Hashtable` and are called when the command “add” (for adding a GUI item) is sent in the message. When an object is created, the `despatch` is sending the message forward to the respective GUI item creator.

**Business Component**

This component is responsible for the implementation of the use cases and is divided into two kinds of “sub-components”:

1. **CreatePen**—A component for the navigation functionality (use case 1). It reads the x and y coordinates indicated by the user and normalise them to be at the range of $[-1,-1]$. In order to simulate floating point operations for the coordinates, `MathFP` was used. This class wasn’t defined as a subclass of `CreateWidget` (like the classes in the next point) but can be considered to be such a class in the future.

2. **CreateWidget**—A component for the creation, deletion and interaction of the user defined GUI items (use case 2, use case 3, use case 4). This component is a superclass with a number of subclasses, each class is responsible for creating and deletion of the respective GUI item, such as a menu, button etc. `CreateWidget` class consists of number of “command” classes, which are special

---

\(^1\)A thread contains of a `run` function which runs during the thread’s life, and is the place to put the actual implementation of the thread.

\(^2\)StreamTokenizer is not included in CLDC and was taken from the J2SE. StreamTokenizer is responsible for parsing a message.
classes, which implement commands, for example, the class AddOps implements the command add, and DeleteOps implements the command “delete”. The implementation of the commands are simply done by calling the respective command for the required object, for example, AddOps calls the function add in CreateButton class, if the required object to be added is a button. The Command Object idea and specification were taken from Design Patterns [5]. The command object helps to issue requests to objects without knowing about the operation being requested. For creating and deleting the GUI items, the kAWT library was used.

**Error Handler**

An additional class ErrorHandler was implemented for the error handling. It is responsible for presenting error messages to the user.
Chapter 5

Conclusions

5.1 The Implemented System

The system succeeded to implement an independent\(^1\) graphical user interface on the PDA display. This GUI is required and defined by the VR application (see the Protocol in the Appendix, for more information) and is sent to the PDA via a serial connection. The GUI may contain buttons, menus, menu-items, slider, labels and text-fields.

When sending the commands for creating or deleting GUI items, an OK or Error signal is sent back.

When interacting with a GUI item, for example by pressing a button or a menu item (which send an event) the definition (name) of the GUI item will be sent to the VR application for further treatment.

The system that was implemented is shown in Figure 5.1

5.2 Test

The tests for the system were done on a LINUX machine (Cockoo) connected to Boye and the Palm via a USB adapter (in order to connect two different kinds of port: Cockoo’s USB port and the Palm’s serial port), as shown in Figure 3.7. In order to communicate between Cockoo and the Palm for sending and receiving messages, the Kermit software was used.

5.2.1 The communication mechanism for the Joystick

In order to understand the test program which was used here, a brief introduction will be made of the Joystick communication to the Host, used today.

Snipe is responsible for sending the signals when buttons are pressed on the joystick to Boye. The Controllerproxy program is responsible for establishing the serial port connection from the joystick and reads the signal sent from it. After reading

\(^1\)The VR application is able to define its own graphical user interface.
Figure 5.1. The implemented system. The implemented GUI was defined by the VR application, according to a special protocol.
the signal, it is sent to Boye via the Controllerd which runs on Boye. Controllerd program writes the signal received, in Shared Memory on Boye, so that the VR application will be able to read it and interpret those signals to its use.

Trackerproxy and trackerd are two programs that run on Boye and communicates with each other. Trackerd which is running on Boye.

The Polhemus unit is connected via serial connection to Boye. There trackerproxy reads data and send that forward via TCP/IP communication to trackerd that writes this information in the shared memory.

These programs were written by number of developers at the Center for Parallel Computers (PDC) at the Royal Institute of Technology in Stockholm, Sweden. Controllerproxy written by Harald Barth and Trackerproxy, written by Daniel Linder.

The relationship between the different programs can be seen in Figure 5.2.

5.2.2 The Palm test program

A similar mechanism was used here. Trackerproxy was copied to a test machine called Cockoo and the Controllerproxy was replaced with Palmproxyn which is responsible for the serial port communication from and to the Palm. Notice that the connection here is read/write, compared to the read-only connection on the Joystick device. The control program for the joystick is running currently on the Snipe machine and is responsible for sending the signals of the buttons pressed on the joystick to Boye. The relationship between the different programs can be seen in Figure 5.3.
5.2.3 Uncompleted Test

The test program has succeeded to connect the Palm to Cockoo and send a GUI to the Palm but a number of problems have noticed, that hindered the continuation of the tests until further evaluations:

1. The system was slow.

2. The message sent by Cockoo wasn’t read properly by the system. The message was sent via a form of buffer, where this buffer wasn’t getting empty after every message.

3. The program does not ’die’ by itself, because the thread created for PortCommunication can not be killed. That required the user to perform a soft reset\(^2\) before every HotSync\(^3\) operation.

These problems caused that the test phase wasn’t accomplished completely and will be accomplished in a future work.

5.3 Future Work

This thesis work can be expanded to implement this interaction tool with some more functionality and features to offer the user. Suggested functionalities are:

- Update GUI item.

- Implement a wireless connection Infra Red (IR) between the PDA and the CAVE.

\(^2\)A soft reset is an operation to restart the Palm, when saving the workspace, compare to hard reset which restart the Palm with the initial Palm applications and without saving any data from before. A soft reset is done simply by pressing the stylus five seconds on the soft reset button.

\(^3\)This is the operation to synchronize programs into the Palm. That way one can download applications to the Palm from the connected desktop computer.
• Test other PalmPilots, such as Palm’s new “m” series: m105, m125, m130, m500 and m515.

• Test other PDAs such as iPaq (Compaq’s PDA).
Chapter 6

Appendix

6.1 Class libraries

In order to build the application, a number of class libraries have to be downloaded:

1. CLDC. Can be downloaded from Sun’s homepage at: http://java.sun.com/j2me/cldc/.

2. kAWT. Can be downloaded from kAWT’s homepage at: http://www.kawt.de/in the Download section. The kAWT class library has to be copied to the CLDC class library, in order to keep all the files in the same working directory.

3. xKVM. The KVM tool developed by kAWT and works better with kAWT class library. Can be downloaded from kAWT’s homepage in xKVM/ColorKVM section.


5. StringTokenizer is used for parsing the message that is received from the user. This file is not included in the CLDC class library, and therefore it is necessary to copy its .java file (source file) from the standard Java library to the working directory.

6. j2me_cldc-1.0-src-palm_overlay.zip. The CLDC classes for the use of Palm. These files are no longer supported by Sun, because they were a part of the Kjava package that has been moved from CLDC. The zip file is included with the program files.

6.2 Compilation and Running tools

To compile and run a J2ME application, Java tools are used.
6.2.1 Compilation

Javac is used for compilation for example:

    javac *.java

However, when using the CLDC class library, a link to that class has to be specified, by changing the classpath to read the CLDC library, for example:

    javac -bootclasspath c:\j2me_cldc\bin\api\classes;
c:\j2me_cldc\bin\common\api\classes *.java.

Preverification

To preverify the application, use the preverify tool in the CLDC class library:

    c:\j2me_cldc\bin\win32\preverify

In order to preverify the program files, type the following command:

    c:\\j2me_cldc\bin\win32\preverify -cldc -classpath
    c:\j2me_cldc\bin\api\classes;c:\j2me_cldc\bin\common\api\classes;
c:\j2me_cldc\bin\common\api\classes\net PDAKawtlet MessageHandler
    MessageHandler$PortCommunication java.lang.IllegalArgumentException
    net.jscience.math.kvm.MathFP ObjectCreator CreateWidget CreateWidget$1
    CreateWidget$AddOp CreateWidget$DeleteOp CreateWidget$Ops
    CreateWidget$ValidateOp CreateButton CreateButtonSp CreateLabel CreateMenu
    CreateMenuItem CreateTextfield CreateSlider CreatePen CreatePen$Pointer
    StreamTokenizer ErrorHandler

After the preverify process, binary files are created and laid under an automatically created output map. Those files get the extension *.class.

6.2.2 Conversion to a .PRC file

The executable file format of a Palm Pilot program is *.prc. In order to get such a file, a Java conversion tool MakePalmApp, which is included in the CLDC class library, is used.

This tool can be found in the directory: c:\j2me_cldc\tools\palm\src\palm\database. These files are not compiled yet, so before building a .prc file, a compilation has to be done.

The compilation is done in the directory: c:\j2me_cldc\tools\palm\src\palm\database by the command: javac *.java.

In order to build a Palm executable file, type the following command at the output directory of the application’s working directory:

    c:\jdk1.3\bin\java -classpath c:\j2me_cldc\tools\palm\src
    palm.database.MakePalmApp -bootclasspath .;c:\j2me_cldc\bin\api\classes;
c:\j2me_cldc\bin\common\api\classes;c:\j2me_cldc\bin\common\api\classes\net
    PDAKawtlet MessageHandler MessageHandler$PortCommunication
As a result a .prc file is created with the same name of the interface class that implements the Kawtlet interface. As can be seen, the special libraries MathFP and Kawt were included here. For further information about compiling and running see the kAWT homepage http://www.kawt.de, in Documentation and FAQ section, go to the link kAWT installation and compilation instructions for the CLDC Reference Implementation (Palm OS Devices) under the title Installation and Compilation.

6.2.3 Installation of a .PRC file

In order to install a .prc file, the Palm Desktop Software is used. The Palm Desktop Software is included with the Palm-device and is compatible with Windows platform.

The installation of a .prc file is done simply by double-click on the .prc file’s icon. Thereafter a question to confirm if one wanting to add this file to the Palm, is displayed. After confirming that, the .prc file is ready to be installed on the Palm. The actual installation is done when a HotSync operation is performed.

6.2.4 Running a .PRC file

Before running a .prc file, the KVM has to be installed. It is better to install the kAWT’s xKVM that works better with the kAWT class library. The related files are: KVM.prc and KVMUtil.prc. After downloading them, it is necessary to install them on the Palm in the same way as was done with the application file. To run the .prc file, one simply has to tap the application icon.

6.2.5 Communicating over the Serial Port with Telnet

In order to communicate the serial port, the Telnet software was used. Telnet makes it possible to open the serial port for reading and writing. This way we can send data to the serial port and receive data from it. The commands were written according to the protocol described in the next section. The tests were done using the following commands written to a file called: interface.txt:

```
button add jump Jump
button add stop Stop
button add select Select
buttons add fly Fly Walk
buttons add menu Menu-on Menu-off
button add exitb Exit
```
This file describes the required GUI for a VR application. The GUI consists of six buttons Jump, Stop, Select, Fly/Walk, Menu-on/Menu-off and Exit. The result is shown in Figure 5.1.

6.3 Protocol

6.3.1 General

This chapter describes the protocol used for the commands for communication with the PDA. These commands build the GUI on the PDA-display.

6.3.2 ASCII protocol

This protocol uses ASCII format.

6.3.3 Button

button Command Name Label

- Command—Add, Delete
- Name—name of button to be sent back as a signal.
- Label—label to be displayed on the button.

6.3.4 Buttons

Special kind of buttons that change their label as soon as they are pressed. buttons Command Name Label1 label2

- Command—Add, Delete
- Name—name of button to be sent back as a signal.
- Label1 Label2—label to be displayed on the button.

6.3.5 Menu

menu Command Name Label

- Command—Add, Delete
- Name—name of menu to be sent back as a signal.
- Label—label to be displayed on the menu.
6.3.6 Menuitem

`menuitem` Command `MenuItemName` `MenuName`

- Command—`Add`, `Delete`
- `MenuItemName`—name of menu item to be sent back as a signal. This name will be displayed on the menu item.
- `MenuName`—Menu that the menu item will be added to, or deleted from.

6.3.7 Label

`label` Command `Name` `Label`

- Command—`Add`, `Delete`
- `Name`—name of label.
- `Label`—text to be displayed on the label.

6.3.8 TextField

`textfield` Command `Name` `Label`

- Command—`Add`, `Delete`
- `Name`—name of text field.
- `Label`—text to be displayed on the text field.

6.3.9 Slider

`slider` Command `Name` `Value` `Visible` `Min` `Max`

- Command—`Add`, `Delete`
- `Name`—name of slider.
- `Value`—value that the scrollbar starts with.
- `Visible`—a number which specifies the increment unit.
- `Min`—minimum value to be set.
- `Max`—maximum value to be set.

6.3.10 Location

This event will send the location to the Host. The location will be sent as x and y coordinates of the position required by the user. This event will be sent to the VR application for 3D interpretation.
6.3.11 Return value

The return value is 0 if everything is ok and error code according to 6.3.12 if an error has occured.

6.3.12 Error codes

<table>
<thead>
<tr>
<th>Error code</th>
<th>Error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Command name is empty or not correct</td>
</tr>
<tr>
<td>11</td>
<td>Specification of GUI item is empty or not correct</td>
</tr>
<tr>
<td>12</td>
<td>Name of item is empty or not correct</td>
</tr>
<tr>
<td>13</td>
<td>Item name is already exist.</td>
</tr>
<tr>
<td>14</td>
<td>Label name is empty or not correct</td>
</tr>
<tr>
<td>15</td>
<td>Duplicate name or name does not exist for deletion.</td>
</tr>
<tr>
<td>16</td>
<td>Command is not complete. Wrong number of variables.</td>
</tr>
<tr>
<td>17</td>
<td>Item to be deleted does not exist.</td>
</tr>
<tr>
<td>19</td>
<td>Menu to be deleted is full. Delete menuitems first.</td>
</tr>
</tbody>
</table>

6.4 Program Notes

The program notes are added in ftp://www.pdc.kth.se/pub/PDAKAWTlet/
References


MathFP/.
j2me/.
[15] Sun Microsystems. Mobile Information Device Profile (JSR—37), JSR Spec-
midp/.
[18] Sun Microsystems. Mobile Information Device Profile, Specification, Final re-
Brody, and James Sosnoski. Distance learning classroom using Virtual Harlem.
In Proceedings of the Seventh International Conference on Virtual Systems and
evl.uic.edu/cavern/harlem/.
pdc.kth.se/projects/vr-cube/tech-info.html.
[22] Ivan Poupyrev, Numada Tomokazu, and Suzanne Weghorst. Virtual notepad:
[23] James Rumbaugh, Ivar Jacobson, and Grady Booch. The unified modeling
developer/pilot_os.html.
as an interaction device to a virtual environment. In Proceedings of the 3rd