Integration of Distributed User Input to Extend Interaction Possibilities with Local Applications

Kay Kadner and Stephan Mueller

SAP AG, SAP Research CEC Dresden
Institute of System Architecture, Faculty of Computer Science,
Dresden University of Technology
kay.kadner@sap.com, sm853234@inf.tu-dresden.de

Abstract. Computing devices do not offer every modality for interaction that a user might want to choose for interacting with an application. Instead of buying new hardware for extending the interaction capabilities, it should be possible to leverage modalities of independent existing devices that are in the vicinity. Therefore, an architecture has to be developed that gathers events on distributed devices and transfers them to the local device for execution. This allows the user to choose devices even at runtime that are better suited for a particular input task. For a convenient use, the system should support input that can be both independent and dependent from the application. Application-dependent input commands imply that meta-information about the application is provided. Since the system should allow the extension of existing applications, the meta-information has to be provided in a way that is transparent for the application. The following paper describes a system that realises those features.

1 Introduction

Electronic devices are ubiquitous in the modern society. According to a market survey [1], the number of mobile phone contracts exceeded the number of residents in Germany at the end of 2006, whereas this is already the case in other countries. PDAs are getting increasingly popular (besides mobile phones), which allow the user to communicate over a variety of technologies like Bluetooth, WiFi, GSM, UMTS and so on. Users always carry PDAs and mobile phones with them, which makes those devices ubiquitous computing units, whose capabilities are barely utilised today.

The different interaction modalities of independent devices like pen input of PDAs can not easily be used to extend the interaction modalities of applications running on a PC, for instance. Many work has been done for extending a PC's desktop, e.g. by remote desktops like VNC [15]. Similar approaches allow implementing distributed services, which can be accessed from various devices resulting in various user interfaces. As a consequence, most related work has explicit impact on applications in order to extend them for remote access or simply replicates the original appliation's user interface. A system, which allows a lightweight extension of an application's user interface without the need of modifying the implementation or replicating the whole user interface while being open to arbitrary modalities is still missing.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-92698-6_37

J. Gulliksen et al. (Eds.): EIS 2007, LNCS 4940, pp. 275–284, 2008.

Therefore, we developed a system, which enables the user to leverage the additional interaction modalities of independent devices by capturing distributed input and forwarding it to the application PC, which does not necessarily has to be a desktop PC or laptop. In fact, every device is suitable, as long as it is able to run the remote control application, which receives input and issues that in into the system. For the remainder of the paper we assume that the application PC is the entity, whose interaction capabilities are limited and thus should be enhanced by the use of additional devices. The independent devices are connected to the PC over a network. An overview of the architecture and its participants is shown in Fig. 1. Once the independent devices are connected, the user can use them to issue application-independent input (e.g. simple keyboard input) and, after determining the currently active application on the application PC, the remote devices can also be used to issue application-dependent input commands (e.g. "next slide" in MS PowerPoint).

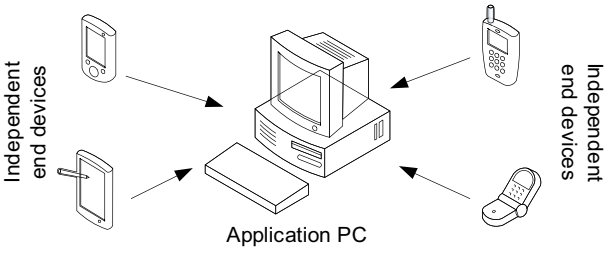

Fig. 1. High-level view of the remote input system

The presented architecture does not provide a separate view on the application through the independent end devices. It is impossible to use the end devices without having access to the application PC, because they do not give feedback originating from the target application to the user. They only provide an extension of the traditional input capabilities according to their local modalities, whereas the extended input capabilities are used in addition to the traditional ones. This results often in additional buttons, but is not limited to a particular representation, which only depends on the devices' modalities.

The paper is structured as follows: an example use case and the requirements to such a system are described in Section 2. Concepts of the architecture are explained in Section 3. Section 4 contains a description of the implemented prototype before the paper closes with a brief overview about related work in Section 5 and a conclusion and outlook in Section 6.

2 Use Case and Requirements

This section gives a brief description of an example use case followed by seven requirements that must be fulfilled for developing a convenient and secure system for the described scenario. At the end of the paper, the requirements are used to assess the developed prototype.

2.1 Use Case

For graphical designers, it is often more easy to use a stylus for drawing instead of the mouse because of the different way of interacting with the device. Therefore, they usually have a tablet for stylus input, which is attached via cable (e.g. USB) to the PC or laptop. However, such a tablet is not available in every situation, e.g. if the graphical designer is travelling from his company to the customer. His PDA offers the same way of interaction, so he turns it on and configures it for its use as stylus input device on the laptop. An application is started, which captures the cursor movements within a certain area and forwards those to the application PC. Arriving at the customer site, the designer has to give a presentation, which he prepared as a set of slides. As he does not want to be tied to his laptop, he again uses his PDA and configures it for controlling the presentation application. The PDA provides access to commands that are most important for giving a presentation (previous, next, home) and leaves out other commands that are rather used for editing (copy, paste). In both cases, the PDA has no physical connection to the laptop but communicates over wireless networks.

Besides replicating already available interaction means like buttons or menus on the client device, it is also possible to create shortcuts to functionality that might be hidden or hard to access as well as a combination of multiple functionalities. In MS Word for instance, if you want to turn on the thumbnails view, you have to click the respective button from the View menu. By creating a shortcut button to that functionality on the PDA, you only need one click instead of two. More complex commands like selecting and formatting a paragraph to a certain font type, font size, and line spacing can also be defined.

2.2 Requirements

The requirements for the scenario are described in the following:

1. For ensuring a high flexibility with regard to extensibility of the application PC, the system must allow the use of independent and distributed devices for remotely issuing input. Due to their distributed nature, those devices must be connected to the PC over a network (R1), which is still a loose coupling.
2. The independent devices must allow for both application-dependent (R3) and application-independent input (R2). Since the devices are heterogeneous, they present their local user interface according to their user interface capabilities, which results in multimodal interfaces and therefore multimodal control of the PC's applications.
3. Furthermore, the system must provide means for securing the communication channels between the application PC and the independent devices (R4), because modifying or recording the communication channels by unauthorised parties must be prevented. This is especially critical for such a system as it is described here, because it realises the interaction between user and applications, which is regarded to be safe in traditional scenarios (PC and attached keyboard and mouse).
4. Since the system must allow the extension of existing applications (R7), it needs information about available application commands that the user wants to use.
5. Because it is not feasible that existing applications are modified and not all applications provide interfaces for accessing this information, the system must implement a means to provide the command description for application-dependent input

commands in another way (R5), which allows the independent devices to create output in their available modalities.
6. The independent devices must always be informed about the currently active application (the target application) on the PC, because this affects their locally offered application-dependent input commands (R6).

3 Architecture of the System

In the first part of this section, the architecture of the system will be presented. The architecture supports both application-independent and application-dependent input. The latter type is called *input commands*, since this input is usually at a semantically higher level than application-independent input. Application-independent input is realised by simply forwarding the keyboard and mouse events, which can be regarded as simple input commands and are therefore not explained further in this paper. The realisation of application-dependent input commands will be explained in the last part of this section after a detailed view on the server-side input receiver.

3.1 Architecture

The system architecture is shown in Figure 2. The application PC can be controlled by traditional local input via keyboard and mouse. The independent end devices make also use of their local input methods, whereas the remote control application can distinguish between application-independent or application-dependent input. Application-independent input can always be used, whereas application-dependent input is especially tailored for the application that should be controlled. Depending on its concrete implementation, the remote control application can offer separate subsets of the overall interaction possibilities on the end device for supporting different kinds of tasks, for instance navigating requires cursor keys, page up/down, home/end, whereas for text input, a soft keyboard is be required.

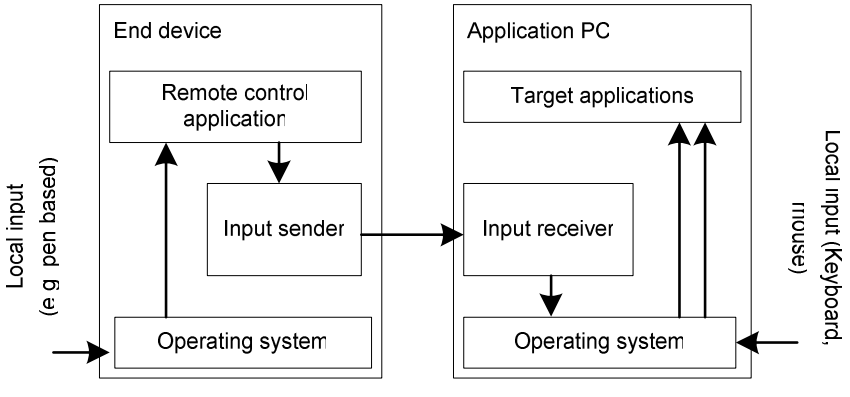

Fig. 2. Architecture of the system

Application-independent input is mapped on device-independent keyboard or mouse events before it is forwarded to the application PC. If application-dependent input commands are entered, their ID, as specified in the application description document (see 3.3), is forwarded to the application PC where the ID is mapped to concrete keyboard and mouse events. Mapping input commands to concrete events on the application PC is advantageous because a different implementation of the input receiver may be able to realise the command's function in a different way, e.g. by directly accessing some interface of the target application. In this case, the input sender implementation does not need to be changed. However, the application description also supports the other way, i.e. executing the events that belong to a command on the end device. This solely depends on the concrete implementation of the remote control application.

3.2 Details of the Input Receiver

The input receiver is responsible for receiving and processing the events according to application-dependent and application-independent input on the client devices. The actual reception is implemented in the Receiver component (see Fig. 3). It listens on a specific port for incoming connections and checks with the Security component, if the requesting client is allowed to connect. The security component can for instance simply display a warning to the user and asks for permission. The channel component encapsulates the communication details of sending to and receiving from the client device. Sending messages is omitted in Fig. 3 since it is only used for notifying the client about updates of the currently active application. If the channel component receives a message with input events, it dispatches them to the appropriate subsequent component. Application-independent events are forwarded to the Event service, which generates the according events in the operating system. Application-dependent input commands are first processed by the Command service, which maps the input commands to concrete key events and forwards them to the Event service for generation. As already mentioned, another implementation of the Command service may directly access the target application via an interface and execute the input command by calling an appropriate method. The context component provides information about the application PC, which is relevant for generating application-independent events on the client, like the display resolution or the used keyboard layout.

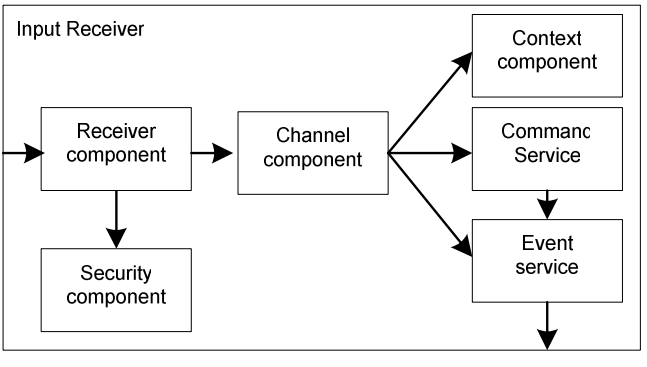

Fig. 3. Detailed architecture of the Input Receiver

Both the channel component and the event service are responsible for handling sudden disconnections of client devices. This needs special attention because otherwise it might happen that a "key press" event is properly executed but the according "key released" event does never occur. The Event service is notified by the channel component, which device has been disconnected. Now, the event service is able to perform compensation actions. The compensation is realised by artificially creating the proper compensation event (the "key released" in case of keyboard input).

3.3 Creation of Application-Dependent Input Commands

For creating application-dependent input commands, the remote control application needs information about the target application on the application PC. The remote control application then downloads a description file of this application from the application PC, which contains a list of application-dependent input commands. For ensuring a convenient user interface on the independent devices, the file consists of information how each command can be realised in different output modalities because an extensive textual explanation can be useful for a monitor but not for voice output.

Figure 4 shows an example of an application description file, which defines the command "next" that is used for forward navigation in a presentation. The remote control application chooses between a long or short name for rendering, depending on the current situation for rendering output (e.g. number of commands, screen size). If speech recognition is used, a voice recognition grammar can be created based on the provided voice commands. In the `<events>` section, the concrete key event types and codes are specified, which shall be created on the PC. The two key events shown in the example are the Eclipse SWT [3] definition of the right cursor key.

```
<application name="MS Powerpoint" version="2003">
  <command id="next">
    <shortname>Next</shortname>
    <longname>Next Slide</longname>
    <speechcommand>next</speechcommand>
    <events>
      <keyboard type="keypress" keycode="16777220"/>
      <keyboard type="keyrelease" keycode="16777220"/>
    </events>
  </command>
</application>
```

Fig. 4. Example of a configuration file for application dependent commands

4 Prototype

The prototype was implemented in Java for ensuring platform independence. The determination of the currently active application on the application PC is not possible in Java and would imply implementations on operating system level. Since this contradicts with the platform independence, we decided to leave the task of application determination up to the user. Therefore, the user has to define manually the currently

active application on the application PC by selecting it in the administration tool (R6, Figure 5). A combination of the Eclipse Standard Widget Toolkit [3] and the `Robot`-class of the Java Abstract Window Toolkit [9] was necessary to create events on the application PC because neither of them is capable of creating all possible events.

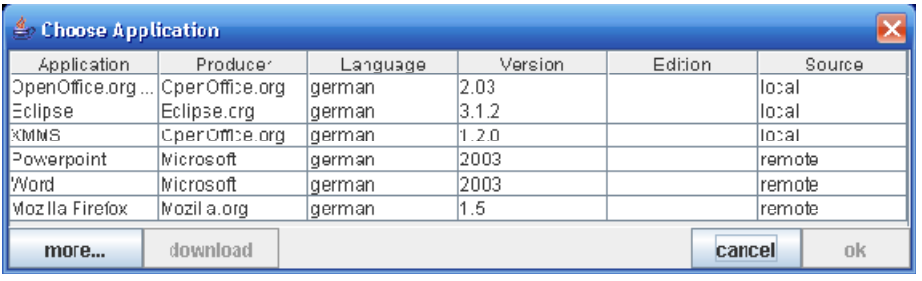

Fig. 5. Screenshot of the administration tool

We decided to implement the application on the end device with Java Micro Edition (JME, [8]) for supporting a wide range of client devices like mobile phones and PDAs. This application allows establishing a connection over plain sockets to the application PC (R1) and issuing of application-dependent and application-independent events (R2, R3, Figure 6). Because of both, the indirect injection of the input events from the PC into the target application via the operating system and the external description of applications (R7), a transparent extension of existing applications is possible (R5).

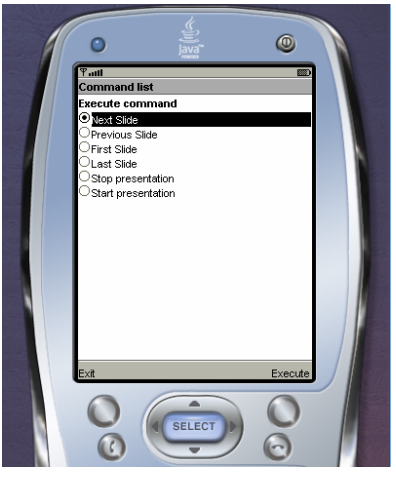

Fig. 6. Screenshot of the JME client

If the end device connects to the PC, the user, which is currently logged onto the application PC, will be asked whether to allow or deny the access by the end device. The connection would have to be encrypted for completely fulfilling the security requirement (R4). It was first intended to use an HTTPS connection, but since the SSL implementation Java ME caused several problems, which were not solvable within the project, we decided to leave secure communication out. Figure 5 shows the configuration application of the PC and Figure 6 contains a screenshot of the JME application on the right.

5 Related Work

The Input Adaptation Project (IAT) has the goal to replace standard input devices like mouse or keyboard with other input devices [10]. A complex theoretical model is the basis for mapping input events of other devices to concrete keyboard events like pressing a key. The process of adapting the input events is done in three subsequent phases. First, the input events are transformed by a device-specific InputAdapter to a uniform state space. In the second phase, the uniform state space is mapped to the states of the target device. Finally, in the third phase, the states of the target devices are mapped to one or multiple concrete events, which are then created by the target devices. The last mapping is realised by a device-specific OutputExplorer. Since the mappings are defined in a special description language and a change of that mapping description is not possible at runtime, the mapping cannot be changed according to the currently active application, which is a major goal of the presented system.

The project Pebbles [6] investigates scenarios in which PCs and PDAs can collaborate. Pebbles' architecture is divided in three components: the *service user* runs on the PDA and offers the user interface. The *service* runs on the PC and controls the application and the system, and the *PebblesPC* runs on the PC and acts as name service as well as message router between the service user and the service. This architecture offers the possibility to create adaptations of special applications with application-dependent services. Nevertheless, each application that should make use of the Pebbles architecture has to be implemented seperately. Therefore, the flexibility of enhancing an arbitrary application is limited. Furthermore, PebblesPC only supports the collaboration of PDA and PC for including pen input. Other modalities were not considered.

With the help of the Personal Universal Controller (PUC, [13]), users are enabled to interact with arbitrary appliances in their environment. The PUC architecture consists of four elements: the appliance adapter, the communication protocol, the specification language and the user interface generator. The communication protocol supports peer-to-peer communication, thus allowing multiple user interface generators to be connected to multiple appliance adapters. The specification language describes the user interface in an abstract way and is used to transport state information of the appliance. The PUC system focussed on automatic generation of high-quality user interfaces for appliance control, which is different from providing additional input means to applications, because each appliance needs a special adapter with a special user interface specification. In contrast, controlling a previously unknown application is always possible, because application-independent input is always supported.

The ICrafter service framework allows the user to interact with services in the user's interactive workspace [14]. Services can be devices or applications that provide useful functions. The service framework is used by developers to deploy services and to create the services' user interfaces for various appliances. Appliances request user interfaces from the interface manager for a particular application, specified by an application description. The interface manager retrieves all necessary information (service and appliance description, context information) and generates a user interface, which is returned to the requesting appliance. As the PUC before, services for the ICrafter system must be explicitely implemented. Our solution focusses on the extension of existing applications without the need to reimplement them.

The aim of the OSI Virtual Terminal Service (VTS) is to provide access to virtual terminals within a distributed network [11]. The VTS employs a mapping of events to a uniform state space on the client device and vice versa on the terminal device, which is quite similar to the mapping function of the IAT project. Although the system distinguishes objects that are used for either output or input only, it is used for remote terminal access, which involves input and output of information through the same device. In contrast to the presented approach, the VTS aims at replicating the target applications at the application level. This implies an enormous implementation effort on the application PC, because not all applications are suitable for that. This prevents a seamless and easy extension of existing applications.

Microsoft developed the Remote Desktop Protocol [12] for accessing Windows machines (the server) from different hosts (clients). This is realised by forwarding the user interface (i.e. the desktop) to the client where the local input is gathered and forwarded to the server. By this, the user can access the server's applications through a different device. However, the user can still use mouse and keyboard only. The client does not act as an extension of the already existing interaction capabilities but instead replaces the currently existing device.

Furthermore, there are several voice navigation systems (e.g. Realize Voice [7], VRCommander [4], and e-speaking [2]) that enable the user to extend existing applications by voice commands. This includes the use of application-dependent input commands in order to control the applications in a more goal-oriented way. Using these systems limits the extension to voice interaction and the static use of command mappings since they are unable to dynamically adjust the mappings at runtime.

The goal of Salling Clicker (http://www.salling.com) is similar to that of our approach. However, the conrete architecture is not described on their website. The scripts for application extension are based on Javascript and do not support the employment of additional modalities like voice. Additionally, application-independent input is not possible.

6 Conclusion and Outlook

The presented system offers a transparent extension of existing applications with interaction capabilities of various end devices. This is achieved by gathering distributed input events and forwarding them to the applications on the PC. Due to the explicit support of heterogeneous end devices, multimodal application control based on a federation of end devices is possible. This covers the input side of the architecture

for federated devices, which is sketched in [5]. The transparency is achieved by using external application description documents for defining application-dependent input commands, which define the mapping of higher level input commands to conrete mouse or key events, which are executed on the application PC. The application description supports rendering in multiple modalities by providing several representations of a certain element, which can be used by the remote control application according to the current rendering situation.

The system can be further developed for supporting program APIs for command execution instead of mappings to keyboard events and indirect injection through the operating system. However, this requires a more comprehensive application description than the current one. It should also be evaluated, if and how this approach can be used to increase the robustness against erroneous user input especially if multiple end devices are used. Furthermore, the possibilities to ensure trust and integrity have not been fully employed in the current prototype and might be realised by the use of SSL communication or certificate authentication. The application-dependent input may be extended by supporting parameters that enhance the actual command.

References

1. Axel Springer, A.G.: Telekommunikation 2006. Market report (2006)
2. e Speaking.com: Voice and Speech Recognition, http://www.espeaking.com
3. The Eclipse Foundation: The Standard Widget Toolkit, http://www.eclipse.org
4. Interactive Voice Technologies: VRCommander, http://www.vrcommander.com
5. Kadner, K.: A flexible architecture for multimodal applications using federated devices. In: Proceedings of Visual Languages and Human-Centric Computing, Brighton, UK, September 2006, pp. 236–237. IEEE Computer Society, Los Alamitos (2006)
6. Myers, B.A.: Using handhelds and PCs together. Communications of the ACM 44(11), 34–41 (2001)
7. Realize Software Corporation: Realize Voice, http://www.realizesoftware.com/
8. Sun Microsystems: Java Micro Edition, http://java.sun.com/j2me
9. Sun Microsystems: The Abstract Window Toolkit, http://java.sun.com/j2se
10. Wang, J., Mankoff, J.: Theoretical and architectural support for input device adaptation. In: CUU 2003: Proceedings of the 2003 conference on Universal usability, pp. 85–92. ACM Press, New York (2003)
11. Lowe, H.: OSI virtual terminal service. Proceedings of the IEEE 71(12), 1408–1413 (1983)
12. Microsoft Corp.: Understanding the Remote Desktop Protocol (RDP), http://support.microsoft.com/kb/186607
13. Nichols, J., Myers, B.A.: Controlling Home and Office Appliances with Smart Phones. IEEE Pervasive 5(3) (July-September 2006)
14. Ponnekanti, S.R., Lee, B., Fox, A., Hanrahan, P., Winograd, T.: ICrafter: A service framework for ubiquitous computing environments. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) UbiComp 2001. LNCS, vol. 2201, p. 56. Springer, Heidelberg (2001)
15. Richardson, T., Stafford-Fraser, Q., Wood, K.R., Hopper, A.: Virtual network computting. IEEE Internet Computing 2(1), 33–38 (1998)