Playing with the Real World

Paul Holleis, Matthias Kranz, Anneke Winter, Albrecht Schmidt
Research Group Embedded Interaction
Amalienstrasse 17
80333 Munich, Germany
{paul,matthias,anneke,albrecht}@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 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. A Flash component can now access the data that is stored in XML format and directly use it in the application. This component can be easily integrated into any Flash application.

Keywords

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

1. INTRODUCTION

Gaming is and always was ubiquitous and pervasive: children play card games on long travels in the back seat of the car, teenager use their high-end personal computers and business people play with their high-sophisticated mobile device - as stated in [6] 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 sever is a lightweight web server for that can be accessed via the HTTP protocol which is available to a great variety of programming platforms and languages.

On the gaming application side, we demonstrate the integration 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, preprocessing 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, especially design, electrical layout or hardware used, is needed by the application programmer. These facts are completely hidden and shielded.

The main structure of the paper is as follows: We first motivate the need for novel input devices in Section 2 where we also discuss some input and output modalities. The Flash component is presented in detail in Section 3. We present the implementation of the surrounding architecture that communicates between raw sensor values of the input device and the applications that are going to use them in Section 4. In Section 5 we finally show some example programs which we have implemented to test and evaluate the system.

2. PROGRAMMING BEYOND THE DESKTOP

The focus of traditional programs and especially games has primarily been on running these applications on standard computer, mobile devices, and game consoles. Such a device unified input and output devices (e.g. in the case of a standard PC mouse, keyboard and screen). Similarly with the emergence of multi-player games and later internet-based online gaming platforms the computer screen and speakers remain the only output device – besides force feedback controls – and still mouse, keyboard and joystick are 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 computing gaming, novel interaction devices like a torche [3], a flying jacket [8] and or a cushion [9] 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 been largely 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 allow interaction in the physical space with games on a PS2 [11].

Input Devices

Since the invention of the computer mouse, developed in 1970 by Douglas Engelbart [1], a great varity of input devices has been developed. Most input and interaction devices are not a general as the mouse and hence they are of great value in specific domains.

Recently a focus is on interacting by directly observeing the movements or gestures of a user and translate into interaction events. However, this kind of processing needs either an augmentation of the user's environment or the user. Examples are presented in [13],[5] or [7].

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 the interaction to perform , we can be sure that users will quickly be able to start using the application.

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

- **Compass**: Can be used to detect absolute orientation on the surface with respect to the north pole or the change in orientation.
- **Accelerometers**: Can be used to detect orientation along their respective axes and (in some restricted way) rotational as well as translational (quick) movements.
- **Gyroscopes**: Can be used to detect absolute angle with respect to earth magnetic field and relative rotational changes in a horizontal plane.

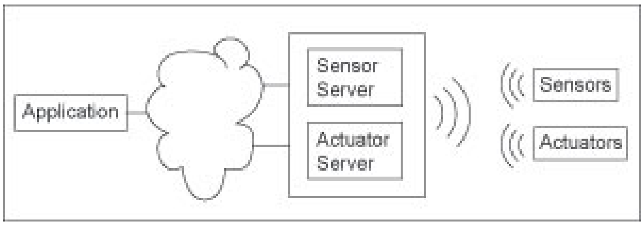

Figure 1: Visualization of the basic architecture.

- **Pressure Sensors**: Can be used to sense whether a user holds or squeezes a device in his hands, has put it in a pocket or exerts pressure to initiate some action, etc.
- **Light Sensors**: Can be used to decide whether a device is put in some bag or outside; can 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.). The sensors listed above however prove to be interesting in our experiments for creating Engaging and animating input devices. In many ways users can interact with devices like a cube or a chair. This can be detected and interpreted easily.

We will show in Section 4.1 how we use a subset of the mentioned sensors to capture movements of interest of a user on a augmented IKEA balance cushion ([9]).

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 of any appropriate appliance.

Particularly interesting seems to be to use of 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 [4].

From a 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 4. 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.

2.3.1 Support for Sensor and Actuator 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 [2] or Particles [12] 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 5.1) 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 correctly. 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 4.2 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 COMPONENT FOR FLASH MX

Our experienced from previous project 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 a 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 on he or she can use them like every other global variable for the application. 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 4.2). The converted data is then available to the Flash application. 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.

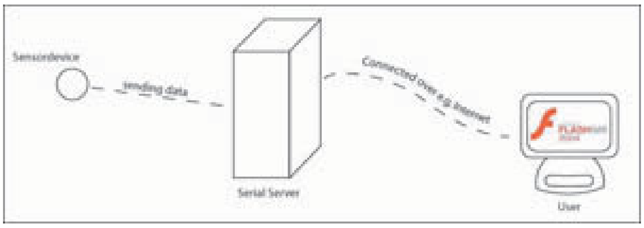

Figure 2: Setup of the Virrig game application.

4. IMPLEMENTATION

In this section we described the implementation and focus on sensors. The connection of actuator systems is analogous

4.1 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 does the 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 the Smart-Its platform [2] platform 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 Radiometric 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 4.2.

In a new implementation we used the Particles, similar devices as the Smart-Its. Particles are developed at TecO [12]. They have the advantage of being much smaller and having 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.

Independent from that is 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 [9]. 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 5.1.

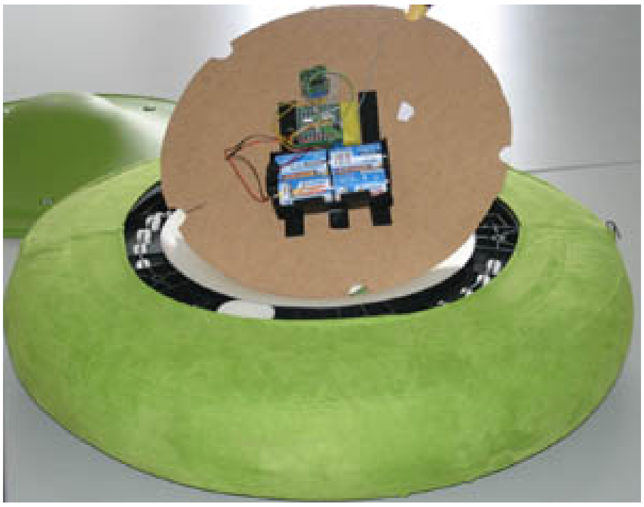

Figure 4: Opened Virrig with the integrated Smart-It.

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
- 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.

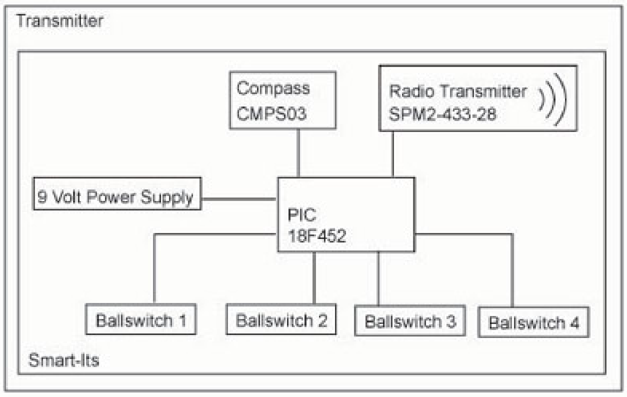

Figure 5: Hardware architecture of the input device.

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 5.2.

4.2 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 ([10]) 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. It also enables the specification of content structure and easy validation of incoming data. The sensor value generating part as well as the application can therefore rely on a specific DTD being followed by transmitted sensor data. This dramatically reduces the complexity of implementing both sides.

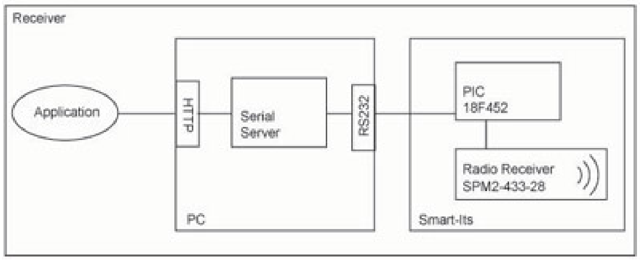

Figure 6: Hardware architecture of the receiver.

4.3 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 oulines the basic concepts in more detail. Afterwards, 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.

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 "configuredtd" >
<config>
 <interval>
   <min unit= "sec" value="1" />
   <max unit= "sec" value="3" />
 </interval>
 <vars>
   <var name="rotation" startvalue="0" type="integer" />
   <var name="left"     startvalue="0" type="boolean" />
   <var name="up"       startvalue="0" type="boolean" />
   <var name="down"     startvalue="0" type="boolean" />
 </vars>
</config>
```

Tag	**Tag Explanation**
<interval>	minimal and maximal rate in seconds rates how often the sensor data has to be refreshed
<vars>	block with the 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 "variables.dtd" >
<changedVars>
 <var name ="rotation" value="30" />
 <var name ="left"     value="1" />
 <var name ="right"    value="0" />
 <var name ="up"       value="0" />
 <var name ="down"     value="0" />
</changedVars>
```

Tag	Tag Explanation
<changedVars>	block with the changed variables
<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.

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 from 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.

5. SAMPLE APPLICATIONS

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

5.1 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 each age.

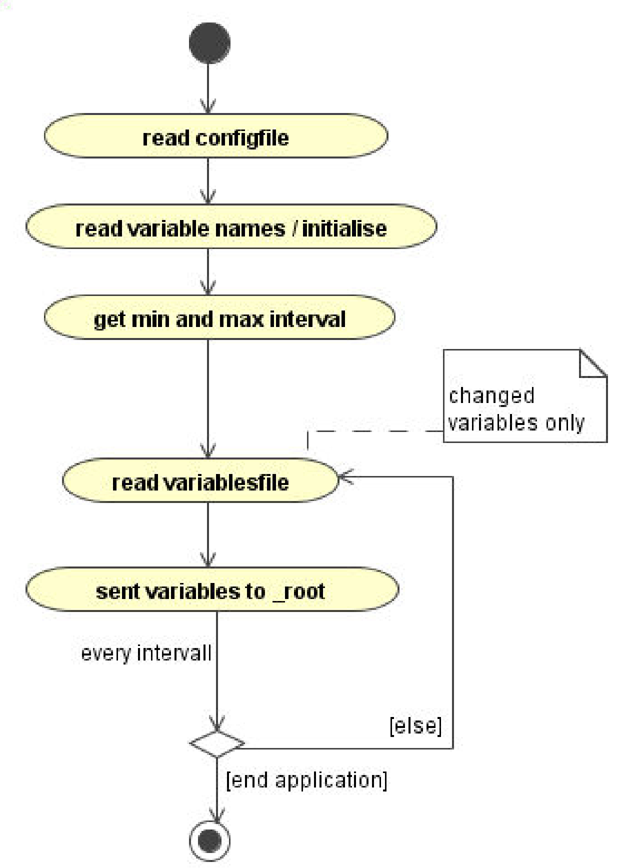

Figure 7: Activity diagram of the Flash component.

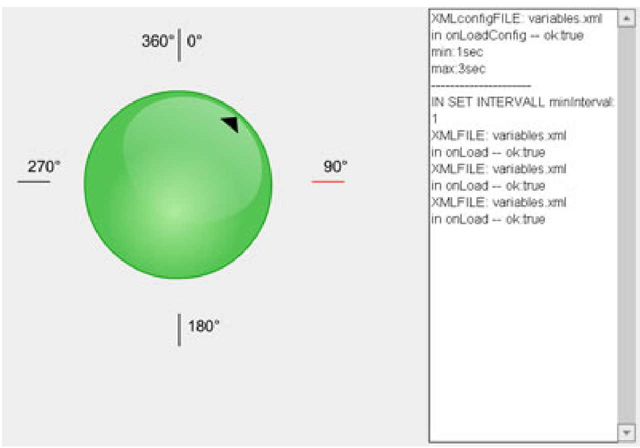

Figure 8: Screen dump of the test application.

The user interface has been designed as follows: First the user chooses the mode to play: using the Virrig or the keypad as input device. This choice is confirmed by either pressing the space bar or by tilting forward (see Figure 9 for a screen dump of the startup screen). 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 the car automatically stops when you approach them. A window then appears displaying a question (see Figure 11). Here the user chooses one answer by rotating the Virrig. The user commits the choice by clicking, i.e. tilting forward. One could also think of realizing the clicking by hopping on the cushion as a load sensor is placed directly under the user in the inner part of the cushion.

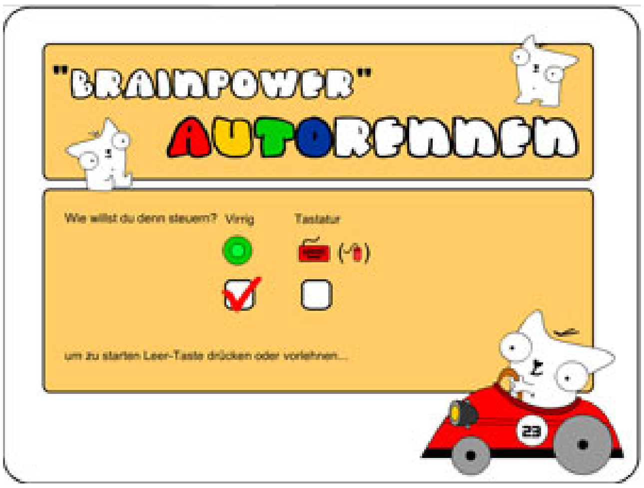

Figure 9: Start screen of the game application.

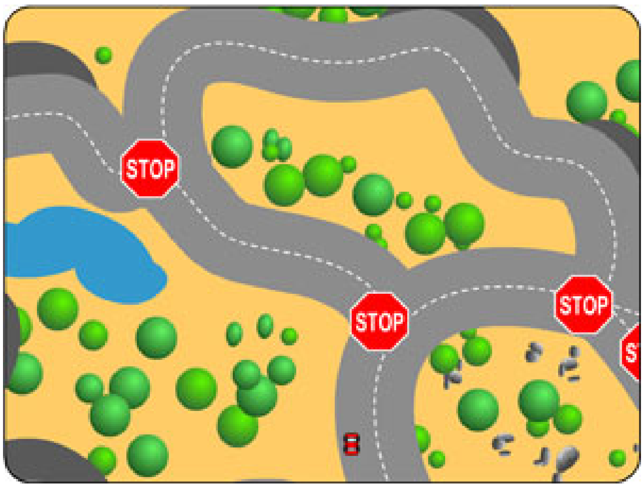

Figure 10: Screen dump of the game application.

The user can continue driving after giving a (potentially wrong) answer. If the answer was wrong, the sign gets transparent till 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 are score board displaying the 10 fastest users or those with the least number of wrong answers.

5.2 Further Applications

Another application shows the occupation of rooms in a building, e.g. if the room is used for a meeting, a lesson

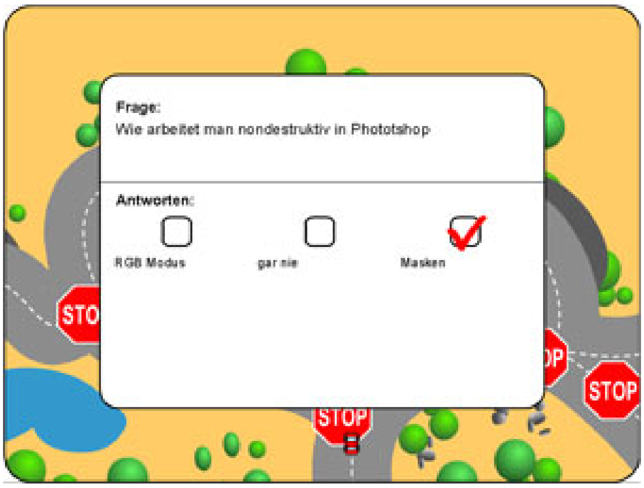

Figure 11: Screen dump of the quiz window.

or is empty (see Figure 13). The natural choice for controlling such an application is a input device situated 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.

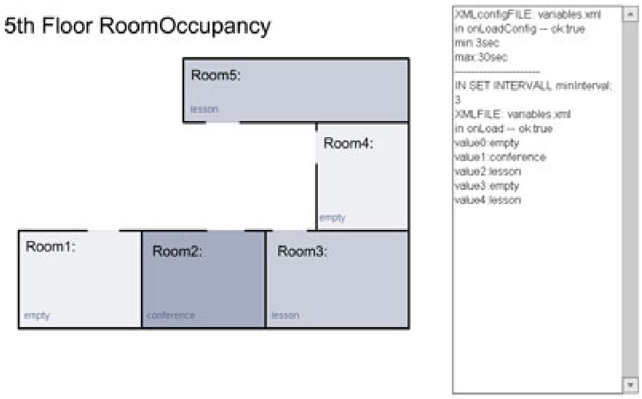

Figure 12: Screen dump of the room occupation system.

6. CONCLUSIONS

In this paper we have introduce 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

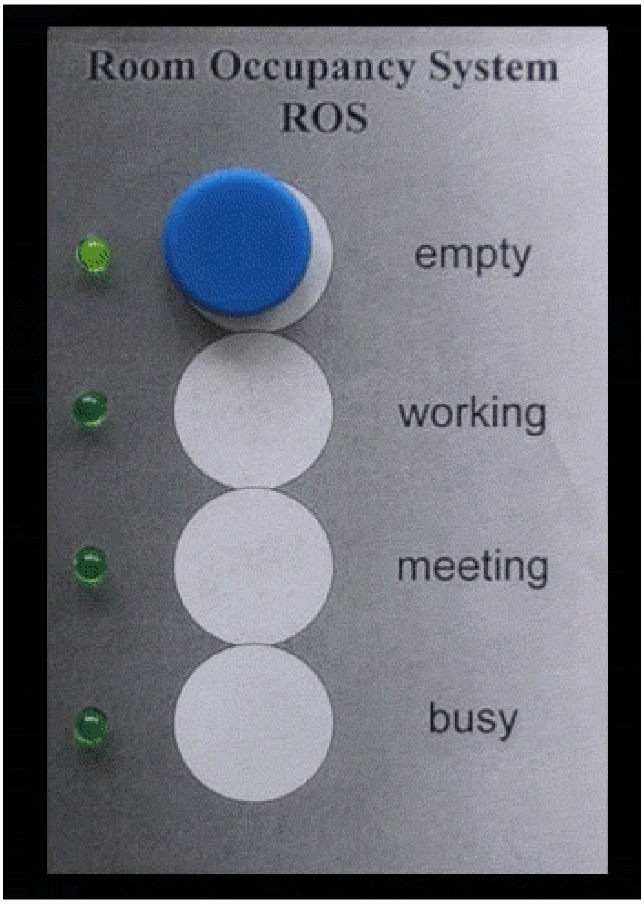

Figure 13: Picture of the Room Occupation System user interface.

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 200ms. But even given this time delay games that need "immediate" reaction can be prototyped with the infrastructure described.

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.

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').

8. REFERENCES

[1] D. Engelbart. X-y position indicator for a display system. U.S. Patent # 3,541,541, http://sloan.stanford.edu/MouseSite/1968Demo.html.

[2] H.-W. Gellersen, G. Kortuem, M. Beigl, and A. Schmidt. Physical prototyping with Smart-Its. *IEEE Pervasive Computing Magazine*, 3(3):74–82, July–September 2004.

[3] J. Green, H. Schnädelbach, B. Koleva, S. Benford, T. Pridmore, K. Medina, E. Harris, and H. Smith. Camping in the digital wilderness: tents and flashlights as interfaces to virtual worlds. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 780–781. ACM Press, 2002.

[4] C. Magerkurth, M. Memisoglu, T. Engelke, and N. Streitz. Towards the next generation of tabletop gaming experiences. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 73–80. Canadian Human-Computer Communications Society, 2004.

[5] C. Metzger, M. Anderson, and T. Starner. Freedigiter: A contact-free device for gesture control. In *ISWC '04: Proceedings of the Eighth International Symposium on Wearable Computers (ISWC'04)*, pages 18–21. IEEE Computer Society, 2004.

[6] J. Nielson. Killing time is the killer application. TheFeature: It's all about the mobile internet, http://www.thefeature.com/article?articleid=8183, 2000.

[7] J. Rekimoto. Gesturewrist and gesturepad: Unobtrusive wearable interaction devices. In *ISWC '01: Proceedings of the 5th IEEE International Symposium on Wearable Computers*, page 21. IEEE Computer Society, 2001.

[8] Y. Rogers, M. Scaife, E. Harris, T. Phelps, S. Price, H. Smith, H. Muller, C. Randell, A. Moss, I. Taylor, D. Stanton, C. O'Malley, G. Corke, and S. Gabrielli. Things aren't what they seem to be: innovation through technology inspiration. In *DIS '02: Proceedings of the conference on Designing interactive systems*, pages 373–378. ACM Press, 2002.

[9] A. Schmidt, P. Holleis, and M. Kranz. Sensor virrig - a balance cushion as controller. UbiComp 2004 - Workshop 'Playing with sensors', 2004.

[10] A. Schmidt, M. Kranz, and P. Holleis. Research Group Embedded Interaction Serial Server Project. http://www.hcilab.org/resources/webserver.htm.

[11] Sony. Eye-Toy. http://www.eyetoy.com.

[12] U. K. T. Telecooperation Office. Research group embedded interaction particles project. http://particle.teco.edu/.

[13] A. Vardy, J. Robinson, and L.-T. Cheng. The wristcam as input device. In *ISWC*, pages 199–202, 1999.