Towards a framework for the rapid prototyping of physical interaction

Andrea Bellucci, Alessio Malizia and Ignacio Aedo
Universidad Carlos III de Madrid
Avenida de la Universidad 30, 28911, Leganés, Madrid, Spain
abellucc@inf.uc3m.es, amalizia@inf.uc3m.es and *aedo@ia.uc3m.es*

Physical Interaction is based on sensors and emitters that convert real world data into digital data and viceversa. Accessing to these data in a meaningful manner can be a hard process that requires knowledge of the underneath physics and many hours of programming. Furthermore, data integration can be cumbersome, because any device vendor uses different programming interfaces and communication protocols. We introduce preliminary work for the design and implementation of a framework that abstracts low-level details of individual devices. We aim at providing access to sensors and emitters by means of a unified, high-level programming interface that can be used for the rapid prototyping of interactions that explore the boundaries between the physical and the digital world.

Physical interaction, ubiquitous interaction, programming toolkit.

1. INTRODUCTION

Human Computer Interaction is a multidisciplinary research area that embraces knowledge from computer science, psychology, sociology, cognitive science and design among others.
The profile of researchers in interaction and user experience design, who deal with new interactive technologies, found its archetype in the Renaissance man: a man with an insatiable curiosity, a great power of invention and a broad knowledge of different subject, from mathematics to architecture, engineer, anatomy and painting. Nevertheless, mastering different areas of knowledge can be difficult and time consuming and there are very few (if none) Leonardo da Vinci out there. In any case, researchers and designers who want to build prototypes of interactive systems need to have some basic knowledge of different related subjects. For example, interaction designers should have basic programming skills and know some basic electronics in order to develop prototypes for tangible and physical interaction. Programming environments such as Processing [1] and Wiring [2] are intended to facilitate the development of interactive artefacts by providing an Application Programming Interface (API) for handling visual and conceptual structures as well as the communication with physical components. However, although they provide a good level of abstraction, we noticed that they do not provide a general API to communicate with different hardware components. You can interface with a sensor and get data from it, but it will only provide raw data that you have to analyse and interpret to get some results. This is not a difficult task for a user with sufficient programming skills, but it could represent a serious obstacle for the end-user (e.g. an interaction designer or a digital artist) that simply want to use the sensor capabilities in her project. In this case, programming libraries written by expert users can be exploited to interface with hardware devices. For example, currently, there is a Processing library for interfacing with the Kinect [3] RGB and Depth (RGBD) cameras and there are also many code samples for getting data

Proceedings of BCS HCI 2012 Workshops
Physicality

from other specific sensors (e.g accelerometers, gyroscopes and compasses). Nevertheless these are only examples of isolated efforts to provide final users with libraries for managing sensors data. These attempts do not follow the rationale of a reference architecture or framework and, for this reason, they cannot be structured in a functional API.

2. MOTIVATION

A Physical Interactive system communicates with the real world by means of sensors and emitters. Sensors convert real world inputs into digital data, while emitters are mostly used to provide digital or physical feedback (e.g. a speaker emitting sounds or a blinking LED). From the experience we gathered in implementing multi-modal interaction systems [4] and [5], employing such a variety of hardware devices in a real application can be difficult because their use requires knowledge of underneath physics and many hours of programming work. For example, a digital 3-axis accelerometer is a sensor that gives you acceleration on the three dimensions. Once you get these data, you should interpret them in order to extract some meanings. It is not so straightforward to get the rotation along the y-axis (pitch) from the raw gravity data provided. Furthermore, integrating data from different devices can be cumbersome because any device vendor uses different programming interfaces and communication protocols. This is true also for the same device from different vendors. Imagine that you spent many hours programming the behaviour of the accelerometer of a Nintendo Wiimote Controller [5] and want to use the same routines in a new project with the accelerometer of an Apple Ipad [7]. That is almost impossible, because of the different interfaces and protocols used by each sensor.

These examples illustrate that there is a need in the art of toolkits and frameworks that lighten the programming of physical interactive systems and that take into account different input modalities and interaction techniques, from tangible objects to TUI-VR interactions to full-body movement input. We introduce preliminary work for the design and implementation of a framework for physical interaction in ubiquitous environment. In this paper we focus on a toolkit that abstracts low-level details of individual devices. We aim at providing access to sensors and emitters by means of a comprehensive and unified, high-level programming interface to supporting the rapid prototyping of interactive systems and the reuse of software components in different applications.

3. RELATED WORK

To help designers and HCI researches to rapidly give life to physical-digital interaction prototypes, several projects have been created, following the End-User Development (EUD) and Do-It-Yourself (DIY) philosophy. Arduino [8] is a clear example: an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software, particularly intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments. The programming language for Arduino is Wiring [2], especially designed to facilitate the creation of sophisticated physical interactive artefacts. Wiring is built on the top of Processing [1], an open source programming language and environment for people who want to create images, animations, and interactions. Today many users exploit Processing for designing, prototyping, and production.
Many frameworks and toolkits have been built in the last years, all of them trying to ease the development of interaction in ubiquitous systems.

OpenNI [9] is a software framework that provides an API for writing touchless interaction applications using RGBD cameras. Its APIs cover communication with both low-level devices, as well as high-level middleware solutions (e.g. for visual tracking using computer vision). Microsoft provides a library with the same purpose, the Kinect SDK [3], which exploits the Kinect RGBD camera and a microphones array to programming gestural and voice interaction. These approaches are limited in scope, as they support only a particular class of devices (RGBD cameras).

Other frameworks and libraries do offer support to a wide range of devices, but focus only on a particular interaction modality. Examples are Mt4j [10], libTISCH [11] and CCV [12] for multi-touch interaction or Papier-Mache [13] and reacTivision [14] for tangible interaction. Another drawback we found in the state of the art is that all of these frameworks require a quite high user's programming expertise. Squidy [15] is an exception: its objective was mainly to provide a unique library that unifies different post-WIMP frameworks and tracking toolkits. Conversely from our approach, they offer a palette of ready-to-use devices and do not provide an abstraction level of devices into general classes. Squidy's most interesting feature is the visual programming approach they use, which hides/shows on-demand the technical implementation details to the final users. Unfortunately the project seems no longer active. Another framework that employs a visual dataflows programming and integrates several devices and toolkits is OpenInterface [16]. Again, they offers pre-defined device modules and do not provide devices abstraction as we do.

The need to provide unified access in environments where heterogeneous input devices coexist has been pointed out by Taylor et al. [17]. Specifically, the found that, in Virtual Reality systems *"*different devices may have radically different interfaces, yet perform essentially the same function; some require specialized connections (PC joysticks) or have drivers only for certain operating systems*"*. Therefore they developed a software library that supports different devices by providing interfaces to a set of functions, instead of drivers for specific devices. There are other approaches that aim at providing comprehensive support to different technologies (devices and interaction techniques) in the same environment such as TUI-VR [18] for the use of tangibles in virtual reality systems and ROSS [19], which especially focus on ubiquitous interaction.

GISpL (Gestural Interface Specification Language) [20] also demonstrates research efforts towards the abstraction of input devices in the area of gestural interaction. It is a formal language that allows unifying different input modalities by the unambiguous description of gestural interfaces behaviours.

4. HAT: HARDWARE ABSTRACTION TOOLKIT

We aim at designing and developing a general framework for physical, tangible and, in general, ubiquitous interaction. To this end, we defined a set of APIs for interacting with hardware devices, which can be directly used by the final user (developer, researcher or designer) in her projects.

We view sensors and emitters as a bridge between the real world and the digital world. When a user is interacting with a computer system, she is really interacting by means of sensors, which capture data from the real world and convert these real data into digital information and emitters, which provides digital and physical feedbacks.

The Hardware Abstraction Toolkit (HAT) abstracts from the low-level details of specific devices. In this way it provides unified access to sensors and emitters, independently of their implementation or communication protocols. It defines a

general and modular hierarchy where the top-level classes are all interfaces, which allows for flexible and generic access to device features.

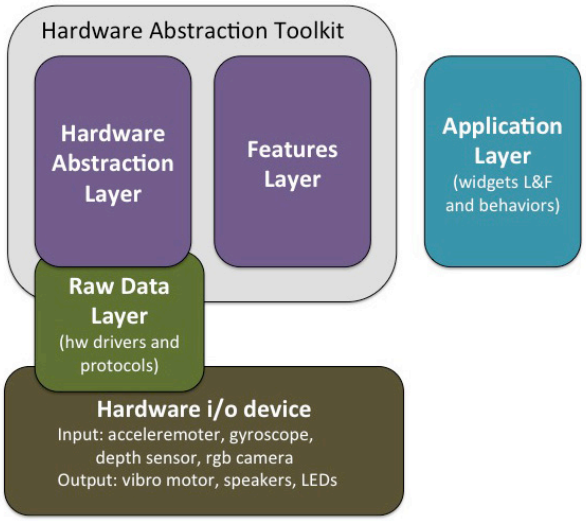

Figure 1. *The general architecture of our framework*

Within the rationale of our framework, we can broadly define three components: *Hardware*, *Abstraction* and *Application* (see **Figure 1**). In the *Hardware* level there are physical devices: sensors, emitters, physical controls and actuators. As said, via sensors we can get data from the real world and many devices can also be viewed as a composition of sensors and emitters (e.g. the Kinect is composed by an RGB camera, a depth sensing camera and an array of microphone or the Nintendo Wiimote is composed by an acceleremoter, a gyroscope, several buttons, a vibro-motor and a speaker). This idea lead us to the definition of *Entity* in our environment as a physical, tangible object that may be composed by different devices. For example, a human hand is an input device that can be considered as a passive *Entity*, because it needs an external device to be tracked. A touch display surface is another example of *Entity* that provides both input (touch surface) and output (display screen) operations. Moreover, there are virtual *Entities* that can be digitally coupled representation of physical *Entities* or independent virtual objects that can interact with other physical or virtual *Entities* The capability to conceive and define objects in this way is the main purpose of the *Abstraction* layer. The *Abstraction* component represents the core library. Here we specify the interface through which we can elaborate the raw data from a sensor and so specify an API that abstracts from the specific device implementation. For example, in the case of an accelerometer, we defined methods like `getYAcceleration(): float`, in order to retrieve the acceleration in the *y* dimension from raw data. We can also define higher-level methods like `getRoll(): double` or `getPitch(): double` in order to retrieve rotations in the *y* and *x* dimensions. The implementation of these methods is completely transparent to the user, who does not need to know how the raw data are processed to get the final value. In this way we support devices interchangeability and code reuse, because the same code for, let's say, the accelerometer of the Nintendo Wiimote will work for the accelerometer of an iPad (and any device that is compliant with the HAT specification). The abstraction toolkit is powerful enough to allow the composition of devices. For example, an accelerometer can be combined with a gyroscope to create a general Inertial Measurement Unit (IMU) component. This level also Presently, the abstraction level supports a range of device types such as accelerometers, gyroscopes, LED, display screen, touch sensors, RGB cameras and Depth sensors among others.

On top of this API, different middlewares can be developed that, for example, implement gesture detection from sensors data (the *Features Layer*, which has not yet been developed). At the *Application* level, software applications can directly exploit functionalities provided by a specific middleware.

In our framework we will also consider output channels for feedbacks, while other similar frameworks do not [15]. For example, the speakers can be used as output for giving some audio feedback to

the user. LEDs (Light Emitting Diodes) can be employed to create ambient displays giving visual feedback and small motors can provide haptic feedback (via a rumble feature). Therefore we will provide APIs also for defining and managing the output of the interactive system itself, in term of events perceived in the real world (e.g. an LED blinking) originated by some digital event (e.g. a control value exceeding a threshold) which was caused by a physical event (e.g. user's hand too close to a specific object: this event can be captured by means of a depth sensor).

4.1. Data types

Abstracting from heterogeneous devices implementations require the definition of a high-level data types that can describe raw data from hardware devices in a unified manner. To this end, we make use of Wallace's hierarchy of graphic input device semantics [21], in a similar way the Squidy [15] framework does. Nevertheless, we also needed to extend it, because Wallace's classification was not able to capture the semantics of all the devices we may encounter in ubiquitous interactive systems (see **Table 1**).

Table 1. *HAT data types*

Data type		Example of device
Value		Potentiometer, depth sensor
Location	2D	Touch surface
	3D	3D pointer
	6D	Wiimote
Choice		Button, touch sensor
String	1D	Microphone
	2D	RGB camera
	3D	RGB camera + Depth
Pick		Mouse, light pen

Value are discrete, one-dimensional data. A potentiometer sends discrete values. *Location* are data related to information of a physical space: for example the position of a contact point in a 2D surface or orientation and acceleration with respect to the three dimensions. They are represented as a n-dimensional vector. *Choice* are boolean data: a touch sensor can be a prototype of this kind of devices, for it sends 'yes' or 'no' data, depending on the contact. *String* data represents a stream of information like the one produced by microphones (one-dimensional audio data) or RGB cameras (two-dimensional video data) or RGB cameras plus Depth sensor (three-dimensional video data). The *Pick* data are a reference to an object being selected (e.g. through a 2D pointer) and it is mandatory to implement visual feedbacks of a selection. Although *Pick* data type can be implemented using *Location* data, we believe it is useful to have reference data to be logically separated from location data.

4.2. An example: the accelerometer

To better explain how our framework works, we present here a portion of its metamodel for a real sensor: an accelerometer.

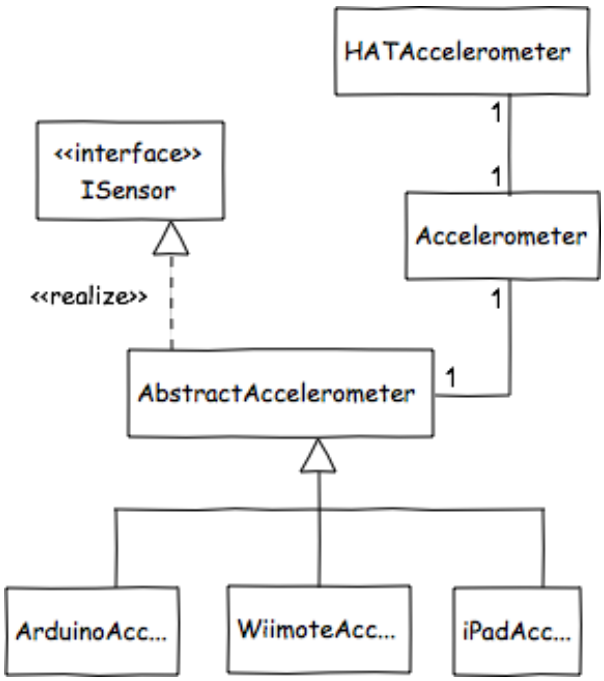

Figure 2. *Metamodel of an accelerometer*

As shown in **Figure 2**, an *AbstractAccelerometer* implements the interface *ISensor* and provides method to connect with a specific accelerometer and get raw data. The *Acceleremoter* is an instantiation of an *AbstractAccelerometer*

that make sense of the raw data (e.g. define the y-acceleration). Lastly, the *HATAccelerometer* uses 'primitive' data computed by the *Accelerometer* class in order to provide higher-level data (e.g pitch values). This information can be used to interact both with virtual and real entities. For example a system made of a microcontroller and an accelerometer can be used to rotate a virtual box (see **Figure 3**) or to tilt a physical board by means of a servo (watch the video at http://youtu.be/CsLeMpc_ykM?t=12s).

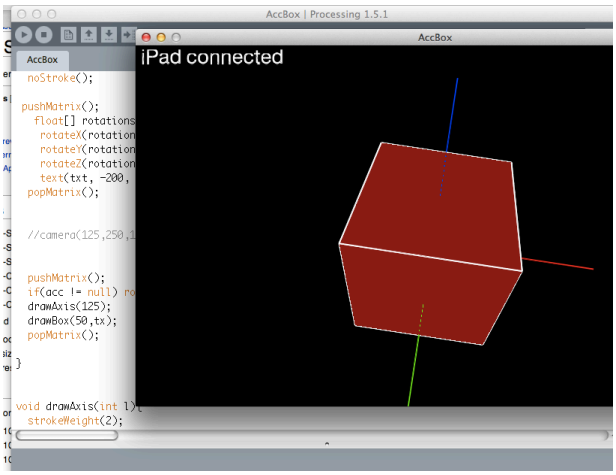

Figure 3. *A virtual Entity.*

5. CONCLUSIONS AND FUTURE WORK

We presented a first step towards a framework that eases the prototyping of physical interaction by means of abstraction of hardware devices. Preliminary studies with HCI and Computer Science master students highlighted that the APIs do reduce the programming effort (measured in terms of number of errors per lines of code and time to completion). We are now implementing APIs for a wide range of different interaction devices that can be used to define interactive objects by composition. How to achieve consistent spatial integrity among objects is still an issue. Furthermore we are designing a visual environment for our framework. It could be possible to define visual elements corresponding to desired abstract devices and functionalities. In this way end-users, with no programming skills, can quickly develop their prototypes, as also proposed by [15] and [16].

6. ACKNWOLEDGMENTS

This work has been funded by the research grant TIPEx (Spanish Ministry of Science and Innovation, TIN2010-19859-C03-01).

7. REFERENCES

[1] Processing. http://www.processing.org.
[2] Wiring. http://wiring.org.co/.
[3] Microsoft Kinect. http://www.microsoft.com/en-us/kinectforwindows/develop/overview.aspx
[4] Bellucci, A., Malizia, A., Diaz, P. and Aedo, I. (2010). Don't touch me: multi-user annotations on a map in large display environments. In *Proc. of the International Conference on Advanced Visual Interfaces (AVI'10)*, 2010, 391-392.
[5] Bellucci, A., Malizia, A. and Aedo, I. (2011). TESIS: Turn Every Surface into an Interactive Surface. In *Proc. of the International Conference on Interactive Tabletops and Surfaces (ITS11)*, 2011. ACM, New York, USA.
[6] Nintendo Wiimote. http://www.nintendo.com/wii
[7] Apple Ipad. http://www.apple.com/ipad/.
[8] Arduino. http://arduino.cc.
[9] OpenNI. http://openni.org/Documentation/.
[10] Laufs, U., Ruff, C. and Zibuschka, J. (2010). Mt4j a cross-platform multi-touch development framework. In *Proc. of the Workshop on Engineering Patterns for Multi-Touch Interfaces at Symposium on Engineering Interactive Computing System (EICS'10)*, 2010, 52-57. ACM, New York, USA.
[11] Echtler, F. and Klinker, G. (2008). A Multitouch Software Architecture. In *Proc. of the 5th Nordic Conference on*

Human-Computer Interaction (NordiCHI'08), 2008. ACM, New York, USA.

[12] Community Core Vision (CCV). http://ccv.nuigroup.com/

[13] Klemmer, S.R. and Landay, J. A. (2009). Toolkit Support for Integrating Physical and Digital Interactions. In *Human-Computer Interaction 3*, July 2009, 315-366.

[14] Kaltenbrunner, M. and Bencina, R. (2007). reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction. In *Proc. of the 1st international conference on Tangible and embedded interaction (TEI'07)*, 2007. ACM, New York, USA.

[15] König, W.A., Rädle, R. and Reiterer, H. (2009). Squidy: A Zoomable Design Environment for Natural User Interfaces. In *Proc. of the 27th International Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '09)*, 2009. ACM, New York, USA.

[16] Lawson, J-Y. L., Al-Akkad, A-A., Vanderdonckt, J. and Macq, B. (2009). An Open Source Workbench for prototyping Multimodal Interactions based on Off-the-Shelf Heterogeneous Components. In *Proc. of the 1st Symposium on Engineering Interactive Computing Systems (EICS'09)*, 2009. ACM, New York, USA.

[17] Taylor, R. M. II, Hudson, T. C. Seeger, A., Weber, H., Juliano, J. and Helser., A. T. (2001). VRPN: a device-independent, network-transparent VR peripheral system. In Proc.of the ACM symposium on Virtual reality software and technology (VRST '01), 55-61, 2001. ACM, New York, USA.

[18] Johann Habakuk Israel, Oliver Belaifa, Adrienne Gispen, and Rainer Stark. 2010. An object-centric interaction framework for tangible interfaces in virtual environments. In Proc. of the fifth international conference on Tangible, embedded, and embodied interaction (TEI '11), 325-332, 2011. ACM, New York, USA.

[19] Andy Wu, Sam Mendenhall, Jayraj Jog, Loring Scotty Hoag, and Ali Mazalek. 2012. A nested APi structure to simplify cross-device communication. In Proc. of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI '12), 225-232, 2012. ACM, New York USA.

[20] Echtler, F. and Butz, A. (2012). GISpL: gestures made easy. In *Proc. of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI '12)*, 233-240, 2012. ACM, New York, USA.

[21] Wallace, V. L. The semantics of graphic input devices. In *Proc. SIGGRAPH'76*, 61-65, 1976. ACM, New York, USA.