Features of a Toolkit for open Human-Computer Interaction with Ambient Services

Andreas Lorenz
Fraunhofer Institute for Applied Information Technology
Schloss Birlinghoven, 53754 St. Augustin, Germany
andreas.lorenz@fit.fraunhofer.de

Abstract: For the deployment of ambient services it is unrealistic to base user-computer interaction on traditional mouse or keyboard. Replacing them with other proprietary devices just moves boundaries in interacting with ambient services to other designated hardware. It still does not pay attention to preferences and capabilities of particular users. Our aim is to abstract from concrete devices used for controlling services in the environment. In this paper we introduce a toolkit for developing software solutions working with any device as input device delivering meaningful events to services.

1 Introduction

In traditional graphical User Interfaces (GUI), the user employs a physical input device like a mouse or keyboard in order to deliver input to application dependent graphical components. Most programming languages offer components to support the development of graphical user interfaces, together with back-end mechanisms for distributing user input (like subscribe-inform). In this case, the service provides a processing method that is associated to input events occurring on the control component.

The physical input device is a transporter of the user intention to the service. The expression of input in ambient computing environments is generally not bound to a specific device but to the meaning to the service. In fact, any device could be used to express user input in such environments. Like toolkits supporting the development of graphical interfaces, our aim is to support efficient development of ambient systems receiving input from any device. Our abstract solution is shortly explained in the next subsection.

1.1 Framework Description

A software framework is an abstract design for a category of software systems. It defines a set of cooperating components, and the control flow in the system [LL07]. Our goal is to enable software developers to create innovative input methods for ambient services. For definition of our architecture, we therefore abstract from concrete user devices by defining a *Virtual Input Device*.

Virtual Input Device A Virtual Input Device is a source of meaningful user input without constraints for physical shape of the device, type of user interface or interaction style.

The framework proposes a prototypical design for the software architecture. The Virtual Input Device crosses the boundaries of the input device and the controlled device. On both sides it virtually creates a local event flow independent of concrete realization, physical distribution, programming languages and operating systems. The elaboration of the framework is described in more detail in [LEZ08].

In this paper we describe helpful features of software-tools aiding a developer to instantiate the architecture and work with the code automatically derived from the specification of Virtual Input Devices. A first realization of the tools, currently under evaluation by experts, uses Java as programming language for client and server.

2 Related Work

The widespread adoption of mobile devices has encouraged the development of alternatives to keyboard and mouse. Current tools map different devices with the mouse pointer and keyboard to work with the WIMP-metaphor. As an example, the USB Overdrive [USB] handles similar devices (like trackball, or joystick) for mouse pointer control. Head-tracker solutions like [Kje06] are designed to work with gestures for replacing traditional pointing devices. Using a Web-cam, it allows users to point and click by simply aiming their face. A combination of pointer position and keystroke input device is described in [AM06], using miniature video cameras that track finger position where the user can type or point in the air. The advantage is the high integration with the operating system. After connection, the device is able to perform all actions of the original device. The user is not requested to learn any new control for operating her favorite applications.

At the same time, this close link is one of the most important drawbacks: As kind of simulator, the interaction style and expression is adapted to exactly fit the same behavior of the simulated device. The new device or interaction style cannot use it's full potential for creating new innovative interaction, potentially being more adequate. Furthermore, it is limited to interact with windows-based systems where mouse and keyboard have a certain meaning. Other modalities apart from mouse / keyboard are not supported. Moreover, the solutions have limited portability to other hosts, specifically crossing operating systems. In particular where hardware drivers are of importance, they do usually not run on different operating systems. The USB Overdrive for example is only working on Mac OS X, because other drivers are not available. If systems fall back on a wired connection of the input device, e.g. by USB, the remote aspect of input is questionable anyway.

The iStuff Mobile architecture [BMRB07] is a platform combining (physical) sensor enhanced mobile phones and interactive spaces. The platform uses an Event-Heap [JF02] for distributing events of a specific type with specific fields. A mobile phone is then capable of sensing (local) user activity (e.g. key pressed) that are posted as (iStuff-)events on the heap. Though the events are similar defined by other vendors, the encoding and way of

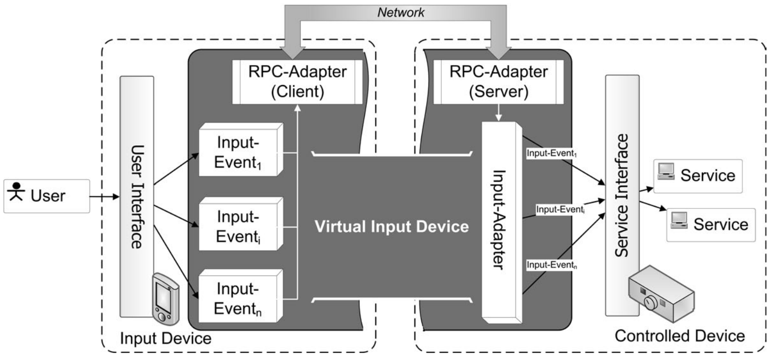


Abbildung 1: The software architecture supported by the toolkit

transmission are not compatible with others.

3 The Toolkit

The framework described above proposes a prototypical design of a system's architecture. The Virtual Input Device specifies the interface between user interface components and remote ambient services. This specification of the interface can be transferred automatically into software components doing much of the work of the back end:

- components implementing information transfer (i.e. network connection management, information encoding/decoding, data transmission)
- components for creating events on the client (stub)
- components receiving and processing events on the server (skeleton)

Neither the developer of the user interface on the input device, nor the developer of the ambient service needs expertise in distributed system development.

Figure 1 illustrates one instantiation of the framework using Remote Procedure Calls [XML]. All solutions build with our toolkit will apply to this reference software architecture. The Virtual Input Device virtually builds a bridge between software components physically distributed on input device and controlled device. In the figure, the top cover of the box is open for illustration purposes. In daily work, the components inside the box do not need attention by the developer, because they are derived automatically from the specification of the Virtual Input Device. In addition, stub and skeleton include default implementations available for quick tests of the information exchange.

In general, the aim of a toolkit is to help developers to build applications more efficiently using a set of programs, scripts, macros, documentation, and other aids. In our work, we will use the following definition of a toolkit:

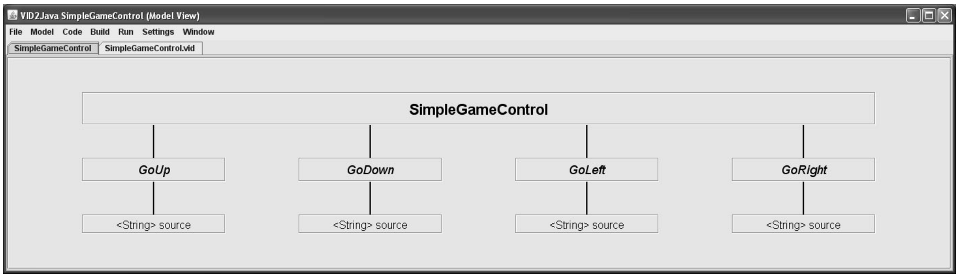
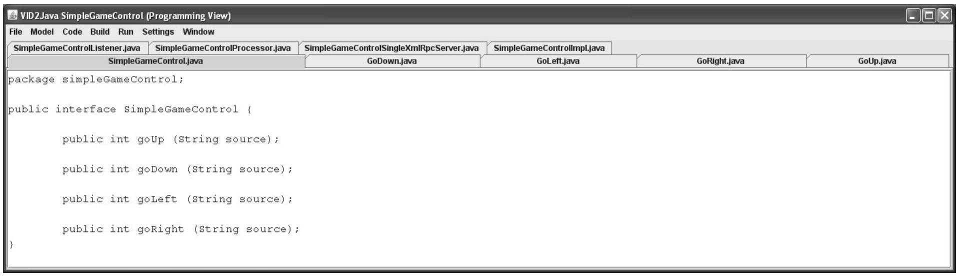


Abbildung 2: Example model of a simple gaming control


VID2Java SimpleGameControl (Programming View)
File Model Code Build Run Settings Window
SimpleGameControlListener.java
SimpleGameControlProcessor.java
SimpleGameControlSingleXmlRpcServer.java
SimpleGameControlImpl.java
SimpleGameControl.java
GoDown.java
GoLeft.java
GoRight.java
GoUp.java

```
package simpleGameControl;

public interface SimpleGameControl {

        public int goUp (String source);

        public int goDown (String source);

        public int goLeft (String source);

        public int goRight (String source);
}
```

Abbildung 3: The code files for the Java-server of the simple gaming control

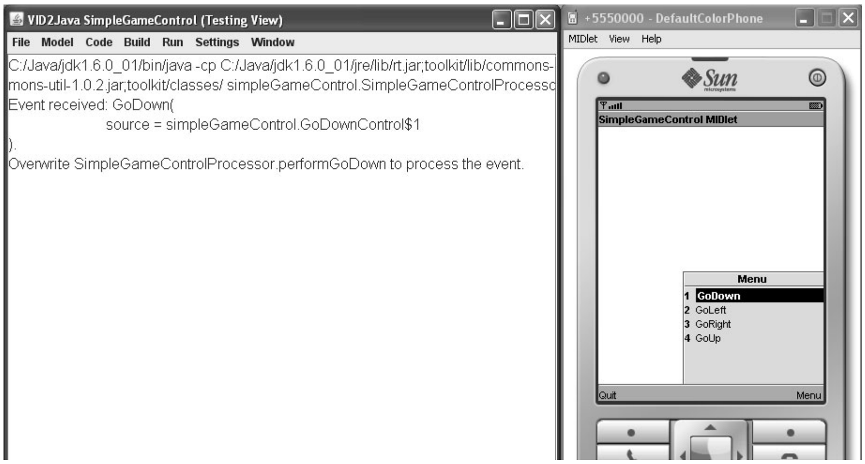
Toolkit A Toolkit is a compilation of re-usable software, specification of interfaces, and advice for developing software systems for a specific problem.

The aim of our toolkit is to provide a comfortable way of specification of the Virtual Input Device, quickly checking whether it meets its requirements, and creating libraries to be extended with more functionality. It is a compilation of five different tools, each supporting a single step in the system engineering process: Modeling of the interaction, creating code, building and testing prototypes, and deployment of a final solution. In this chapter we will explain each tool and exemplify a solution for remotely controlling a simple gaming application, which was used for controlling a PacMan-game in [ELZ08].

3.1 Modeling

The modeling tool provides a graphical interface for defining the Virtual Input Device. With the graphical tool, the developer models the data that will be transferred to the target service(s). In particular, the developer defines the events and their parameters. As illustrated in Figure 2, only the meaning is defined regardless of the physical shape of an event source (i.e. the input device) or other details of realization.

Example: A developer defines four control commands of a simple game controller: Up, Down, Left, and Right. Each command is associated with an identifier of the event source.


VID2Java SimpleGameControl (Testing View)
File Model Code Build Run Settings Window
+5550000 - DefaultColorPhone
MIDlet View Help
SimpleGameControl MIDlet
Menu
1 GoDown
2 GoLeft
3 GoRight
4 GoUp
Quit
Menu

```
C:/Java/jdk1.6.0_01/bin/java -cp C:/Java/jdk1.6.0_01/jre/lib/rt.jar;toolkit/lib/commons-
mons-util-1.0.2.jar;toolkit/classes/ simpleGameControl.SimpleGameControlProcessc
Event received: GoDown(
            source = simpleGameControl.GoDownControl$1
).
Overwrite SimpleGameControlProcessor.performGoDown to process the event.
```

Abbildung 4: Testing the server (on the left) with emulated MIDlet (on the right)

3.2 Programming and Building

The programming tool translates the abstract definition of a Virtual Input Device into programming code. The tool creates a local server notifying any registered listener about incoming events, where the developer has to overwrite the event-performing part of the skeleton in order to integrate own service logic. The tool also creates the client-stub that can be contacted by the concrete user interface handing over the data. The generated software automatically distributes the events to remote services over the network.

Example: Figure 3 illustrates the Virtual Input Device translated into a Java-interface. In the tabular list on top of the code you see implementation for the events, an event-listener, the skeleton of the processor which the developer has to extend with own service logic, and the class implementing the Virtual Input Device.

The building tool is then used to compile programming code into executable code. The main purpose is to enable rapid checks of changes of the Virtual Input Device: When the device was changed, the code is automatically updated by the programming tool and the building tool can be used for compiling.

3.3 Testing and Deploying

The testing tool is able to execute code generated by the building tool. It starts a default server instance and potential clients. It uses the emulator of Sun's Wireless Toolkit for running Java-mobile clients. The testing tool empowers the developer for quickly testing different versions of client user interfaces or service logic, and rapidly evaluation of the effect of changes of the Virtual Input Device.

Example: After building the generated code, the developer tests the communication by running a server instance (on the left side of Figure 4) in combination with an instance of the client MIDlet (on the right side of that image).

With the deployment tool all binaries, libraries and scripts are packed to form a solution. Usually, it will not be delivered to customers without modifications. It is more intended to serve as a basis for further implementation, i.e. for overwriting and extending the code.

4 Summary and Conclusion

In our work we developed a method for connecting input devices with ambient services. Our approach of Virtual Input Devices enables for the abstract definition of the input without physical restrictions or interaction dogmatism. The toolkit translates the framework architecture into a software architecture based on XML-RPC. The output of the kit covers the complexity of the procedure to deliver input events to remote service(s).

Literatur

[AM06] F. Ahmad und P. Musilek. A keystroke and pointer control input interface for wearable computers. In *4th Intl. Conference on Pervasive Computing and Communications*, Seiten 2–11, 2006.

[BMRB07] Rafael Ballagas, Faraz Memon, Rene Reiners und Jan Borchers. iStuff Mobile: Rapidly Prototyping New Mobile Phone Interfaces for Ubiquitous Computing. In *SIGCHI Conference on Human Factors in Computing Systems*, Seiten 1107–1116, San Jose, CA, 2007. ACM Press.

[ELZ08] Markus Eisenhauer, Andreas Lorenz und Andreas Zimmermann. Interaction-Kiosk for open Human-Computer Interaction with Pervasive Services. In *Advances in Pervasive Computing: Adjunct Proceedings of the 6th International Conference on Pervasive Computing*, Seiten 134–137. Österreichische Computer Gesellschaft, 2008.

[JF02] B. Johanson und A. Fox. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. In *IEEE Workshop on Mobile Computing Systems and Applications*, Seite 83. IEEE, 2002.

[Kje06] Rick Kjeldsen. Improvements in vision-based pointer control. In *SIGACCESS conference on Computers and accessibility*, Seiten 189–196, New York, 2006. ACM.

[LEZ08] Andreas Lorenz, Markus Eisenhauer und Andreas Zimmermann. Elaborating a Framework for Open Human Computer Interaction with Ambient Services. In *4th International Workshop on Pervasive Mobile Interaction Devices*, 2008.

[LL07] Jochen Ludewig und Horst Lichter. *Software Engineering*. dpunkt.verlag, Heidelberg, 2007.

[USB] USB Overdrive. http://www.usboverdrive.com.

[XML] XML-RPC Home Page. http://www.xmlrpc.org/.