

# Integrating Sensor Data into Flash MX 2004

Anneke Winter

April 2005

## 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, Flash.

## Publications:

Workshop PerGames 2005, held in conjunction with Pervasive 2005, Munich, Germany.

## Contact:

[anneke@hcilab.org](mailto:anneke@hcilab.org)  
[albrecht@hcilab.org](mailto:albrecht@hcilab.org)



# Integrating Sensor Data in Flash MX 2004

Anneke Winter  
anneke@hcilab.org

21st April 2005

## **Abstract**

For computer gamers there were about three major input devices available in the past few years - mouse, keypad and joystick. The problem of new and innovative input devices so far is their integration into common gaming environments. There is no interface for the average game developer to integrate these devices as easy as integrating mouse or keyboard for interaction. To make it easier to use new and innovative input devices, I thought about creating a Flash component to make use of sensor data in a Flash application easy. I chose Flash as Flash is one of the mayor programs to create small games very quickly. This component reads the sensor data out of an XML file which is published by a server that delivers the sensor data. The component therefor simply needs to be integrated per drag and drop into the Flash application an then configured with 2 parameters, the link to the server that delivers the sensor data and a link to a configuration file that tells the application in advance which variables are going be sent over the server to make parsing quicker. The developer then can access the variables on the root time-line like any other local variables before. So accessing and using these parameters is easy for the Flash user.

# Contents

<b>Titlepage</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
<b>2 Related Work</b>	<b>4</b>
2.1 XML . . . . .	4
2.2 SOAP . . . . .	4
2.3 Widgets / Phidgets . . . . .	5
2.4 Smart-Its . . . . .	5
<b>3 Integration of External Data in Flash</b>	<b>6</b>
3.1 Text Files . . . . .	6
3.2 XML . . . . .	6
3.3 Web Services . . . . .	7
<b>4 Component implementation</b>	<b>9</b>
4.1 Idea . . . . .	9
4.2 Software Architecture . . . . .	10
4.3 Implementation and Interfaces . . . . .	11
4.4 Hardware Architecture . . . . .	14
4.5 Test Application for the implemented component . . . . .	18
<b>5 Room Occupancy System</b>	<b>19</b>
5.1 Basic concept and architecture . . . . .	19
5.2 Physical interaction hardware . . . . .	19
5.3 Microcontroller implementation . . . . .	20
5.4 Flash Implementation . . . . .	21
<b>6 Virrig Car Race Game</b>	<b>23</b>
6.1 Basic concept and architecture . . . . .	23
6.2 Physical interaction hardware . . . . .	23
6.3 Microcontroller implementation . . . . .	24
6.4 Flash Implementation . . . . .	24
<b>7 Results</b>	<b>26</b>
<b>Bibliography</b>	<b>27</b>
<b>List of Figures</b>	<b>28</b>

# 1 Introduction

Right now for Flash developers there are 2 mayor input devices designated by the producers of Flash, which are the keypad and the mouse. For these it is easy to ask for properties like movement, position or which key is pressed. For new input devices it is hard to get any integration into the Flash environment, which supposes that there are no games taking advantage of new and innovative user input devices differing from the two mentioned above. This is a pity, since Flash offers a great platform to quickly and easily develop games or edutainment applications - a mixture of education and entertainment applications. Unlike Java or C++, Flash is easy to learn which holds true also for non programmers. So I thought about creating a component for Flash to make it easy to use new devices for a Flash application. The requirements were the simplicity of use for the Flash developer and the usage of a standard format for providing the data. As a standard I chose XML because it is a nice and easy way to describe data, and is easily understood even by laymen. For the component I invented an object that can be included in any Flash application by drag and drop and only has to be configured with two parameters. The variables are, after configuring, available on the main time-line called 'root'-time-line. To set everything up, the developer just needs to know the location of the XML file providing the data and the configuration of an XML file that configures the variables for Flash on the first loading called the config file, which are both described later on more precisely.

In this work I describe a project where I invented a specific design for integrating sensor data into Flash MX 2004 trying to keep this mechanism as simple as possible for the fictive end user, a not as practiced Flash programmer. In the first section I am showing different techniques and standards that were already invented, and what or why I used them or not, in the following section there are described several mechanisms for data integration already implemented for Flash. After this I take a look at the implementation of the component and show its work on three sample applications. At the end I show up the results based on the discussion about the practical use of the component.

## 2 Related Work

### 2.1 XML

XML has been approved as a standard by the World Wide Web Consortium (W3C) for creating special-purpose Markup Languages, and been adopted by many companies so far, like Microsoft, Netscape and now also Macromedia. XML is a markup language derived from the Standard Generalized Markup Language (SGML). XML is used to describe data. The goals of XML therefor is different from HTML as in XML the focus is on describe the information contained in the data, and HTML much more focusses on describing how the data described in the document is actually going to look like. Also in XML there are no predefined tags like in HTML the `< body >< /body >` tag for example. The user can define its own tags to describe its own data, witch makes him freer. To make sure the Documents still all have the same structure the user can define the wanted structure in a Document Type Definition (DTD) or an XML schema. By this DTD or schema a programm using the XML file can validate it, which means to check weather it has the right structure or not. XML files as well have to be well-formed, which means all tags used have to be closed and every new tag opened has to be closed before an outer tag is closed. As well there is only one root element allowed and all attributes are places within single (') or double(") quotes.

### 2.2 SOAP

SOAP, formally known as "Simple Object Access Protocol" is an light weight protocol to send information between applications through Web Services. SOAP is an extensible and decentralized framework that can work over multiple computer network protocol stacks. SOAP is a protocol that enables applications to use web services. It can be run on top of all the Internet protocols, but HTTP is the only one standardized by the W3C and therefor is the most common on the Internet nowadays. SOAP is based on XML, which means it uses XML syntax to transfer information between the applications. [9]

In the Flash environment Macromedia also adopted it and it is an envelope that encapsules the messaging to and from the Flash client application to the Web Service. The data that is sent by SOAP is just raw textual data in an XML format, while other protocols use a binary format to sent information. The disadvantage here is that the SOAP protocol needs more bandwidth than the binary ones, because they can compress their data better. But the advantage is that SOAP is a standard, it is not proprietary like the Macromedia Flash Remoting protocol AMF and is widely spread across the Internet. [3]

## 2.3 Widgets / Phidgets

A widget is a graphical component, or control, that the user interacts with, such as a window or a text box. Widgets are sometimes qualified as virtual to distinguish them from their physical counterparts, e.g. virtual buttons that can be clicked with a mouse cursor, vs physical buttons that can be pressed with a finger. Widgets are often packaged together in widget toolkits. Programmers use widgets to build graphical user interfaces (GUIs). [9] Thus a Phidgets is a physical widget, they package and abstract input and output devices, which means they hide the details about implementation and construction, but expose the functionality of the device. In contrast to widgets Phidgets are a connection manager to track how devices appear on-line, a simulation mode to allow programmers to develop, test and debug the physical interface even without having the physical device available. The idea behind Phidgets is to enable everyday programmers to develop physical interfaces quicker than before just by using Phidgets.[5]

## 2.4 Smart-Its

Smart-Its are small, embedded context aware devices that integrate sensing, actuation, processing and communication. They are small embedded devices that can be attached to everyday objects to set them up with sensors and actuators to let these common objects communicate with its surrounding objects or the user. They are uniquely developed for each application and can be programmed for any kind of purpose. The hardware design allows a lot of possible setups of sensors and actuators. Therefore Smart-Its are very flexible in use. There are even wireless radio transmitter one can equip its Smart-Its with so integration becomes more and more simple and scalable. [1]

## 3 Integration of External Data in Flash

There are already several mechanisms for integration of external data into Flash MX 2004, but most of them, like the XML component require a lot of knowhow and programming skills from the developer to use them. Let us have a look at the different ways to integrate external Data into Flash that were available.

### 3.1 Text Files

One Method for Flash developers is using text files for getting information or variables for the Flash application. Therefor in Flash there are built in methods and Objects for reading text files. These Objects allow the developer to request external text files, load them into the Flash movie, see whether they are already loaded or not, and read the variables declared in the text file and then having them available inside the loading object. The object used for this is called the LoadVars object. It offers all these methods for getting information about the state of the data loaded and methods to load or send data, like `load(url:String)` for loading the text file, `getBytesTotal()` to get the amount of bytes Flash has to load for the whole text file, `getBytesLoaded()` to get the amount of bytes Flash has actually loaded, `send()` to send all the variables known by the current instance of the LoadVars object or `sendAndLoad()` for simultaneous send and load of variables out of a text file. The idea of loading text files is to create text files with variables inside so that the developer does not have to parse the variables himself, but having Flash doing this for him. The text file therefor has to have the following format:

```
(variableName1=variableValue1)[& variableNameX=variableValueX]*
```

By loading this variables string into Flash, Flash automatically detects the variables and stores them into the loadVars instance, where the developer can use them. This is a good way to easily get variables into Flash, but the developer must know how to use the LoadVars object and therefor it is not that easy to use for not programming experienced developers.

### 3.2 XML

XML can also be used like the data files we saw before for reading and writing data. Flash adopted the XML technology since XML becomes more and more important as a format to send and receive structured data. Flash therefor only needs the XML file, and ignores any DTD or Schema written to describe the structure. Flash does not even care if the file is well formed or not or whether it has two root elements or one, you only might get problems parsing it later on. For loading an XML file in Flash the developer has an XML object very similar to

the LoadVars object described in the subsection before. The XML object delivers similar methods like `load()`, `send()`, `sendAndLoad()`, `getBytesLoaded()` and `getBytesTotal()` as well, but beyond these it also offers a variety of methods and attributes to browse through the loaded XML file or manipulate the loaded data. The syntax here is much like XPath. Given attributes like `ChildNodes`, `FirstChild`, `NodeValue` etc. allow the user to easily navigate with loops through the document. This method is not as easy to use as the loading of text files, since there the document has to be parsed by the programmer. Flash does not deliver the variables on the fly into the loading object. But the advantage is that only needed variables have to be loaded, and the data provided by an XML file is structured, which means some variables stand in context with others. This structure can not be provided as easy in text files. An other strength of XML files is that its form is standardized by the W3C and well spread among the Internet. XML is a platform independent format which simplifies data exchange between different domains.

### 3.3 Web Services

A new and one of the most talked about and hyped feature of the Internet are Web Services. They are even called one of the few revolutionary ideas among all the acquisitions of the Internet within the IT industry. And its being assumed that Web Services are potentially going to change the Internet world as we know it. Web Services offer server-side objects and methods that expose application functionality across the Internet. The Web Services themselves are written in server side languages as ColdFusion, Microsoft .NET or J2EE. There are public Web Services available on the Internet to serve the most different tasks like checking traffic in a city on the other side of the world or translating text. The idea of Web Services is to offer methods that other applications can use to extend their own functionality. These applications does not need to know the language the Web Service was written in, but only its selves. So Web Services are also platform independent. Web Services send their information by means of XML packets, so they can handel nested data structures. The consumer of a Web Service only has to know the structure of the XML file, and then can use it like any other XML file as well which makes it easy to connect applications to other applications in a platform-independent way. The two standards are WSDL (Web Service Description Language) and SOAP (explained Section 2.2). WSDL is a language to describe the content of Web Services while SOAP is a protocol to send information between applications through Web Services. To use Web Services in Flash MX 2004 the programm offers a Web Services panel (see Figure 1), that takes the URL of any WSDL Web Service location and translates it into plain English. This means the panel shows the developer the method names offered by the Web Service and the parameter these methods require and what the return value of the given methods.

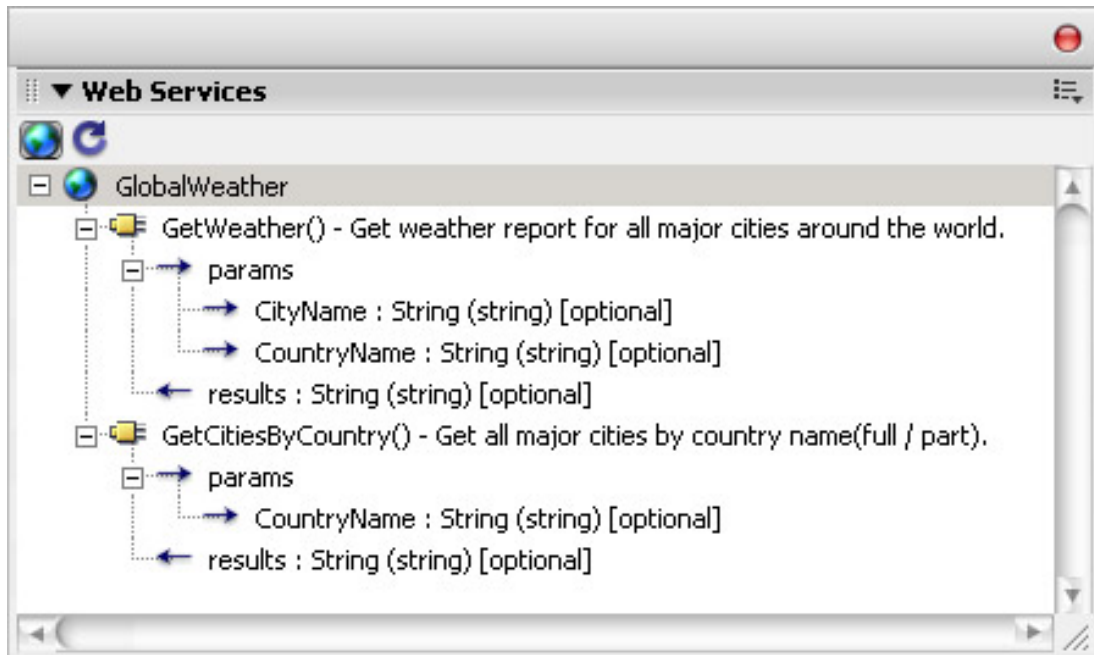


Figure 1: Window to visualize a Web Service in Flash MX 2004.

SOAP sends the enquired files as plain text, and not in an compressed binary format which unfortunately makes the application consume more bandwidth. To avoid this Macromedia created its own protocol named AMF, short for "Action-Script Messaging Format". The AMF is binary and so needs less bandwidth than SOAP. But on the other hand SOAP is spread more widely over the Internet. The developer therefore needs to deliberate about which service is the best for his application. The big advantage of using Web Services in Flash MX 2004 is that Flash automatically processes the given XML files from the Web Service URL converting it into native Flash data structures. This spares the developer from having to write complex loops to parse the XML file as described in Section 3.2. This mechanism makes the parsing quicker since it is a native parse algorithm implemented into the Flash development environment. [3]

## 4 Component implementation

### 4.1 Idea

The first question was which platform to develop for. I chose Flash MX because it is commonly used by developers and designers and offers a nice development environment for programmers and even non-programmers. 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. I therefore created an ActionScript 2.0 component that can be integrated in any Flash application by simple drag and drop. The idea was to create a component that makes it easy for not that experienced Flash programmers to integrate new input devices or sensor data into their applications. The shown methods in Section 3 all need a lot of know how in programming and the use of Action Script 2.0 and thus are not really useful for beginners. Hence the component requirements were simplicity in use and scalability so that different kind of sensor devices could be used in Flash by this one component. The idea was having a component with as few parameters for configuration as possible, but as many as needed to fit for a big scale of applications. Components are small executable blocks to reuse in every application. Therefore the Flash development environment offers a panel showing all components currently available in the Flash component directory of the user.

After dragging the component onto the Flash "stage" the user 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 details. Afterwards the developer has access to all variables made available from the component on the main time-line called "root-time-line" in Flash MX.

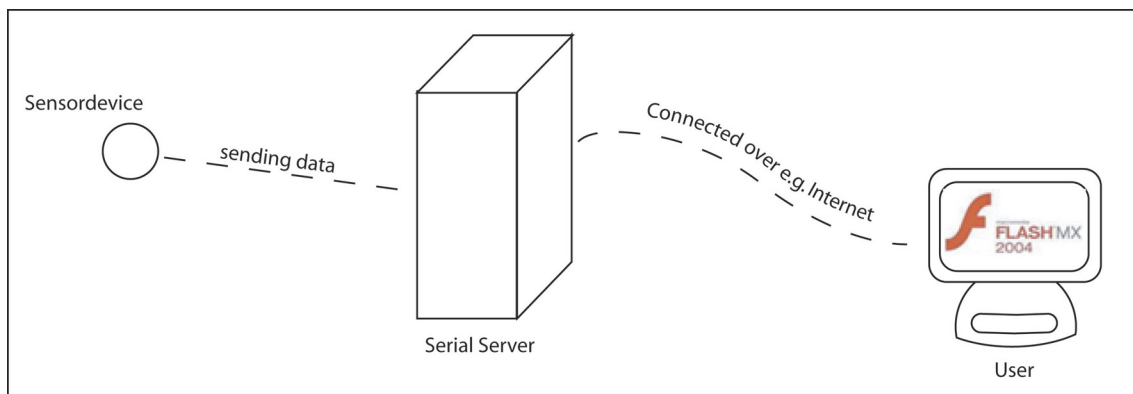


Figure 2: Setup of the Component and Application. The sensor device is connected to a server to send the current sensor data. The server then converts it into readable XML for the component, and makes it available for the Flash application.

From there on he or she can use them like every other global variable within

the application. All kinds of input devices like a head tracking sensor, a glove, a new game pad, or any other device the user invents 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 (described in the following subsection). 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.

## 4.2 Software Architecture

For input I chose XML files since the mechanism described in Section 3.2 is the most adaptable one for the requirements, allows creating nested data, and XML is different from the text files mechanism a W3C web standard. The idea was letting the component read a configuration file at the beginning to initialize the expected variables. Then the component only gets the changed variables in a different XML file. The work flow of the component is shown in the activity diagram in Figure 3. The diagram visualizes how the component works: After reading the configuration

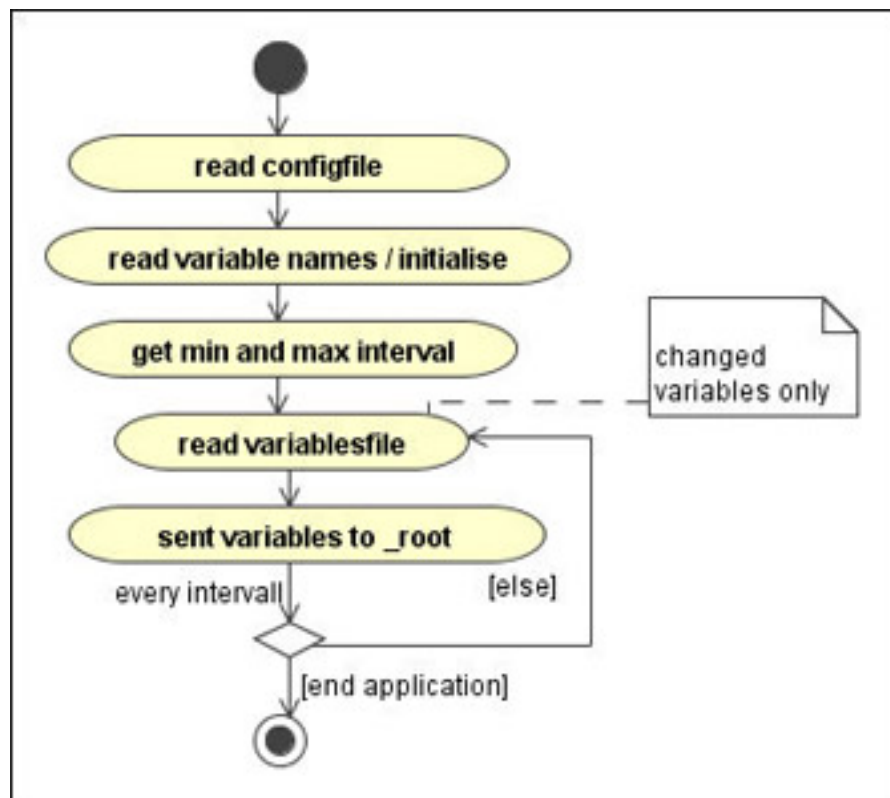


Figure 3: Activity Diagram showing the workflow of the new created component.

file the component initializes the variables on the main time line called "root"-time-line. Then it sets the interval for rereading the variables file. now it starts reading the variables file and setting the variables to their current values. This loop of reloading the changed variables it does as long as the application runs or the component object lives in the application.

### 4.3 Implementation and Interfaces

Now an introduction into the implementation of the component. I created a class which I called XMLVariables. This class had to inherit from the class mx.core.UIComponent to offer us all the component specific methods. Since the UIComponent class inherits from the main class MovieClip I can now use all the Flash specific methods like `onLoad()`, `onEnterFrame()` which I needed to set up the component and program the reload.

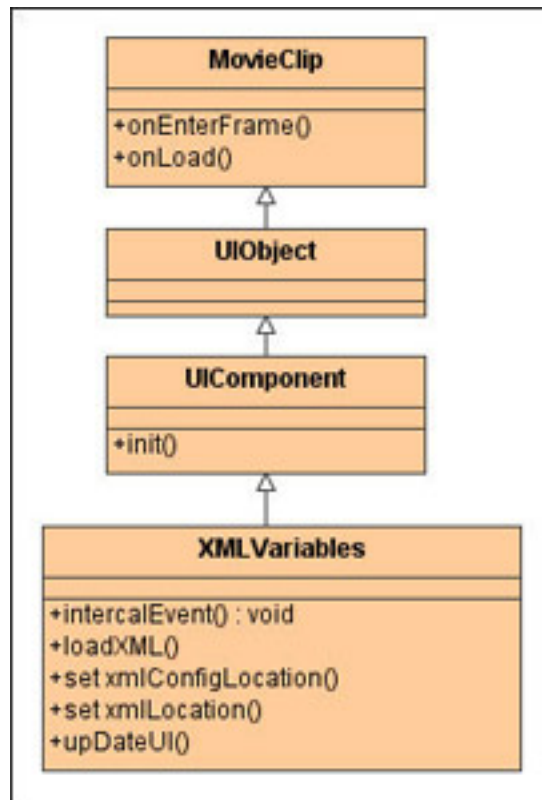


Figure 4: Class Diagram of Flash MovieClip Inheritance. Here you see the Flash Object structure, and how even a Component Object inherits from MovieClip.

As seen in Figure 4 the component inherits at the end from MovieClip. This is the main Object in Flash. A MovieClip object has certain public methods the component can now make use of. For us the most important was inheriting the `onLoad()` to know when I have to start loading the configuration file.

To define the parameters for the component I want the user to just enter the parameters within the parameter window of the Flash environment and not having him to write code I need a special syntax using ActionScript, two of our attributes `xmlConfigLocation` and `xmlLocation` are equipped with getter and setter methods. Here I used the implicit getters and setters of Flash and a special syntax that makes the attributes appear as parameter in the parameter panel:

```
[Inspectable( type="String", defaultValue="config.xml")]
public function set xmlConfigLocation (xmlConfigFile:String){
    this.xmlConfigFile = xmlConfigFile;
    invalidate();
}
public function get xmlConfigLocation():String {
    return xmlConfigFile;
}

[Inspectable( type="String", defaultValue="variables.xml")]
public function set xmlLocation( xmlFile:String ):Void {
    this.xmlFile = xmlFile;
    invalidate();
}
public function get xmlLocation():String {
    return xmlFile;
}
```

The `Inspectable` keyword makes the attributes editable in the component Inspector or parameter panel later on for the non programming user of our component (see Figure 5).



Figure 5: Flash MX 2004 Component Parameter Panel. The user only has to enter 2 values to configure the component.

To use the component the user has to drag it out of the component panel onto the so called stage, the main working area. There he can place it anywhere. Now he has to configure the component with the two parameters.

The parameters are the URI to the XML files. They can be on the same server as the application but also on a different one where you need a file called `crossdomain.xml` that makes the URI accessible for the Flash application. Without that file a Flash application can not access any other foreign server. This is a security mechanism from

Flash MX 2004 Flash Users have to deal with. The crossdomain.xml has to be of the following format:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd" >
<cross-domain-policy>
  <allow-access-from domain="*"></allow-access-from>
  <allow-access-from domain="http://www.your_Flash_app_uri.com"></allow-access-from>
</cross-domain-policy>
```

The \* allows all applications to access the server having the crossdomain.xml on its root folder. This is not quite smart since you could now have unwanted users using your information in its application. Therefore it is better practice to use the second version and just allow several specific URIs the access.

The two XML files the application gets its information from have to be of the following shape.

First comes the configuration file:

```
<?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>
```

<b>Tag</b>	<b>Tag Explanation</b>
<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

I chose making a configuration file where the user declares all the variables the application has to expect in advance to make the periodical reading and parsing of the variables quicker, since now only the changed variables have to be sent and red, and no variables

have to be initialized during this periodical reading. All the variables provided by the sensor device are declared and initialized in advance, later while reading the variables file only their values are set new.

The variables file has to have the following structure:

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

Here the user only sends the changed variables to make parsing quicker but he can also sent all variables every reload, which is going to thwart his application, but for some purposes and applications speed probably does not matter.

## 4.4 Hardware Architecture

I have implemented several sensing devices which share the same basic hardware architecture. Basically they all work the same way. I am going to explain their functionality using the Virrig cushion, shown in Figure 6 . Here 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 all our implementations I used the Smart-Its platform described earlier in this paper 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.[1]

In newer implementations I used the Particles, which are similar devices to the Smart-Its. Particles are developed at TecO [8]. 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.

Independently from that is the actual object that is used as input device and into which I embed the sensors. I deliberately searched for objects that are known to most people but are not yet used as input devices. In this section, one out of the many possibilities I found is presented. I used an IKEA balance cushion named Virrig shown in Figure 6 [6] [4]. 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 6: The Virrig input device shown from the side. It is very flexible in use as the user can sit, stand, kneel or lie on it and it is very robust.

The digital device I attached inside the hemisphere does not change the affordance or the physicality of the cushion [2]. The user still can sit or stand on it as before, and since I use radio technology for data transmission there are no cables leaving it, so it can still be tilted and rotated like before. I show how to use the cushion in an edutainment application in Section 6.

Inside the cushion, attached to a wooden plate, there is a Smart-Its (see Figure 7) with the following components:

- four large batteries are used as power supply; this ensures that the device needs not be opened even for long term user studies
- four 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 8.

Now I describe some 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

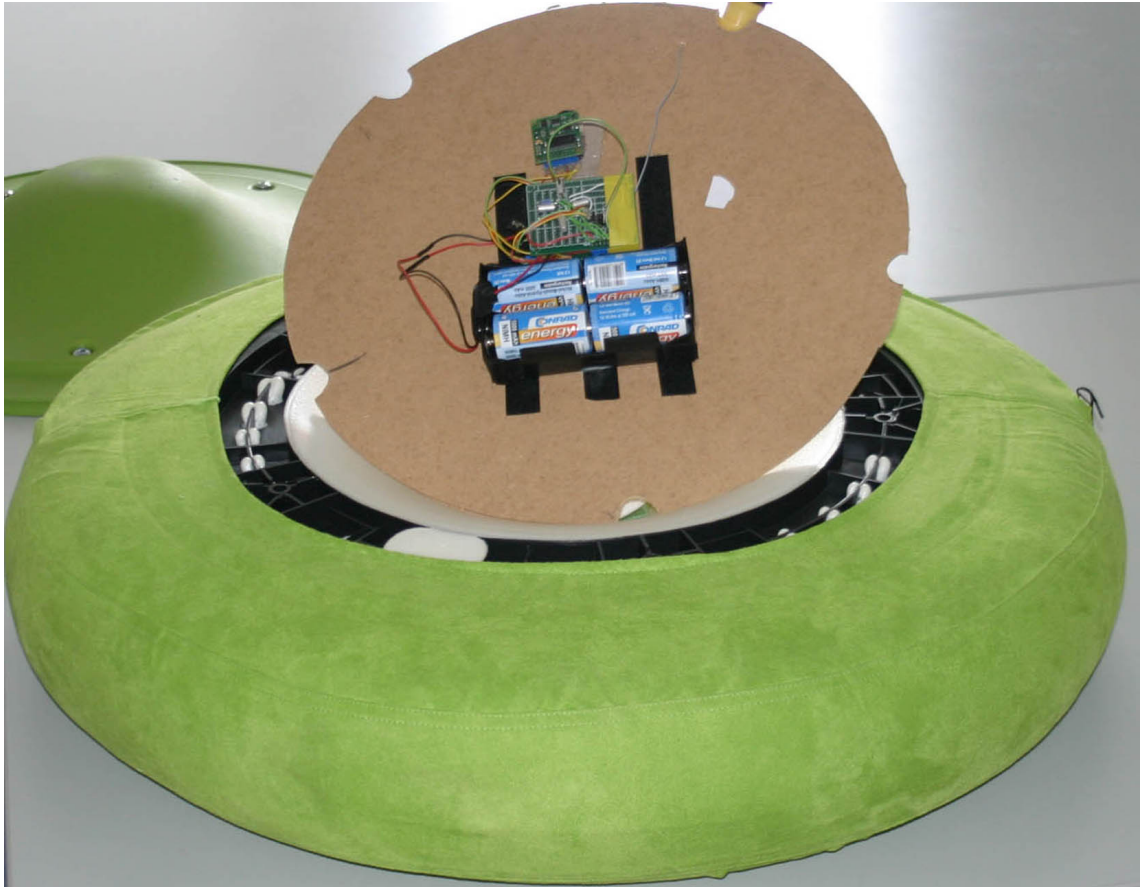


Figure 7: Opened Virrig with the integrated Smart-It.

cushion and forwards it to a program called Serial Server ([7]) 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 ?? . As has already been stated, I 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.

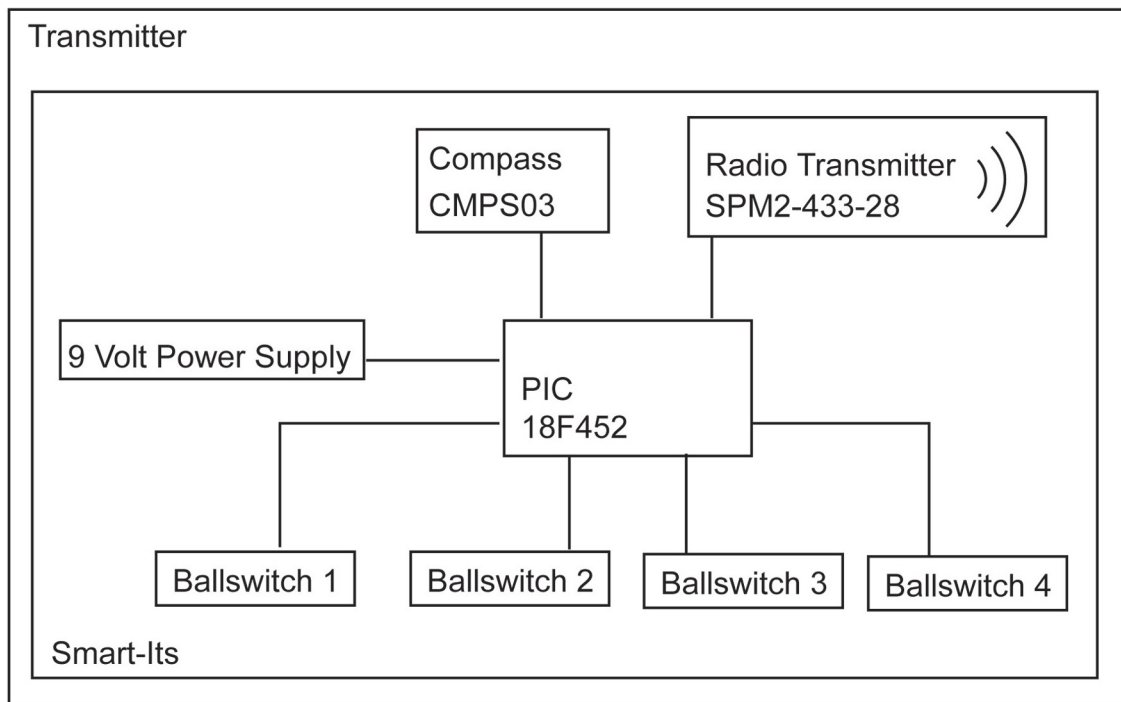


Figure 8: Hardware architecture of the input device: Four ball switches and a CMPS03 I2C compass module are connected to the PIC 18F452 microprocessor which is powered by a 9V block battery. Data is sent by the Radiometrix SPM2-433.28 rf module.

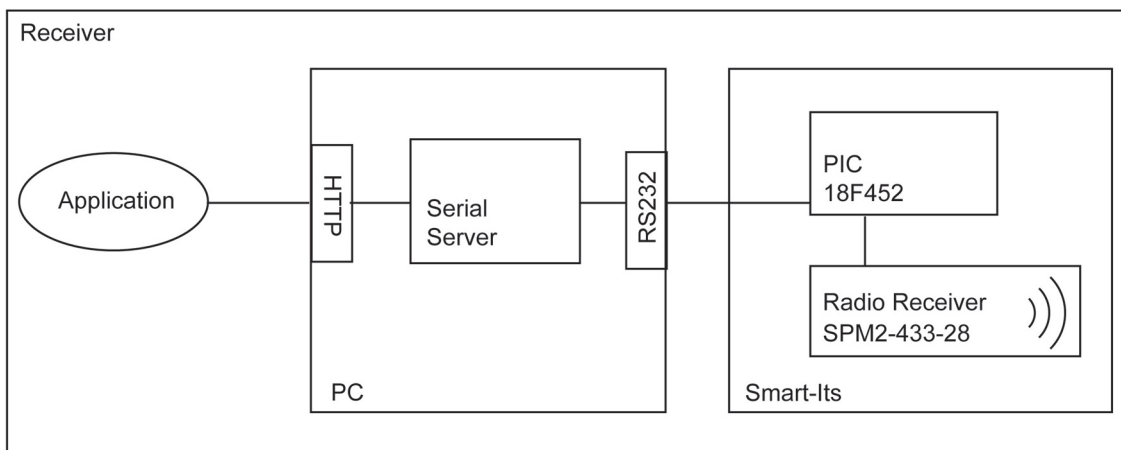


Figure 9: Hardware architecture of the receiver: another SmartIt is connected to the PC. PC and SmartIt communicate over RS232. The PC then uses the Serial Server to convert the received data into the wanted xml format and then makes it available over HTTP for the Flash Application.

## 4.5 Test Application for the implemented component

Now I invented a test application to prove the correct work of the Flash component. Therefore I set up the system as followed:

I connected the application computer over the internet to the Serial Server. This server provided the needed XML information, the changed variables file, and the configuration file for us on a certain URI and port. This address I had to enter as a property for our component in the Flash environment. The server got his information about the changing variables from a sensor device which sent its data to the server wireless by radio technology. As a sensor device I chose an IKEA cushion called Virrig, you have already read about the functionality of the cushion in the previous subsection. In the application I only visualized the rotation and tilt of the cushion. A screen shot of the application can be seen in Figure 10

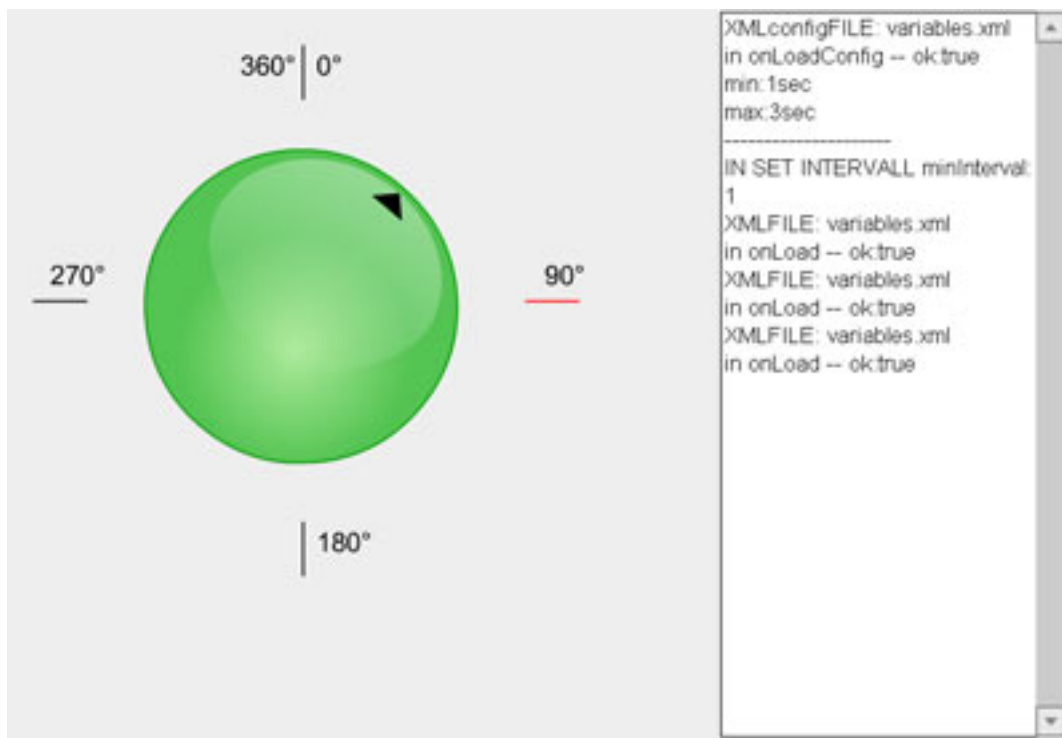


Figure 10: Screen Shot of the Test Application. Here the Virrig cushion is visualized showing its current tilt and rotation.

On the left side one sees the cushion from the bird's eye view. Here he can see the rotation of the Virrig cushion shown by the black arrow on top of the green circle and the lines around it indicate if marked red the tilt of the cushion. On the right side there is a there is a debug output panel to proof the correct work.

## 5 Room Occupancy System

### 5.1 Basic concept and architecture

The second application developed using the component was a Room Occupancy System. The input device here was a little wooden box which shows all states of a room e.g. occupied, empty etc. A magnetic pin is placed on certain spots of a ferromagnetic plate to indicate whether the current state of the room. The boards is equipped with magnetic switches that recognize where the magnet is placed the Smart-Its inside communicates with the server which provides the variables file for the Flash application. This is the way the application gets to know where the magnet is currently placed at. In Figure 11 one can see a screen shot of the screen shot.

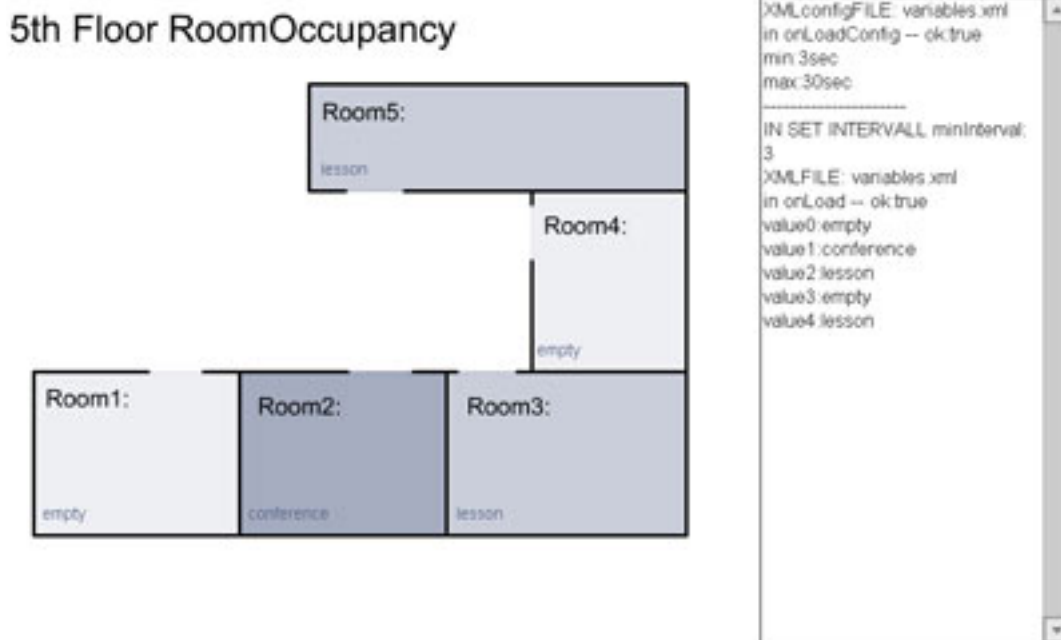


Figure 11: Screen Shot of the Room Occupancy Visualization.

### 5.2 Physical interaction hardware

As an input device, I 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. I 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.



Figure 12: Room Occupancy System: A simple non-digital information door system at a door at University of Technology, Munich.

### 5.3 Microcontroller implementation

Using a little wooden box we arranged a SmartIt into it this can be seen in Figure 13. The SmartIt of the Room Occupancy System is equipped with 4 magnetic reed switches, which is the only difference to the Virrig hardware setup, this is shown in Figure 14 . The micro controller is programmed the same style as the Virrig controller. It sends the data in specific packages to the Serial Server using radio technology.

#R:0:0:0:0:1:V§

The String has a starting symbol, and an ending symbol so that the Serial Server can proof if a whole variables string is sent to the sever or the connection is somehow disturbed and an incorrect string has arrived at the Server. The Variables then are splitted at the colon and transformed into XML for the Flash Application. The values are set to the value 1 if the magnet triggers the button inside showing the current selection. The rest of the variables stay zero, meaning their state is not set. The server now knows which state is which occupancy, sending back the XML file for the Flash application with the right states for each room.

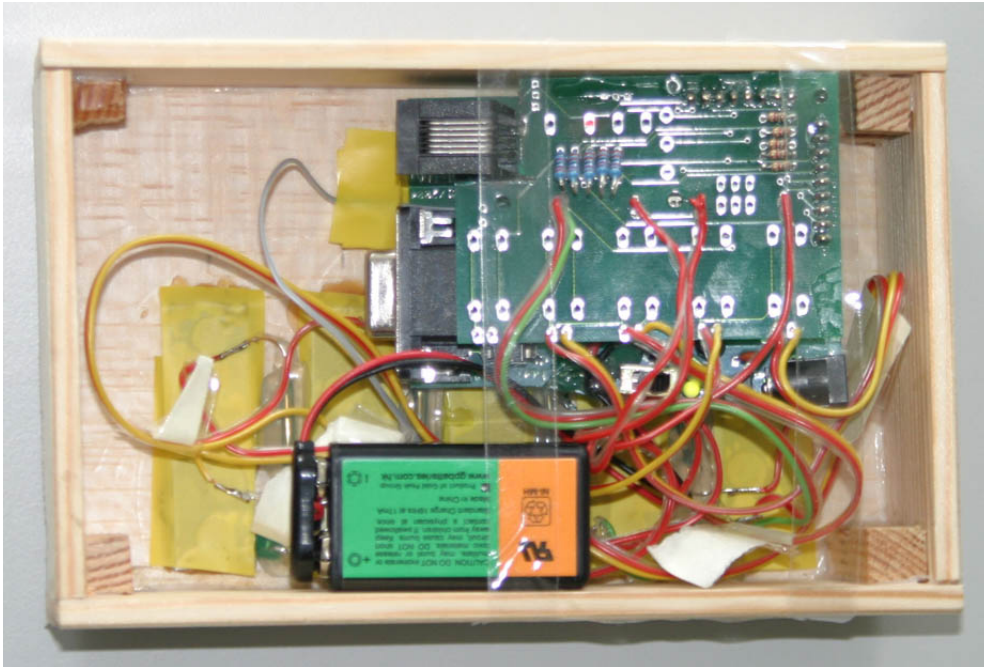


Figure 13: The Wooden prototype of the Room Occupancy System. It shows the SmartIt inside the wooden box, and the setup depicted in Figure 14.

## 5.4 Flash Implementation

To connect our Flash Application I dragged the new Flash Component onto our Stage, configured it with the 2 Parameters described in Section 4. Our configuration XML Files looked like this:

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE eventlist SYSTEM
"config.dtd" >
  <config>
    <intervall>
      <min unit="sec" value="3" />
      <max unit="sec" value="30" />
    </intervall>
    <vars>
      <var name ="room1" startvalue="empty" type="string" />
      <var name ="room2" startvalue="empty" type="string" />
      <var name ="room3" startvalue="empty" type="string" />
      <var name ="room4" startvalue="empty" type="string" />
      <var name ="room5" startvalue="empty" type="string" />
    </vars>
  </config>
```

So this sample application now got 5 rooms and received the information about the occupancy of the rooms from the server within the variables.xml file. There the

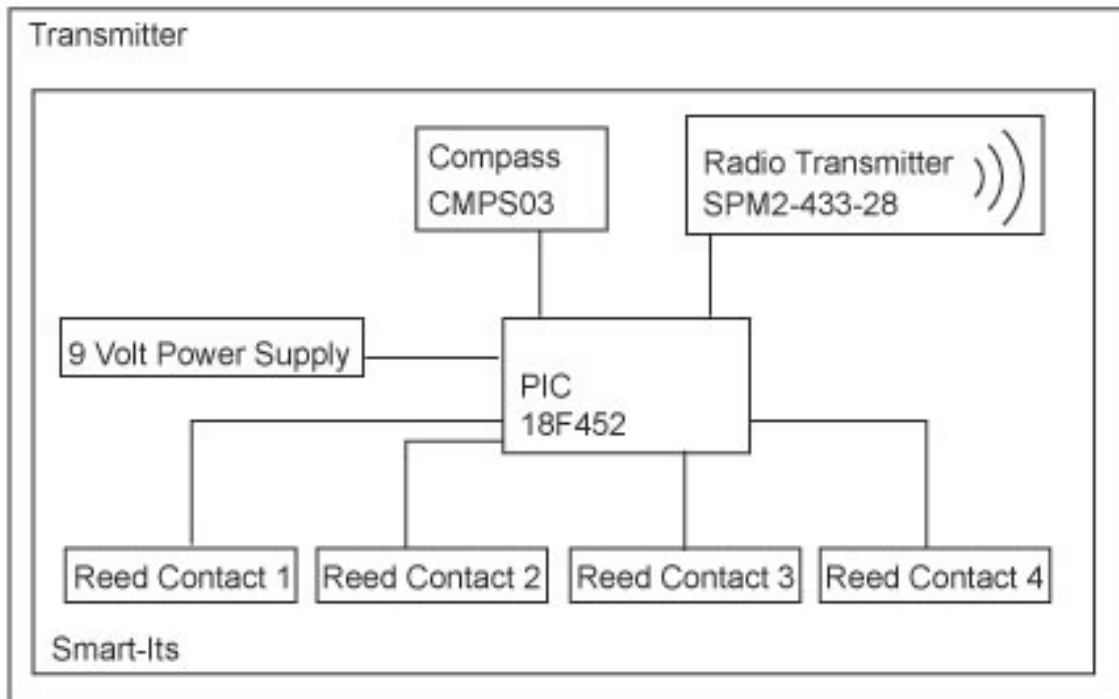


Figure 14: Hardware architecture of the transmitter: Four magnetic reed switches and a CMPS03 I2C compass module are connected to the PIC 18F452 micro-processor which is powered by a 9V block battery. Data is send by the Radiometrix SPM2-433.28 rf module.

attribute value is set to the current state of the room. The variables.xml then changes the display of the Flash application by there setting the color of a room depending on the state and changing the caption of the room figure. On the right side the text Box is only for debugging and checking the correct work of the component.

## 6 Virrig Car Race Game

### 6.1 Basic concept and architecture

This application is the biggest application to proof the functionality of the invented component. Here I used again the Virrig cushion already introduced in Section 4. I implemented an edutainment application - a car race game combined with a multiple choice question and answer game. The start screen of the game is shown in Figure 15.



Figure 15: Screen Shot - Start Screen of the Game. Here the user can choose between the keypad or Virrig as his input device.

Here the user can drive and control the car using the balance cushion or using the keypad depending on his choice. For the Virrig cushion to accelerate the user has to tilt forward. To break backward, and to make turns he has to tilt the way he wants to turn. On some crossroads the user drives through stop signs (see Figure 16) that make the car break and a popup window appear (see Figure 17). Here a question shows up and the user has the choice between 3 different answers. He can choose the assumed correct answer by rotating the cushion. After this he can select it by tilting backwards. Having chosen the correct answer the sign disappears, having chosen the wrong one the sign changes into semi transparent visibility until the driver has tried to answer another question then he can drive back and try to answer the previously wrong answered question again.

### 6.2 Physical interaction hardware

The hardware setup equals the setup for the test application described in Section 4. The cushion sends its data using radio technology to the serial server the server then

converts the received data string into an XML format and makes it accessible for the Flash application. For more details see Section 4.

### 6.3 Microcontroller implementation

I equipped the Virrig cushion with a Microcontroller which is all described in details in Section 4.4. I put this device into the sphere on the bottom of Virrig so that it can not be damaged easily. I programmed the controller so that it sends a String to the Serial Server that looks like the following:

```
#V:250:1:1:1:0:0:V§
```

The conversion into the right XML format is now made by the serial server, which now can proof by the beginning #V and ending character V§whether he got a whole variables string or not, and then converts it into the needed XML format explained in the following Subsection. The values provided by the String are first the rotation, then top, bottom, left, right and the last one is a pressure value used in earlier implementations. All the values are divided by a colon which makes it easy for the Serial Server to split the individual variables to transform them into XML.

### 6.4 Flash Implementation

The application using the new Component has all the variables sent by the web server available on the root time line. First the game loads an external XML file for the questions. Then it shows the welcome screen, where the user can select whether he wants to use a regular keypad to drive the car or wants to use the Virrig as his input device. Having chosen the game begins. Figure 16 shows a screen shot during the game driving the car.

During the whole game the component reloads the URI on the webserver where the Virrig data is provided. After loading the variables are available on the root time line, where the car object looks for its information to drive the car e.g. rotate or move the background giving the impression the car is moving. When the driver hits one of the stop sign on the streets the question window appears (See Figure 17). Here a new sensor data plays a role - the rotation. the compass inside the cushion now indicates a certain rotation which the program uses to find out which question the user selects, meaning the user can influence his selection by rotating his seat. For the initialization of the variables our XML configuration file looked like this:

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE eventlist SYSTEM  
"config.dtd" > <config>  
  <intervall>  
    <min unit= "sec" value="1" />  
    <max unit= "sec" value="3" />  
  </intervall>  
  <vars>  
    <var name = "rotation" startvalue="0" type="integer" />  
    <var name = "left" startvalue="0" type="boolean" />
```

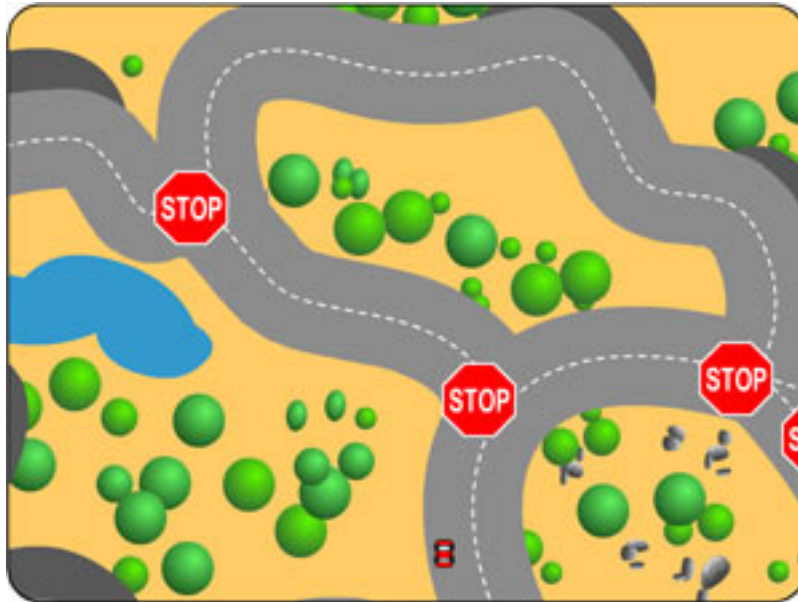


Figure 16: Screen Shot of the Race games driving scene. Here you see the driving scene, and the stop signs on the road for the multiple choice questions.

```

    <var name ="right" startvalue="0" type="boolean" />
    <var name ="up" startvalue="0" type="boolean" />
    <var name ="down" startvalue="0" type="boolean" />
  </vars>
</config>

```

The five variables are initialized at the beginning meaning the loading of the game. Afterwards only the changed variables are submitted by the server trying to keep the performance optimized. A variables.xml file submitted could look like this (but it does not necessarily have to contain all the variables):

```

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE eventlist SYSTEM
"variables.dtd" > <changedVars>
  <var name ="rotation" value="50" />
  <var name ="left" value="0" />
  <var name ="right" value="0" />
  <var name ="up" value="1" />
  <var name ="down" value="0" />
</changedVars>

```

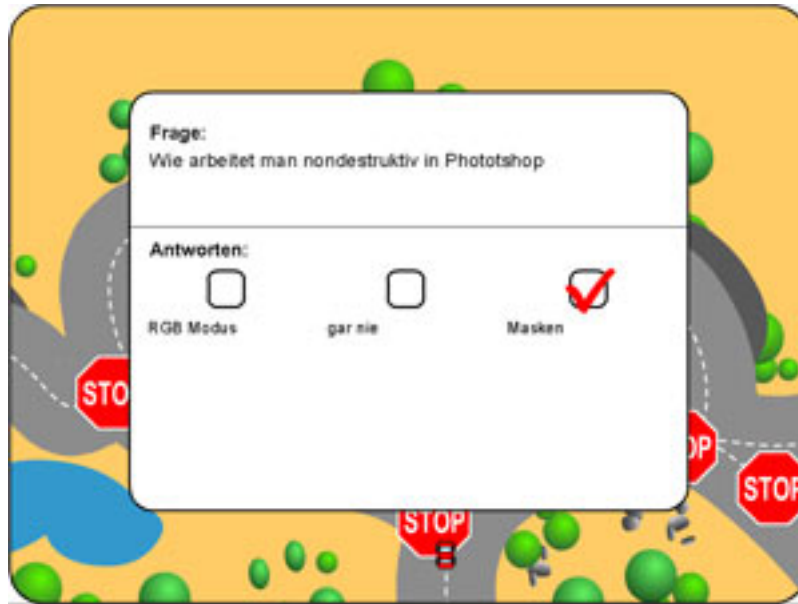


Figure 17: Screen Shot of the Race Games Question Window. The player has to rotate the Virrig cushion to select the wanted answer.

## 7 Results

This work presented a novel approach for integrating unconventional and arbitrary input devices into the Flash programming environment. An open standard-based general architecture was developed for achieving the propagation of sensor values to programming environments that do not directly support the integration of novel sensor-based hardware. The feasibility of the approach was demonstrated by building several working prototypes. The developed Flash component allows even non-programmers (e.g. designers) to use the sensor data for arbitrary applications and games. The complexity of acquiring and transmitting data is completely shielded from the application programmer. The flexible naming mechanism introduced by the usage of XML allows great freedom during application development. The variables of the prototypes built are available in Flash at the root time line. The requirements of simplicity and scalability were met. The prototypical application still suffers from speed constraints but current work shows that this can be resolved using Particle hardware connected via XML sockets to Flash instead of Smart-Its hardware.

## References

- [1] 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. 2.4, 4.4
- [2] J. Gibson. The theory of affordances. In *Perceiving, Acting, and Knowing*. Lawrence Erlbaum Associates, 1977. 4.4
- [3] R. H. Jeff Tapper, James Talbot. *Object-Orientated Programming with ActionScript 2.0*. Voices that matter. New Riders, 2004. 2.2, 3.3
- [4] E. L. Löwenhielm (Designer). Ikea. <http://www.ikea.com>, 2004. 4.4
- [5] Saul Greenberg, Chester Fitchett. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. <http://grouplab.cpsc.ucalgary.ca/phidgets/gallery/phidgets-uist-2001.pdf>, 2001. 2.3
- [6] A. Schmidt, P. Holleis, and M. Kranz. Sensor virrig - a balance cushion as controller. UbiComp 2004 - Workshop 'Playing with sensors', 2004. 4.4
- [7] A. Schmidt, M. Kranz, and P. Holleis. Research Group Embedded Interaction Serial Server Project. <http://www.hcilab.org/resources/webserver.htm>. 4.4
- [8] Telecooperation Office, University Karlsruhe (TH). TecO Smart-Its Particle Project. <http://particle.teco.edu/>. 4.4
- [9] m. Wikipedia. Wikipedia. <http://en.wikipedia.org/wiki>, February 2004. 2.2, 2.3

## List of Figures

1	Window to visualize a Web Service in Flash MX 2004. . . . .	8
2	Setup of the Component and Application. The sensor device is conected to a server to sent the current sensor data. The server then converts it into readable XML for the component, and makes it available for the Flash application. . . . .	9
3	Activity Diagram showing the workflow of the new created component. .	10
4	Class Diagram of Flash MovieClip Inheritance. Here you see the Flash Object structure, and how even a Component Object inherits from MovieClip. . . . .	11
5	Flash MX 2004 Component Parameter Panel. The user only has to enter 2 values to configure the component. . . . .	12
6	The Virrig input device shown from the side. It is very flexible in use as the user can sit, stand, kneel or lie on it and it is very robust. . . . .	15
7	Opened Virrig with the integrated Smart-It. . . . .	16
8	Hardware architecture of the input device: Four ball switches and a CMPS03 I2C compass module are connected to the PIC 18F452 microprocessor which is powered by a 9V block battery. Data is send by the Radiometrix SPM2-433.28 rf module. . . . .	17
9	Hardware architecture of the receiver: another SmartIt is connected to the PC. PC and SmartIt communicate over RS232. The PC then uses the Serial Server to convert the received data into the wanted xml format and then makes it available over HTTP for the Flash Application. . . .	17
10	Screen Shot of the Test Application. Here the Virrig cushion is visualized showing its current tilt and rotation. . . . .	18
11	Screen Shot of the Room Occupancy Visualization. . . . .	19
12	Room Occupancy System: A simple non-digital information door system at a door at University of Technology, Munich. . . . .	20
13	The Wooden prototype of the Room Occupancy System. It shows the SmartIt inside the wooden box, and the setup depicted in Figure 14. . .	21
14	Hardware architecture of the transmitter: Four magnetic reed switches and a CMPS03 I2C compass module are connected to the PIC 18F452 microprocessor which is powered by a 9V block battery. Data is send by the Radiometrix SPM2-433.28 rf module. . . . .	22
15	Screen Shot - Start Screen of the Game. Here the user can choose between the keypad or Virrig as his input device. . . . .	23
16	Screen Shot of the Race games driving scene. Here you see the driving scene, and the stop signs on the road for the multiple choice questions. .	25
17	Screen Shot of the Race Games Question Window. The player has to rotate the Virrig cushion to select the wanted answer. . . . .	26