Playing with the Real World

Paul Holleis, Matthias Kranz, Anneke Winter, Albrecht Schmidt

University of Munich
Research Group Embedded Interaction
Amalienstraße 17
80333 Munich, Germany
{paul,matthias,anneke,albrecht}@hcilab.org
www.hcilab.org

Abstract

In this paper we provide a framework that enables the rapid development of applications using non-standard input devices. Flash is chosen as programming language since it can be used for quickly assembling graphical applications. We overcome the difficulties of Flash to access external devices by introducing a very generic concept: The state information generated by input devices is transferred to a PC where a program collects them, interprets them and makes them available on a web server. Application developers can now integrate a Flash component that accesses the data stored in XML format and directly use it in their application. We show two examples, one from a pervasive gaming background and one from an installation in an office setting.

Keywords: Design, Experimentation, Human Factors, Pervasive Computing, Input Device, Output Device, Sensors, Game Controller, Playful Computing

First presented at the Workshop PerGames 2005 held at PERVASIVE 2005, extended and revised for JVRB

1 Introduction

Gaming is and always was ubiquitous and pervasive: children play card games on long travels in the back seat of the car, teenagers use their high-end personal computers and business people play with their highly sophisticated mobile devices - as is stated in [Nie05] killing time is the killer application and gaming certainly is designated for this purpose.

In this paper we present a general concept for rapid prototyping games that reach out in the physical world. Our focus is on non-standard input techniques using physically embedded controllers. We concentrate on prototyping devices beyond mouse and keyboard as we think that special purpose devices are much more interesting and suitable to gaming.

We provide a general architecture for communication between input and output devices and applications using existing standards and protocols. The data, especially sensor data, representing the physical states of the input device is transferred to a server where it is made available to applications. The actual sensor data is provided in human readable form coded in XML. The server is a lightweight web server that can be accessed via the HTTP protocol which is available in a great variety of programming platforms and languages.

On the gaming application side, we demonstrate the integration of novel input devices in Macromedia Flash. This multimedia authoring tool and the programming language ActionScript offer great flexibility and possibilities to integrate all kinds of media (sound, graphics and movies). This platform has also been widely accepted for small-scale applications, especially games. We therefore provide a Flash component that entirely hides the complexity of the data retrieval, pre-processing and transfer to the application. The Flash programmer can easily and directly access the values delivered by the input device as local variables. No special knowledge of the input device like design, electronic layout or hardware used, is needed by the application programmer. These facts are completely hidden and shielded.

2 Programming Beyond the Desktop

The focus of traditional programs and especially games has primarily been on running these applications on standard desktop computers, mobile devices, and game consoles. Such devices unify input and output (e.g. in the case of a standard PC mouse, keyboard and screen). Even with the emergence of multiplayer games and later on internet based online gaming platforms, the computer screen and speakers remain the only output devices (besides force feedback controls) and mouse, keyboard and joystick or conceptually similar devices like rumble pads or steering wheels are still the dominant input devices used.

From this perspective, we see a large potential for the development of ubiquitous gaming devices. We think, that this area is another fertile field of research which can benefit from rapid prototyping. In recent works on ubiquitous gaming, novel interaction devices like a torche [GSK$^+$02], a flying jacket [RSH$^+$02] or a cushion [SHK04] are proposed.

We contribute to this by providing a generic architecture and implementation for connecting novel and non-standard input devices with applications.

2.1 New Input and Output Devices

The computer domain has largely been dominated by systems with a relatively large display, capable of showing (fast moving) high-resolution images in full color and spatial sound output of high-quality. On the input side a wide range of game controllers and pointing devices are available. Most of them are very similar in function and handling. The keyboard is still the standard way to enter text and the mouse plays a major role in interacting with different parts displayed on the monitor.

More recently, camera based approaches have been introduced as generic controllers for games. The Sony EyeToy, e.g., allows interacting in the physical space with games on a Playstation 2 [Son05].

Input Devices Since the invention of the computer mouse, a great varity of input devices has been developed. Most input and interaction devices are not as general as the mouse and hence, they are of great value in specific domains.

Recently, research and industry focus on interactions by directly observing movements or gestures of users and translating them into interaction events. However, this kind of processing needs an augmentation of the users' environment or the users themselves. Examples are presented in [VRC99],[MAS04] and [Rek01].

Our approach differs from this in that we augment existing objects or appliances that people have already got used to and know their affordances. We then aim for an easily understandable way of translating interaction with these devices into actions and events interpretable by applications. By matching the affordance of the object with the interaction to perform, we can be sure that users will quickly be able to start using the application in the intended way.

For enhancing existing appliances, we identified several different types of sensors that can be cheaply bought and easily integrated:

- **Accelerometers**: Can be used to detect absolute angles with respect to the earth's gravity field as well as dynamic acceleration along their respective axes.
- **Magnetoscopes**: Can be used to detect absolute angle with respect to earth's magnetic field and relative rotational changes.
- **Pressure Sensors**: Can be used to sense whether users hold or squeeze a device in their hands, put it into a pocket or exert pressure to initiate some action, etc.
- **Light Sensors**: Can be used to decide whether a device is put in some bag or left outside; can also give information on the time of day and the type of environment the user is in.
- **Distance Sensors**: Can be used to measure the space between two objects or the user and an object.

This is by no means an exhaustive list as there exist many more types of sensors (temperature sensors, microphone, or more general, sonic sensors, etc.). However, the sensors listed above prove to be interesting in our experiments for creating engaging and animating

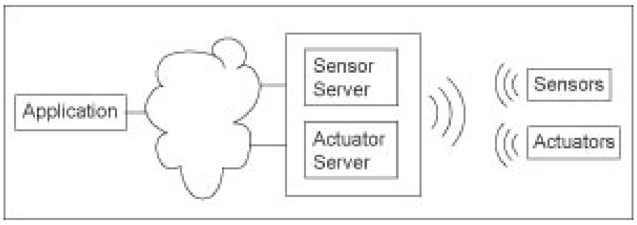


Figure 1: Visualization of the basic architecture.

input devices. In many ways users can interact with devices like a cube ([KSHS05]) or a chair ([Coh03]. This can easily be detected and interpreted.

We will show in Section 3.1 how we use a subset of the mentioned sensors to capture movements of interest of a user on a augmented IKEA balance cushion ([SHK04]).

Output Devices On the other end of the application, we observe the trend to use more and different types of screens and displays for output, depending on the type of data that is to be visualized.

We also do not narrow the term output devices to mere visual screens, but also include various means of output devices, e.g., for tactile or haptic output. Simple occurrences of showing a one bit state can include, e.g., LEDs or switching on and off any appropriate appliance.

Particularly interesting seems to be to use a combination of distributed large public displays placed at various points of interest in the environment and small private displays visible only to a specific user or group of users. Such forms of multi-display game environments have been suggested in [MMES04].

From an architectural point of view, we treat sensors and actuators similar as will be shown in Section 2.2.

2.2 Basic Architecture

As is shown in Figure 1, the application is strictly separated from any issues regarding sensor or actuator hardware. There is a clear interface to access the different devices. A realization for that (indicated by the cloud in Figure 1) will be presented later in Section 3. An application needs to send information to output devices and receive information from input devices. Communication in these two directions is aided by two helpers, namely the Sensor Server and the Actuator Server. They are similar in structure in that they have one or several registered devices with which they communicate. The Sensor Server for example collects data from all sensors known to the component. This information can then be queried and used by the application. The Actuator Server on the other hand has a list of actuators, i.e., displays, lights, etc. The application is then granted access to those via the communication layer according to the capabilities of the respective device.

Of course, it is neither needed nor sensible that every sensor sends its data to the Sensor Server by itself. In most hardware platforms, those will be collected and sent by a central component. On the other hand, this approach allows an arbitrary number of sensors / actuators in the environment to be used without needing any sophisticated knowledge in hardware or communication protocols.

2.3 Abstraction from the Hardware

To accomplish the design goal described in the last section, we must provide an abstraction from the available hardware. Especially to enable application developers and device builders to easily integrate their products into the architecture. In particular, we want to ease the job for people building and integrating hardware components and for game application developers. The field of virtual reality has brought with it some platforms and architectures that use device abstraction. Open Tracker ([RS01]), e.g., bases its tracking framework on XML and defines interfaces to easily support different devices. The Virtual-Reality Peripheral Network (VRPN) [THS$^+$01] includes a set of servers that are designed to implement a network-transparent interface between application programs and a set of physical devices. In the following we describe how our concept can simplify the development process of a real world computer application.

2.3.1 Support for Device Developers

The first thing to build a new interaction method, in the sense we use it, is to search for a suitable device and then decide upon which sensors can be integrated. Subsequently these sensors have to be connected to some hardware platform like Smart-Its [GKBS04] or Particles [DKBZ05] that supports retrieving the data, maybe combine them and send out the information.

The task of a device developer is then to enhance the Sensor Server to receive the data and make it available through a clean interface. Similar actions apply to new actuators. Basic functionality must be provided to be able to control the device. For displays, this may include writing text, drawing lines and displaying images, for others it might only mean specifying some color or switching them on or off. The implementation of these methods will of course again benefit from an abstraction mechanism that hides the need of sending sequences of high and low voltages and offers, e.g., access to each pixel.

2.3.2 Application Programmer

Somebody who wants to develop a game or some other kind of application does not want to have to care at all about hardware details, communication protocols etc. Ideally, he or she does not even need to pay attention to the type of input or output device. Much research is currently done on automatically adapting content to be able to display it equally well on a desktop PC, a smaller PDA display and on a mobile phone. Similarly, using different kinds of input devices that provide the same amount of information and therefore can be interpreted in a similar way should be interchangeable. In our example application (see Section 4.1) the game can be controlled by a new interaction device or by a standard keyboard.

We are therefore hiding as much of the hardware details as possible from the developer providing him or her with an already abstracted interpretation of raw sensor values. As an example, consider a simple ball switch and an accelerometer. The first is either 0 or 1, depending on the way it is placed. The accelerometer can convey exactly the same information when used appropriately. However, the developer will probably not be able to decide that so quickly. Therefore, there will be an abstraction and the developer can build on a set of small events and does not need to cope with raw sensor values if not needed.

2.4 Middle Layer

To be able to get this kind of architecture, a middle layer is needed that is responsible for providing easily accessible interfaces to the application and sensor sides as well as managing the communication between them.

We draw heavily on available standards to ensure that the largest possible number of applications can be used and that the learning effort for developers is minimized. As is described in more detail in Section 3.2 where our implementation of the layer is shown, we currently favor XML as the data wrapping format as it can be easily validated against, is human readable and parsers exist for most applications and programming languages.

As protocol to access the data, one of the most widely supported formats is HTTP. Nearly all applications or languages that allow some kind of external access are capable of reading web pages. The infrastructure needed can be found nearly everywhere and it is particularly easy to create small viewers for prototyping and testing devices.

3 Real-World-Interaction Architecture and Implementation

Our experiences from previous projects showed that, for application developers and designers, the integration of non-standard hardware is extremely difficult. To open up design options for interaction with the physical world, we looked for a solution to easily integrate novel input and output devices into a programming and authoring environment.

We chose Flash MX because it is commonly used by developers and designers. It is suitable for easily and quickly creating games and other small applications that are accessible using web technology. Beginning with version 2004, Flash provides an object orientated programming language called Actionscript 2.0. We therefore created an Actionscript 2.0 component that can be incorporated in any Flash application by simple drag and drop. After dragging the component onto the Flash ’stage’ one has to configure the component by setting two parameters: one for a link to the configuration URI and one for the variables URI on the server - both are described later in more detail. Afterwards the developer has access to all variables made available from the component on the main time-line (called ’root-time-line’ in Flash).

From there, these can be used like every other global variable for the application semantics. As has been shown in the previous sections, all kinds of input devices can be imagined. These produce sensor data in completely different ways. The Sensor Server is responsible to convert this data into a specific XML format (see Section 3.2). The converted data is then available to the Flash application. The same applies to actuators where the flow of actions is reversed. The application needs not be running on the same machine or server, it only needs to know the URI where the XML file is hosted or generated. Thus, it can read and understand the data from the server. The information flow is illustrated in Figure 2. It is a very powerful mechanism to enable communication between hardware device and user interface even when those are not in the same room, building or even country, enabling all sorts of remote controlled applications.

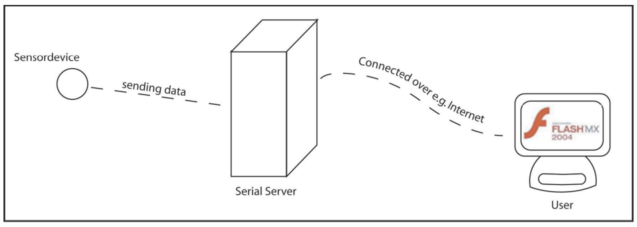


Figure 2: Setup of the Virrig game application.

To be sure that only one device (or, more general, the correct number of devices) is connected to the software, each device is assigned a specific unique ID. At the start of the program, there is a small dialog that lets the user choose which of the devices found in the environment should be chosen for a particular application. Of course, a fixed coupling can also be programmed into the application by removing the dialog and filter only those messages which originate from the device with the specified ID. Since the device IDs could be faked and the RF transmission currently is not encrypted, this does not prevent malicious interference. Systems which rely on tight security would need more sophisticated authentication and authorization mechanisms.

It makes sense to enable the coupling of one device with several programs. This provides different views on one and the same set of sensors. It is also allowed to connect several software programs with one and the same output device. This implies that changes might be overwritten by later messages. Although this might not be desirable in some cases (e.g. when controlling one light), there are many application areas where such a behavior is wanted like competitive or collaborative settings. A display used stand-alone or integrated into an output device, for example, can easily cope with input from many sources by listing them one after the other and maybe tagging them.

In this section, we describe the implementation and focus on sensors. The connection of actuator systems is similar.

3.1 Sensor Hardware Architecture

We have implemented several sensing devices which share the same basic hardware architecture. The sensors are connected to a micro-controller. The micro-controller is responsible for basic data acquisition and processing. Via RF, the data is then sent to a base unit that is connected to a computer in the network.

In one implementation we used the Smart-Its platform [GKBS04] and attached a custom sensor board. The Smart-Its provide a programmable micro-controller (PIC 18F452) and several analog as well as digital inputs and outputs. Sensor data can be sampled at frequencies of several hundred Hertz (if supported by the sensor). The RF sender can send data at a maximum rate of 14400 bps (including overhead for control, etc.) enabling even those applications that rely on quick updates. This data is then transferred wirelessly using a transceiver of the type Radiometrix SPM 2-433-28. At the PC side, a similar construct is used: Another SPM module receives the data and communicates it to the PC via serial line input. From there, it can be processed by software as is described in further detail in Section 3.2.

In a new implementation we used Particles, devices similar to Smart-Its. Particles are developed at TecO, Karlsruhe [DKBZ05]. They are smaller and have more sophisticated ways of transmitting data (including acknowledgment etc.). This change has shown one of the strengths of our architecture: since input devices are separated from processing and the final application, it has been very easy to switch to the new platform. Only the receiving part of the communication server had to be adjusted. The data is no longer sent over serial input but in UDP packets.

Apart from the loose coupling to the used sensor and actuator hardware, the framework also abstracts from the actual object that is used as input device and into which we embed the sensors. We deliberately searched for objects that are known to most people, but are not yet used as input devices. In this section, two out of the many possibilities we found are presented. First, we used an IKEA balance cushion named Virrig shown in Figure 3 [SHK04]. It is a flat cushion mounted on a robust hemisphere. Thus, it can be rotated and tilted in all directions. It is very flexible in use as the user can sit, stand, kneel or lie on it and it is very robust, too, as it is designed for use by children. It can be seen as a regular cushion or as a toy to practice balance.

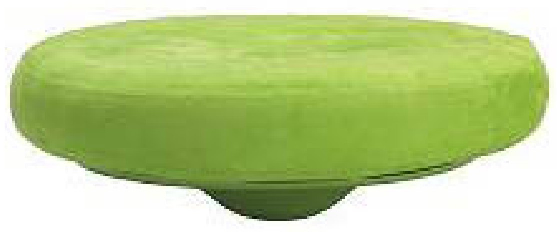


Figure 3: The Virrig input device shown from the side.

The digital device we attached inside the hemisphere does not change the affordance or the physicality of the cushion. The user still can sit or stand on it as before, and since we use radio technology for data transmission there are no cables leaving it, so it can still be tilted and rotated like before. We show how to use the cushion in an edutainment application in Section 4.1.

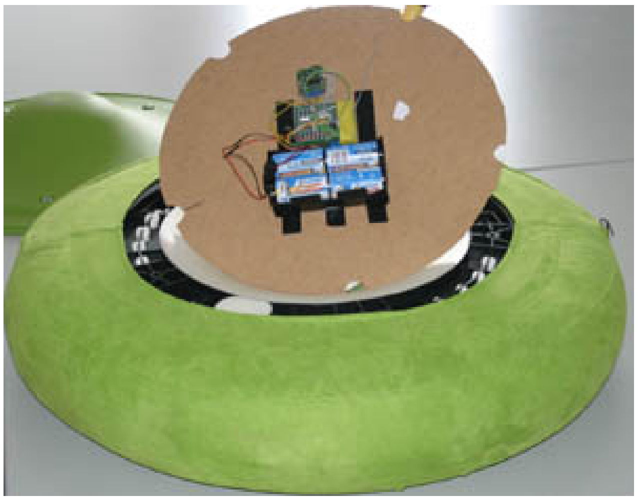


Figure 4: Opened Virrig with the integrated Smart-Its.

Inside the cushion, attached to a wooden plate, there is a Smart-Its (see Figure 4) with the following components:

- 4 large batteries are used as power supply; this ensures that the device needs not be opened even for long term user studies and during everyday use
- 4 ball switches indicate the tilt of the cushion in 8 directions
- a compass that shows relative rotational movements as well as the absolute rotation of the cushion
- a pressure sensor is used to sense if a user is currently sitting on the cushion and detect his or her movements
- a radio transmitter sends the data to a receiver connected to the PC

The overall hardware architecture is depicted in Figure 5.

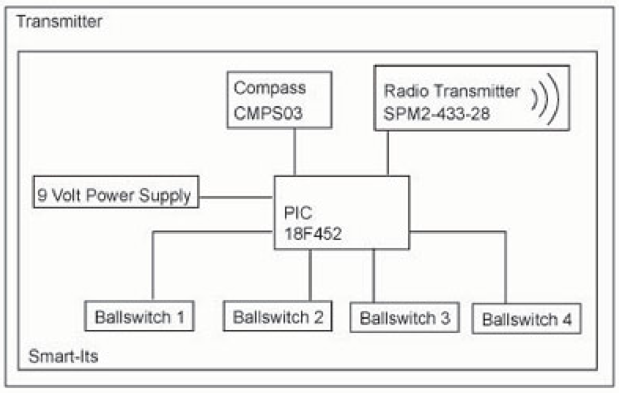


Figure 5: Hardware architecture of the input device.

As another input device, we chose a small appliance that is mounted at the doors of many offices, lecture halls or assembly rooms. It shows the current state of the room or its occupant. A magnetic pin is placed on certain spots of a ferromagnetic plate to indicate whether the person working in that room is in, busy, out for lunch, etc. We enhanced such boards with magnetic switches that recognize where the button / pin is placed and put a Smart-Its in each of them that communicate this state to a central receiver. The application is briefly described in Section 4.2.

3.2 Sensor Server / Communication Server

After having described the parts connected to the input device, this section goes into details about the PC side of the system. A Smart-Its equipped with a radio receiver is attached to the PC. The Smart-Its receives sensor data from the cushion and forwards it to a program called Serial Server over RS232 serial line. The Serial Server interprets all received signals and transforms them into XML format. This data is then stored on a web server making it available for any application capable of using the HTTP protocol.

The architecture of the receiver is outlined in Figure 6. As has already been stated, we believe that providing information via the HTTP protocol is one of the best methods to allow a very high number of different applications as well as programming languages easy access to this data. The choice of using XML as storage and wrapping format has been made in the same sense. A large number of applications and programming languages inherently support reading and writing data coded in XML structures. The fact that the data is human readable and conveys some semantics in its structure and named tags helps to implement applications using devices for which there is no internal knowledge. It also enables the specification of content structure and easy validation of incoming data. The part that generates sensor values as well as the application can therefore rely on a specific DTD or schema being followed by transmitted sensor data. This dramatically reduces the complexity of implementing both sides. In our prototypes we did not experience any significant slow-down using XML instead of, for example, using a file with comma separated values (CSV). Since, however, some system resources are needed for parsing and processing of XML data, this is not the optimal solution for devices with very low processing power. The first step toward tackling this problem is to send only that part of the data that really changed (event based mechanism). Additionally, we are planning on making the protocol XML exchangeable with any other protocol like CSV or binary formats.

3.3 Component and Test Application

To explain how to use the Flash component and how to build applications on top of it we provide a small sample application that outlines the basic concepts in more detail. After that, a more sophisticated program is described for which we plan to undertake user studies to evaluate some assumptions on the impact of physical interaction in learning applications.

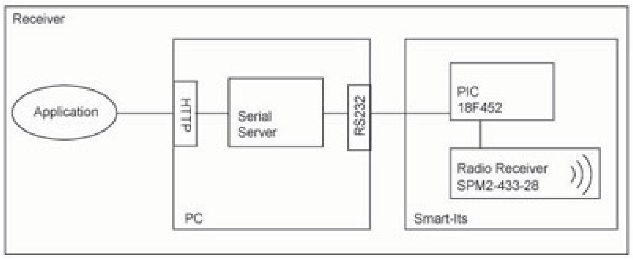


Figure 6: Hardware architecture of the receiver.

The component itself requires two input parameters: the locations (URIs) of the XML configuration file and the XML variables file. The XML configuration file specifies the variables for faster parsing in the Flash application. The component reads the file so it can name and initialize variables and set an interval for the reload rate.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE eventlist SYSTEM "configdtd" >
<config>
 <interval>
   <min unit= "sec" value="1" />
   <max unit= "sec" value="3" />
 </interval>
 <vars>
   <var name="rotation" startvalue="0" />
   <var name="left"     startvalue="0" />
   <var name="right"    startvalue="0" />
   <var name="up"       startvalue="0" />
   <var name="down"     startvalue="0" />
 </vars>
</config>
```

Tag	**Tag Explanation**
<interval>	minimal and maximal sensor refresh rate in seconds
<vars>	block with available variables
<var>	names, start values and types of the variables delivered by the sensor input device

XML Variables File: This file delivers the sensor data to the application. Only the variables that have different values from the last update appear so that the component does not need to read all the variables each time. This renders the Flash application notably faster and also reduces traffic.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE eventlist SYSTEM "vars.dtd" >
<changedVars senderId="...">
 <var name ="rotation" value="30" />
 <var name ="left"     value="1" />
</changedVars>
```

Tag	**Tag Explanation**
<changedVars>	variables whose value changed
<var>	names and values of the variables delivered by the sensor device

When the application starts, the first thing for the component is to analyze the configuration file, to set up and initialize the variables and to set the minimum and maximum interval for reading the XML variables file. Then the application starts reading the variables for the first time. From now on it reads the variables file as often as defined by the minimum interval in the configuration file. It sets the received variables onto the 'root-time-line'. That process is repeated as long as the application is running. This is also shown in the activity diagram in Figure 7. For the rare case of sensors dynamically changing their update interval, the configuration file could be read from time to time to update the minimum nd maximum rate.

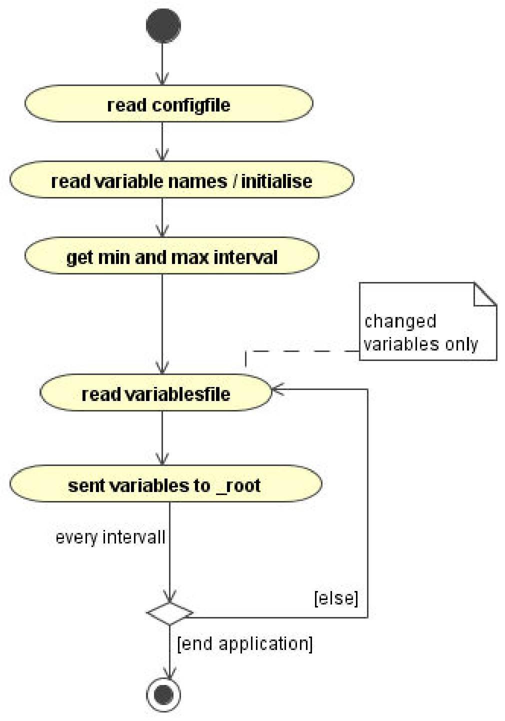
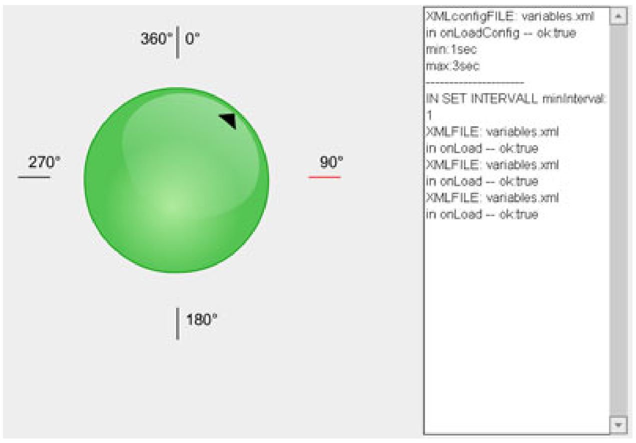


Figure 7: Activity diagram of the Flash component.

After programming the component, we implemented the first test application. This application visualizes movements made by the Virrig cushion. The application shows the cushion in the center of the window and indicates tilt and rotation. All possible movements of the input device are shown on the screen. A screen dump of the application is shown in Figure 8. The green ball in the middle represents the Virrig cushion. The arrow (black triangle in the top right in Figure 8) indicates the rotation sent by the sensor device. The tiny black lines around the ball show, if painted red, the activated ball sensors. The text window on the right is an output window for testing and tracing the correct functionality of the component.



```
XMLconfigFILE: variables.xml
in onLoadConfig -- ok:true
min:1sec
max:3sec
-----------------------
IN SET INTERVALL minInterval:
1
XMLFILE: variables.xml
in onLoad -- ok:true
XMLFILE: variables.xml
in onLoad -- ok:true
XMLFILE: variables.xml
in onLoad -- ok:true
```

Figure 8: Screen dump of the test application.

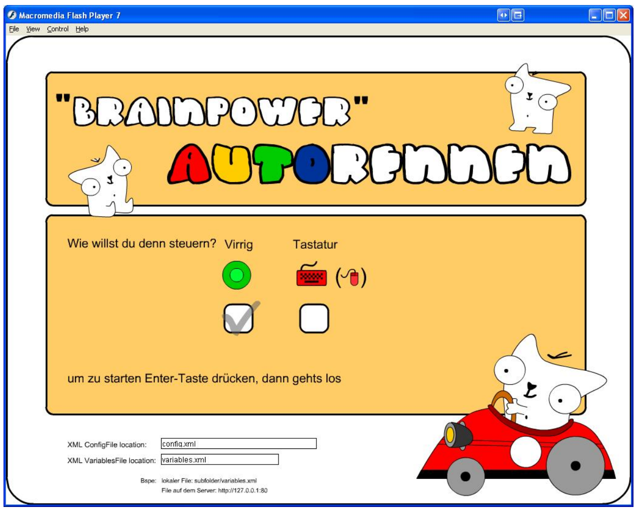
4 Sample Applications

In this section we introduce some applications based on the introduced architecture demonstrating the feasibility of our approach.

4.1 Virrig Race Game

The Virrig Race Game is a game application with the sensor cushion as physical controller. It is a mixture of car race and learning program, i.e. an edutainment application. The car is controlled by the cushion. Since the expected audience will be primary school children, the whole design is aimed to be interesting for smaller kids. Depending on the complexity and type of the questions, the game is also fun and a nice learning experience for people of all ages.

The first step in integrating the cushion into the framework is to augment the Sensor Server such that it converts the sensor values it receives into an XML tree. Second, the flash component used as described in Section 3.3. The flash application can now use the given sensor values to define its behavior. As one implementation, we designed it the following way: The user controls the car by tilting the cushion forward and backward to accelerate or brake and left or right to go in that direction, respectively (see Figure 10). There are stop signs on the crossings where a question is displayed (see Figure 11). In this case, the rotational sensor (compass) is used to implement a simple selection mechanism by turning the Virrig. The user then commits the choice by clicking, i.e. tilting forward. One could also think of realizing the this by means of hopping on the cushion as a load sensor is placed directly below the user in the inner part of the cushion.



Wie willst du denn steuern? Virrig [✓] Tastatur

um zu starten Enter-Taste drücken, dann gehts los

XML ConfigFile location: config.xml
XML VariablesFile location: variables.xml

Bspe: lokaler File: subfolder/variables.xml
File auf dem Server: http://127.0.0.1:80

Figure 9: Start screen of the game application. The user can choose to control the game with a standard keyboard or with the Virrig.

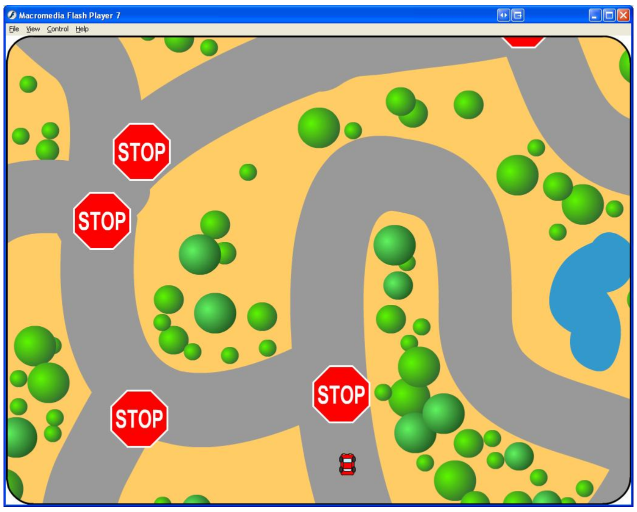
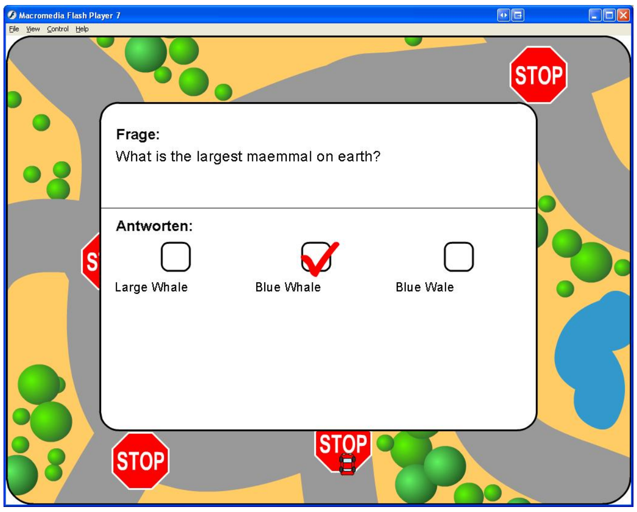


Figure 10: Screen dump of the game application.

The user can continue driving after giving a (potentially wrong) answer. If the answer was wrong, the sign gets transparent until he or she has answered another question. So the memory effect is not that serious but imposes at least some pressure on the user who can now try to answer it again. If the question was answered correctly this time, the sign disappears. After all signs have disappeared (i.e. all correct answers have been given) the race has been successfully completed. It could be imagined to display a score board showing the 10 fastest users or those with the least number of wrong answers.

The framework makes it particularly easy for developers to quickly change the interpretation of the sensor values. The compass could for example be used as sensor controlling instead of tilting. It also enables exchanging the particular device that is producing the sensor values or is consuming the software input with another device, possibly with another set of sensors / actuators.



Frage:
What is the largest maemmal on earth?

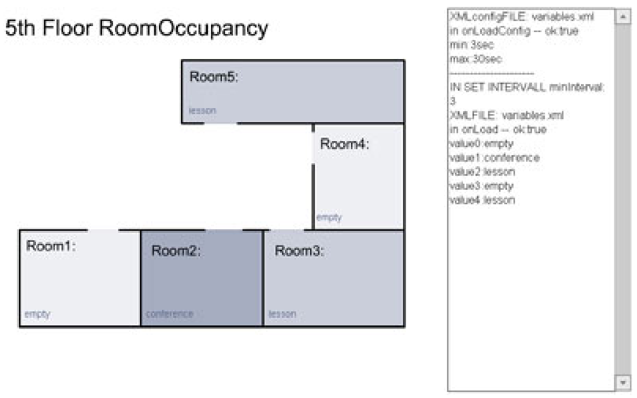
Antworten:
Large Whale [✓ Blue Whale] Blue Wale

Figure 11: Screen dump of the quiz window showing a question on top and three possible answers below. The one in the center has been selected by the user.

4.2 Room Occupancy System

This application visualizes the occupation of rooms in a building, e.g. if the room is used for a meeting, a lesson, or is empty (see Figure 13). The natural choice for controlling such an application is an input device placed at the entrance to rooms. This device is very intuitive to use since a simple version of it can already be found at many office doors. It is a board with switches that are activated by magnets (see Figure 12). The state of a room is set by putting a magnet on the sensitive area of the Hall-effect switch. This data is wirelessly transmitted per room to the Serial Server. The server collects the information of each room and the application get this information from a specific IP and port from the HTTP server.

As can be seen in Figure 12, the device is also equipped with four LEDs that serve as feedback when setting the magnets but can also be controlled remotely. Thus, one could replace the magnets with simple buttons and the LEDs could show the current state of the office. This state can then be changed by, e.g., the same Flash application that shows the state of the room. It simply needs to write a specific value coupled with those LEDs and the Actuator Sensor takes over to pass the commands to the hardware component.



```
XMLconfigFILE: variables.xml
in onLoadConfig -- ok:true
min:3sec
max:30sec
--------------------
IN SET INTERVALL minInterval:
3
XMLFILE: variables.xml
in onLoad -- ok:true
value0:empty
value1:conference
value2:lesson
value3:empty
value4:lesson
```

Figure 12: Screen dump of the Room Occupation System.

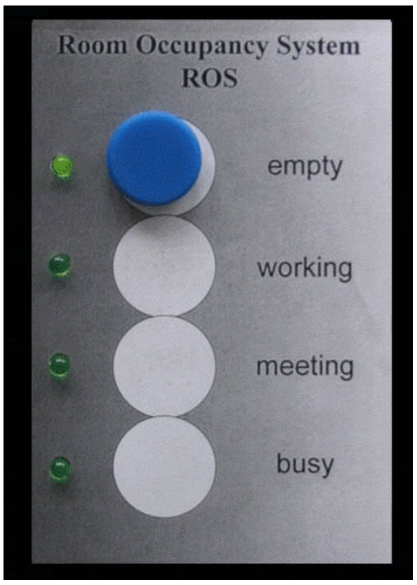


Figure 13: Picture of the Room Occupation System user interface.

5 Related Work

Tangible User Interfaces (TUIs) have been subject to scientific research for a long time. The initial works by Ullmer and Ishii [IU97] and others show the high potentials of intuitive, physical interfaces for human-computer interaction. Holmquist et al. in [HSU04] present a comprising overview of recent tangible user interfaces.

ENI (for example used in [Gro02] is an event and notification infrastructure using a client-server environment. It allows attaching sensors and indicators directly to the client or enables them to communicate with the server via a web-based interface. Adding functionality of new devices requires writing plug-ins and changing existing ENI clients or creating new ones. The programming language needed to support all features is fixed to be Java. The application of the proposed architecture is similar to our work, but oriented towards a collaborative scenario targeted at group awareness and not for making sensor data of physical and tangible user interfaces available to application programmers as presented in this work.

With iStuff, Ballagas et al. [BRSB03], present another flexible prototyping platform for explicit, post-desktop interaction. iStuff allows multiple, co-located users to interact with applications and displays in so-called augmented environments such as the Stanford iRoom. As our approach, their system is event-based and allows dynamic re-mapping of devices to events.

Greenberg in [Gre02] discusses the problems of making physical user interfaces available to application programmers, which is still a challenging task. He compares the difficulties for accessing physical user interfaces to those the first GUI developers had – building everything completely new from scratch. By the proposed architecture we show application programmers a simple way of managing this problem.

Collabolla, presented by Bove et al. [BPW05], is a physical interface for arcade games. Players sit on inflatable balls, so-called Space Hoppers, and by hopping, jumping and rolling control their game characters. As with the Virrig, the playfulness and the affordances of the physical input device are exploited for the game controller.

6 Conclusions

In this paper we have introduced an approach that helps prototyping physical interaction. Our focus is on the support for games. The examples given concentrate on physical input devices and their integration into Macromedia Flash MX. The abstractions provided aim at easing the task for hardware developers and game developers alike by providing a suitable middleware.

By encapsulating the access to physical devices into a component in Flash we see that implementations become much simpler. Providing sensors and actuators as variables makes their physical distribution transparent to the developer. The simple way of exchanging the information via XML and HTTP makes new developments very simple as developers can use libraries already available. For some gaming domains the proposed solutions has restrictions as the time delay between the occurrence of a manipulation in the real world and availability of the data in Flash can take up to 200 ms. But even given this time delay games that need 'immediate' reaction can be prototyped with the infrastructure described. Since some applications are time critical and cannot live with such a delay, we are currently optimizing data transfer, e.g., letting the Flash application query for its data directly via TCP and skip the HTTP server.

We are using this setup in other projects that we are running. Especially colleagues and students who are new to embedded device programming are very pleased to be able to use devices and sensors without having to get deeper knowledge about controlling them.

Currently we are preparing a version of the Virrig Race Game to perform a user study with children. In future work we want to qualitatively and quantitatively assess the advantages of physical controls for edutainment systems.

Another project we are currently working on is concerned with using the same technology of the Virrig cushion. A similar device called therapy top is used in professional sport schools for training and convalescence purposes. Athletes and patients have to perform specific exercises defined by professional therapists. It is very important that these exercises be done correctly. To support trainees without much supervision effort, we are developing a training application where people get immediate audio-visual feedback on their actions. With the concept we presented the underlying logic can be realized very quickly.

7 Acknowledgments

The work has been conducted in the context of the research project Embedded Interaction ('Eingebettete Interaktion') and was funded by the DFG ('Deutsche Forschungsgemeinschaft').

References

[BPW05] Jennifer L. Bove, Simone Pia, and Nathan Waterhouse, *Collabolla*, `http://people.interaction-ivrea.it/s.pia/collabo_1.htm`, 2003, visited 21/10/2005.

[BRSB03] Rafael Ballagas, Meredith Ringel, Maureen Stone, and Jan Borchers, *iStuff: a Physical User Interface Toolkit for Ubiquitous Computing Environments*, CHI '03: Proceedings of the SIGCHI conference on Human Factors in Computing Systems (New York, NY, USA), ACM Press ISBN 1-58113-630-7, 2003, pp. 537–544.

[Coh03] Michael Cohen, *The Internet Chair*, International Journal of Human-Computer Interaction **15** (2003), no. 2, 297–311.

[DKBZ05] Christian Decker, Albert Krohn, Michael Beigl, and Tobias Zimmer, *The Particle Computer System*, Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks (IPSN), April 2005, pp. 443–448.

[GKBS04] Hans-Werner Gellersen, Gerd Kortuem, Michael Beigl, and Albrecht Schmidt, *Physical Prototyping with Smart-Its*, IEEE Pervasive Computing Magazine **3** (2004), no. 3, pp. 74–82, ISSN 1536–1268.

[Gre02] Saul Greenberg, *Rapid Prototyping of Physical User Interfaces*, Proc. Graphics Interface, invited talk, May 2002.

[Gro02] Tom Gross, *Ambient Interfaces in a Web-based Theatre of Work*, Proceedings, 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002, pp. 55–62.

[GSK$^+$02] Jonathan Green, Holger Schnädelbach, Boriana Koleva, Steve Benford, Tony Pridmore, Karen Medina, Eric Harris, and Hilary Smith, *Camping in the Digital Wilderness: Tents and Flashlights as Interfaces to Virtual Worlds*, Conference on human Factors in Computing Systems CHI'02, Extended Abstracts on Human Factors in Computing Systems, ACM Press ISBN 1-58113-454-1, 2002, pp. 780–781.

[HSU04] Lars Erik Holmquist, Albrecht Schmidt, and Brygg Ullmer, *Tangible Interfaces in Perspective: Guest Editors' Introduction*, Personal Ubiquitous Computing **8** (2004), no. 5, pp. 291–293.

[IU97] Hiroshi Ishii and Brygg Ullmer, *Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms*, CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press, 1997, pp. 234–241.

[KSHS05] Matthias Kranz, Dominik Schmidt, Paul Holleis, and Albrecht Schmidt, *A Display Cube as a Tangible User Interface*, In Adjunct Proceedings of the Seventh International Conference on Ubiquitous Computing (Demo 22), September 2005.

[MAS04] Christian Metzger, Matt Anderson, and Thad Starner, *FreeDigiter: A Contact-Free Device for Gesture Control*, ISWC '04: Proceedings of the Eighth International Symposium on Wearable Computers (ISWC'04), IEEE Computer Society, 2004, pp. 18–21.

[MMES04] Carsten Magerkurth, Maral Memisoglu, Timo Engelke, and Norbert Streitz, *Towards the Next Generation of Tabletop Gaming Experiences*, Proceedings of the 2004 Conference on Graphics Interface GI'04, Canadian Human-Computer Communications Society ISBN 1-56881-227-2, 2004, pp. 73–80.

[Nie05] Jakob Nielson, *Killing time is the killer application*, TheFeature: It's all about the mobile internet, `http://www.thefeature.com/article?articleid=8183`, visited 21/10/2005.

[Rek01] Jun Rekimoto, *GestureWrist and GesturePad: Unobtrusive Wearable Interaction Devices*, ISWC '01: Proceedings of the 5th IEEE International Symposium on Wearable Computers, IEEE Computer Society, 2001, p. 21.

[RS01] Gerhard Reitmayr and Dieter Schmalstieg, *An Open Software Architecture for Virtual Reality Interaction*, VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology (New York, NY, USA), ACM Press, 2001, pp. 47–54.

[RSH$^+$02] Yvonne Rogers, Mike Scaife, Eric Harris, Ted Phelps, Sara Price, Hilary Smith, Henk Muller, Cliff Randell, Andrew Moss, Ian Taylor, Danae Stanton, Claire O'Malley, Greta Corke, and Silvia Gabrielli, *Things aren't What They Seem to Be: Innovation Through Technology Inspiration*, Proceedings of the Conference on Designing Interactive Systems DIS'02, ACM Press, ISBN 1-58113-515-7, 2002, pp. 373–378.

[SHK04] Albrecht Schmidt, Paul Holleis, and Matthias Kranz, *Sensor Virrig - A Balance Cushion as Controller*, Workshop Playing with sensors on UbiComp 2004, September 2004.

[Son05] Sony, *Eye-Toy*, `http://www.eyetoy.com`, visited 21/10/2005.

[THS$^+$01] Russell M. Taylor, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser, *VRPN: a Device-independent, Network-transparent VR Peripheral System*, VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology (New York, NY, USA), ACM Press, 2001, pp. 55–61.

[VRC99] Andrew Vardy, John Robinson, and Li-Te Cheng, *The WristCam as Input Device*, Proceedings of the 3rd IEEE International Symposium on wearable Computers ISCW'99, IEEE Computer Society ISBN 0-7695-04280, 1999, pp. 199–202.

Citation

Paul Holleis, Matthias Kranz, Anneke Winter, and Albrecht Schmidt, *Playing with the real world*, Journal of Virtual Reality and Broadcasting, 3(2006), no. 1, April 2006, urn:nbn:de:0009-6-2948, ISSN 1860-2037.