J Multimodal User Interfaces (2018) 12:125–143
https://doi.org/10.1007/s12193-017-0253-z

ORIGINAL PAPER

Multimodal human-machine interface devices in the cloud

B. Estrany[1] · C. Marin[1] · M. Mascaró[1] · A. Bibiloni[1] · Y. Luo[1]

Received: 27 April 2017 / Accepted: 6 November 2017 / Published online: 14 November 2017

Abstract In an increasingly connected and multidisciplinary world, we propose a new paradigm of web application development. It makes the whatever modal input to use the same interface to connect to the applications. It can essentially free the web application programmers and the end users from the need of physically handling the data input devices when they are building a multimodal system. The same application can be used for a whole range of physically different peripherals, but similar from the logical point of view of data entry. This paper discusses the implementation of a pilot project, currently in a local network environment, where all the devices in the LAN are identified and described in an interface server. Users in the local network may, upon request, make use of such devices. The communication of these peripherals with the web applications will be carried out by a network of modules that run under the websocket technology. The whole process of communication and connection establishment is automatic and guided by the existing configurations in the interface server. The entire platform runs under SOA strategy and is fully scalable and configurable. Its use is not limited to games because it has much wider possibilities, interactivity in teaching, accessibility for people with special needs, adaptation of web applications to the use of uninitiated, etc.

Keywords Multimodal · Interface · SOA · Websocket · Virtual interface · Accessibility · MVCI

✉ B. Estrany
tomeu.estrany@uib.es

C. Marin
carlosmarinfernandez@gmail.com

M. Mascaró
mascport@uib.es

A. Bibiloni
toni.bibiloni@uib.es

Y. Luo
y.luo@uib.es

[1] Universitat de les Illes Balears, Palma de Mallorca, Balears, Spain

1 Introduction

In this article, we propose a new paradigm of interaction in the cloud based on a new design pattern that extends the MVC. The Internet of interfaces (I-o-I).

In the last decade, we have witnessed a spectacular development of the web favored and enhanced by improvements in the communications sector and mobile terminals. We started migrating applications from the desktop to the cloud, and then, even the physical storage media. Today, it is possible to configure a virtual computing system with its virtual CPUs, its virtual memory and its applications and services all in the cloud.

Important advances have been made in computer systems since the presentation of Douglas C. Engelbart in 1968 [3]. Despite these achievements in such aspects of computer systems, we are still using the input devices (basically mouse, keyboard and lately touchpad) that, although full of technology, they still are highly primitive for many purposes of interaction. For other devices, they may be configured only for a particular operating system or even a single application.

In addition, these interaction devices are firmly linked to the operating system of the machine running the applications (yes, even being wireless), so that it might become difficult to control them from a web application and gain access to them directly as, for security reasons, the proprietary resources of the OS are still protected by the system. To overcome all these difficulties, this article details the proposal of a new

paradigm, with the aim of facilitating how to connect and use the interaction devices and facilitate a more ubiquitous and versatile interaction with them.

To this end, we have designed an architecture that takes the advantage of some of the new features of HTML5, such as websockets and uses Service Oriented Architecture—SOA concepts for a simple and highly modular and completely reusable implementation. For now, it is a proof of the design concept with a set of implemented examples.

Using these new application development tools, we will be able to deal with the emergence of countless multimodal sensors and low cost devices connected to the cloud by the evolution of IOT. For the versatile connections of input devices and the possibility to interact with them, we propose a new concept: The Internet of Interfaces, or I-o-I (as a memory of those first computer games).

It is possible to object that the interaction devices are already part of the cloud, which are already connected to the internet. This is true in part, but as a negative aspect most of them still need drivers for each of the operating systems that drives them. In addition, they must be directly supported by browsers for their potential use.

Throughout this article, the reader will be able to observe that the fact of disconnecting the interaction devices or any type of sensor from the operating system supporting them, has important advantages in the reuse of code and in the incorporation and adaptation of new interfaces to the existing web applications.

1.1 State of the art

Some protocols have been developed that help to control remote devices and to interact among them such as OSC [1] and TUI / OSC [11] and lately powerful frameworks like Spacebrew [10], although they are specific for very particular applications. These works in open source, are of an extraordinary quality and are in a state of very advanced development. But they do not cover, or only partially cover, the aspects exposed in our article.

In fact, we could certainly use them in the architecture we describe. On the other hand, there is an attempt to describe what has been called the Full Device API, but it is in an embryonic state, each company seems to opt for its own solution.

Of course, a lot of effort is being made to integrate various elements of interaction and develop new protocols and architectures that allow their use [2,5,8]. In fact, the W3C has a specific group for this work [7,12]. In this context, we want to contribute with our grain of sand. For this, we have developed a proof of concept that allows the creation, access and use of different devices of multimodal interaction giving preference to, above all, simplicity, ease of use and possibility of growth with a SOA model.

2 The scenario

To develop the proof of concept of this architecture, we have designed a scenario where to apply this system. The initial application environment would be a family home with its local router and internet access, with various nodes (desktop computers) connected through this local network and input devices or sensors connected by Bluetooth, Wi-Fi, USB and/or serial connection. There could be devices that allow browsing the network (web clients) including smartTV tv screens, video game consoles, mobiles, tablets, etc…

Our purpose is to use these devices as ubiquitously as possible from web pages or from applications that can use these services in a local network environment. That is, the connections to the interaction devices will always be within the local scope in this first interaction scenario. However, using a remote interface in this context does not make much sense. That does not mean that architecture does not allow it, surely you can design scenarios in which this scheme could be useful with remote interfaces.

2.1 Architecture

Analyzing the different possibilities of implementation with the test examples that we developed for the design of the architecture and that will be explained further on, it is interesting to distinguish between different connection modules according to the functionalities offered by these.

Thus, we will have a collection of modules that can be executed independently of their location in some cases and, in others, they are anchored to an operating system, although accessible by any device within the local network. The different modules that our architecture contains according to their function are the following:

Source module: This module is responsible for activating or reading directly from the sensor or interaction device (Arduino, webcam, etc…) and offer its information through a websocket to the web client application that requests it. Arduino is an open hardware initiative that unifies and facilitates the programming of microcontrollers [16].

Processing module: This module can be connected to any other module either source or processing. Its function is to adapt the information of the source to the information acceptable by the client module. It connects to an input websocket and offers an output websocket. Another important function is to perform the processing of information (in some cases it may require a lot of CPU) in specialized or specific nodes of the network to take advantage of their computing power (deep learning, vision analysis, etc.). If this is the case, the module will run on a node with the necessary calculation capabilities. Note that in this way, it is possible to perform different parts of the input processing in different nodes of the network. They could also be external to the local network.

Unification or grouping module: In case of needing more than one interaction device, either multimodal or multiple interaction users, this module will be responsible for grouping information from different sources. It connects to different websockets and offers a single output websocket with unified information. In this way, a mouse could be simulated from three different inputs, two dimensional and one of the button type, located in the local network.

The virtual interface server module: This is another special module that allows multiple virtual interaction devices to be connected to it and offer these elements through a single websocket. In general, web pages designed as virtual interaction devices will be connected to this module and for the purpose of connecting through this module to the client web pages. In principle, a single virtual interface module is used for all interface clients and virtual interface pages. Although it would be possible to use several if necessary.

In Figs. 1 and 2 you can see the graphs with the functional information of the different modules.

In Fig. 1, there are two possible connection schemes. In the upper part a webcam and an Arduino are used for the interaction. For this, two source modules are used to control each device and later two processing modules will calculate positions or mouse clicks that are grouped in one end module that provides the interaction information to the client page. The second scheme is simpler, the information of the device is collected and adapted to be presented to the client page.

Figure 2 illustrates different virtual interface connections with their respective pages or client applications. The virtual interfaces are web pages that collect actions on buttons or sensors of the devices executing them. These pages send the interaction information to the virtual interface server which, in turn, is responsible for distributing this information to the pages or web client applications.

2.2 Websockets servers

Most modules in the system as a whole are located on the websockets servers. A websockets server is a local network node that contains one more global system modules. These modules either contain one or more interaction devices connected to it, or perform a calculation or processing on other websockets servers in such a way that the end clients or other websockets servers connect to them. Websockets servers, normally, contain various system modules.

It can also be a standalone device that already has in its firmware the possibility of serving information through websockets. The Arduino Yun [15] or the ESP32 microcontroller [17] are examples of this case.

A special websockets server is the virtual interface server. This performs the functions of bridge between all the virtual interfaces connected to the system and all the client applications connected to them.

2.3 Client applications

Client applications are those that incorporate a small piece of code, which allows them to access the Web Interface Address Server (WIAS) and then, from this information, to connect to the appropriate websockets server. Once the connection is made, they obviously interpret this information to give it an appropriate use.

In general, they will be web pages that want to use the interaction devices but they can also be applications installed in a specific system as it is the case of one of the examples that are exposed in the article. They can also handle virtual mice and keyboards on a conventional computer with some added software.

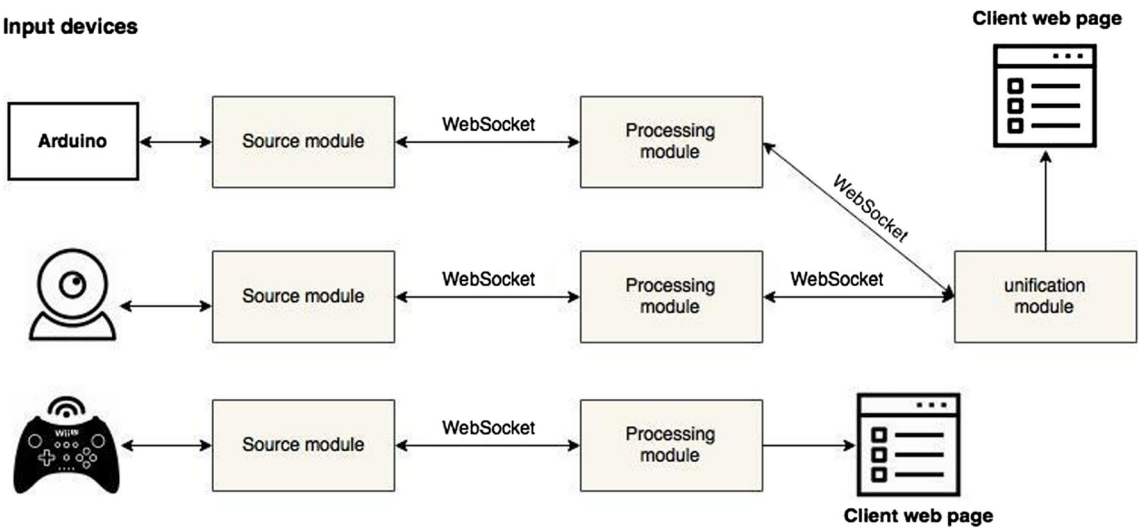

Fig. 1 General functionality scheme of different modules in the system

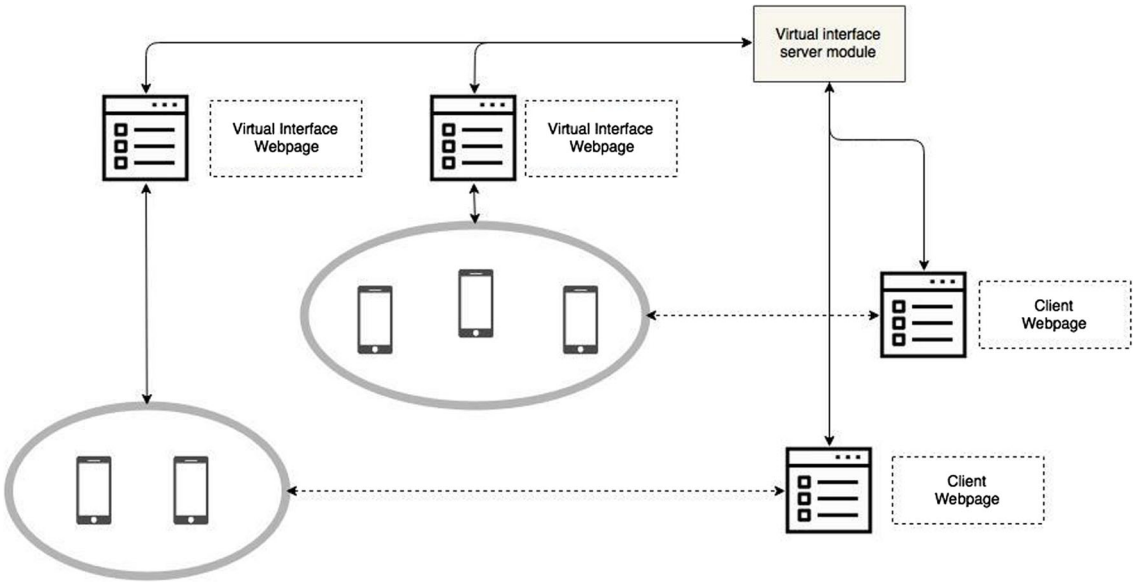

Fig. 2 Different virtual interface connections

A special case is those web pages acting as virtual interfaces and that, in essence, are the interaction servers. But in our architecture, they are treated in the same way except that when being connected to the websockets server, they will indicate their own function and the client page to which they would send their interaction data. The virtual interfaces module will perform the data sending action.

2.4 Implementation of websockets servers and their modules

For the implementation of websockets servers, we have used an open source tool that, although simple in its implementation, solves many of our problems in an elegant way. This is websocketd [13]. Websocketd is an implementation of websockets in the style of the old CGI that were used in the first web servers, developed by Joe Walnes. This tool allows to implement, in a very simple way, the different websockets servers that our architecture needs as well as to be able to provide basic services of http and CGI's.

Websocketd allows to establish a websockets server for each of the nodes that have anchored physical devices (either USB, Bluetooth, webcams, audio, microcontrollers with sensors, etc…) or processing or clustering modules, in addition to the Virtual interface server as follows.

The websocketd has a directory assigned with the code (scripts or executable programs) corresponding to each of the modules that this particular node can activate. The code, as in the old CGI's, can be written in any language that the operating system of the node can execute. According to the occasion, we have used shell, node.js and python although any other language can be used. The use of scripting languages can help with maintenance and unification tasks, although in some specific cases, if a high CPU is required, it may be advisable to use compiled code directly for the native OS of the websockets server node.

The code of each module is programmed using I/O as the standard inputs and outputs of the system, that is input output by console and admitting input arguments. Then the websocket will redirect you to the client's connected websocket just as you did with the old CGIs.

To illustrate the concepts, imagine that in one of the nodes we have an IR camera connected by USB and also an Arduino device that has in turn two potentiometers also connected by USB.

On the other hand, we want to play a two-player pong game. This game only needs two variables corresponding to the position of each of the blades of each player. In short, two variables that control the position of the paddle of each player. We want to play either with the potentiometers or with two infrared pointers in front of a webcam.

In order for the processing modules to activate these source modules (webcam or Arduino) and make use of the information they provide, the processing modules must know the following:

- The IP address or DNS of the server node that contains them.
- The name of the module that manages them.
- The address of the device associated with the node.
- Other values necessary for the activation of the devices

So to activate the Arduino device connected to the serial port COM3 of a computer with windows can write:

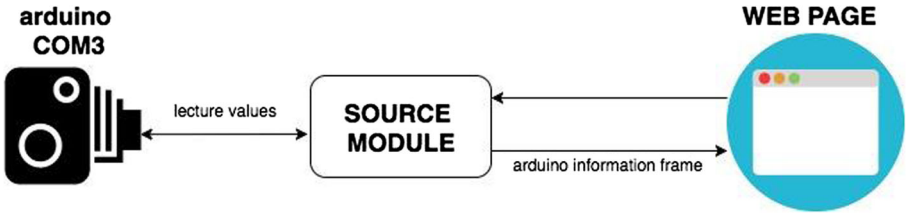

Fig. 3 Connection between an Arduino unit with two potentiometers and the two-player pong game web page

ws://192.168.1.1:8080/arduino?COM3,9600

The Arduino module code simply consists of the instructions needed to open the given serial port, at the specified speed, and then print values by the standard output. It should be noted that the addresses change depending on the operating system of the node but the same code can be used to activate more than one device with similar characteristics.

In the case of the Arduino source module, it is possible that with the values of the potentiometers readings that are sent through the websocket it would be sufficient to handle the client web page. For example, our two-player pong game each of them driving one of the two potentiometers connected to the Arduino. In this way, the client web page can be connected directly to the source module, if it knows all the parameters for the invocation of this particular module, and interpret each plot as the position of the paddle of each of the players, completely independent to the hardware features of the original source (see Fig. 3).

In the case of a source module that controls a webcam or audio, we use internally for its control FFmpeg [18] (FFmpeg is an open source software for the management and treatment of audio and video streams as well as their formats among other things) which allows us a great variety of formats and options for each occasion. For activation of the module, we will also need some parameters to identify the device to use and its frame rate, size, etc.

Thus, in order to obtain images of the first camera connected to the node, we could use a module called webcam and write the following:

ws://192.168.1.1:8080/webcam?0,1024,768,20

Where, the parameters 0,1024,768.20 correspond to camera number, horizontal size, vertical size and frame rate, assuming that it was housed in the same node as the previous example. Of course, in the actual implementation some more parameters are needed but for clarity we have decided to omit it here.

Note that the information provided by the source module, in the case of a webcam, is not enough to control the same pong game that we used in the Arduino unit connection example with the two potentiometers. Now we need to have two variables that give us the position of the infrared pointers and what the module provides is a video stream that, yes, contains the information of the infrared lights detected through the video images of the Webcam in question.

In order to extract this information from the images it is necessary to perform a process on each of the frames of the images to extract the position of each of the points appearing in each frame.

So as to carry out this process, a processing module is needed that is capable of extracting the coordinates (x, y) of each of the points that appear in each of the images. With the help of OpenCV [14] (a powerful open source package for image processing and processing), it is possible to perform this operation in a simple way. Our treatment module will read the images of the camera from the input and print out the detected points in each frame.

It seems a good idea to have this module isolated and available for point detection in our system, but the output is not suitable for controlling two-player pong. We will call it **pointsdetector**.

We need a single variable for each player and now we have two for each one so we would add another processing module connected to the latter that would simply treat each point and assign the position to each of the players taking into account the position of the points detected. The y of the minor component **x**, corresponds to the player of the left and the other and, with the component **x** greater, to the player of the right. We call this module **webpong2p**.

In short, in the case of playing pong with an infrared detective webcam, you need the following:

- Two LED pointers, one for each player.
- A webcam.
- The address, name and connection parameters of the source module.
- The address and name of the processing module that detects the points.
- The address and the name of the final processing module that extracts the components for each player.

Thus, the module that will connect the pong client is as what we have described, the webpong2:

ws://192.168.1.1:8080/webpong2

This, in turn, must be connected to

ws://192.168.1.1:8080/pointsdetector

And finally, this will connect to the webcam with

ws://192.168.1.1:8080/webcam?0,1024,768,20

It happens that who initiates the request of service of websockets is the client page, whereas the functional modules really do not know the connection chain between the parts because they have the possibility to connect in multiple schemes of connection and between multiple devices. For this reason, the client web page must know in advance the connection chain of all the modules involved in this service. Therefore, we decided that the connection string will be made through the passage of parameters to the modules. Thus, the connection with the final processing module (always being backwards) will be

ws://192.168.1.1:8080/webpong2?
ws://192.168.1.1:8080/pointsdetector
ws://192.168.1.1:8080/webcam?0,1024,768,20

For clarity, we have separated each expression in a line but the strings are only separated by a space between them.

Each module, upon receiving the parameter, removes its connection part and invokes the next string, so webpong2 will invoke **pointsdetector** with

ws://192.168.1.1:8080/pointsdetector?
ws://192.168.1.1:8080/webcam?0,1024,768,20

And finally, pointsdetector will invoke webcam with

ws://192.168.1.1:8080/webcam?0,1024,768,20

Note that in Fig. 4, in the case shown, an already developed module is used which is the point detector POINTSDETECTOR. The WEBPONG2 only extracts the **y** positions of the detected points.

There are alternatives to make the connection string that do not need to pass the entire connection string as a parameter, for example, by defining a unique identifier for each connection scheme that is actually contained in the WIAS server that is explained below. But this complicates the programming of the different modules because it involves an extra connection of each module with the WIAS with the consequent increase of code and complexity for each processing module. Another option is to include the whole process in the source module but it does not seem to us an alternative that promotes modularity. Although, sometimes, if there is a lot of information, it may be an option.

In a second example, we will illustrate the use of two mobile phones for a two-player pong game using the virtual interfaces module. All client web pages wishing to use the virtual interfaces must first connect to the virtual interfaces server, indicating the number of virtual interfaces they accept (number of players in our case) and a unique identifier supplied by the WIAS that will serve to address the Interfaces to their respective client applications. Thus, in the two-player pong game, once a virtual interface is selected suitable for the game through the WIAS, it will have a unique identification code which performs a service request on the virtual interfaces module:

ws://192.168.1.1:8080/virtualmodule?c2034,2

So the virtual interfaces module will recognize that a client application (client c and identified by 2034) has been connected, waiting for the connection of two virtual interfaces.

In turn, the virtual page informs the devices acting as virtual devices of the page http that performs this function adding also the corresponding unique identifier as follows

http://192.168.1.1/virtualinterface1?virtualmodule, 202034

The virtualinterface1 web page will be uploaded and this, in turn, will be connected to the virtual interface server through a websocket identifying itself as the server of the client page 2034. It will make the connection:

ws: //192.168.1.1: 8080/virtualmodule? S2034

In our example, the web page that is loaded on mobile phones only registers the position of the phone on one of its axes and transmits it through the websocket. Each of the virtual interfaces sends the information to the virtual interfaces

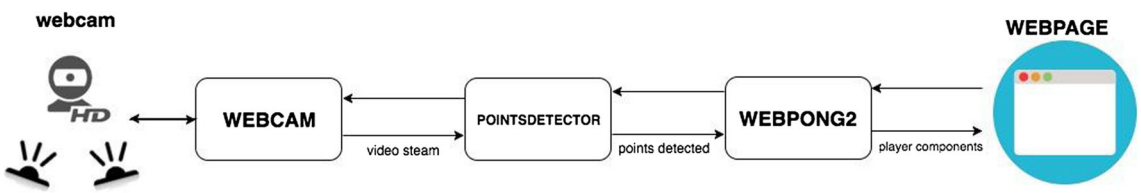

Fig. 4 Connection between a webcam with two IR pointers and the two-player pong game

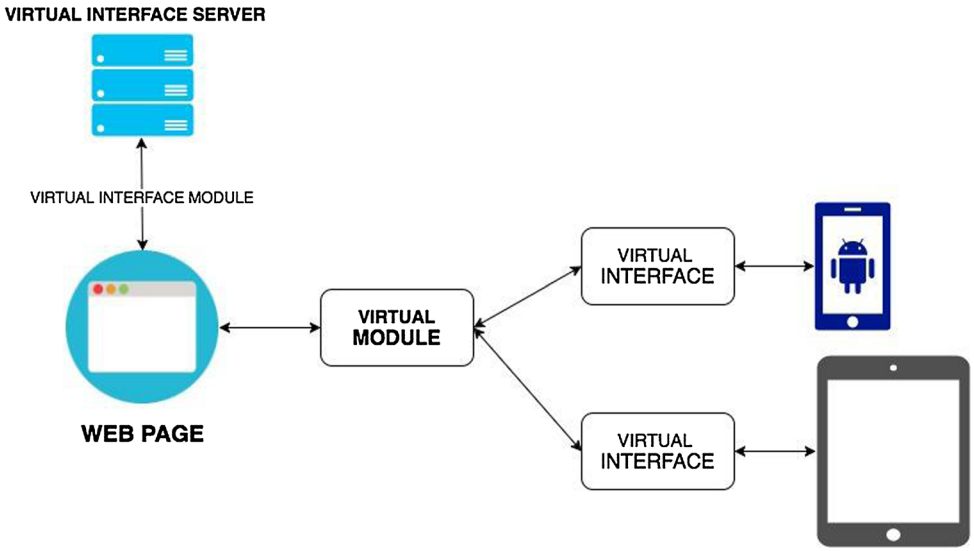

Fig. 5 Connection of two virtual devices for the two-player pong game

module which, in turn, distributes the information between the different connected client applications.

In the case shown in Fig. 5, the client page requests two virtual interfaces to the WIAS, this gives an identifier and tells the users of the mobile which virtual page to connect in order to interact.

2.5 Connecting the interaction devices

There are several ways to connect physical and virtual interaction devices to a local network depending on the type of device.

Most of the existing ones need to be connected to a computer if they are not already connected to them. As an example, a web camera connected by USB or integrated directly to a computer that is part of the local network. The same goes for Arduino sensors or other microcontrollers. They are in general connected by USB or, in their case, by Bluetooth. By Bluetooth, the problem is conceptually the same, but without cables. The device is paired to one of the computers and only accessible from the computer.

A different case is a device that has a Wi-Fi integrated in it (for example an IOT). In this case, the device can be accessed directly by the local Wi-Fi network. There is no need to have a websocket server on one of the nodes. We hope that in general the IOT devices tend to be this type of connections. With the current developments in Open Hardware, it is possible to use devices of this type for a reasonably low price (ex. ESP32).

Thus, we will have a set of interaction devices or sensors that are anchored to the different nodes of the network and that need specific source modules for each specific OS. Therefore, as already mentioned above, each of these nodes will necessarily need a websockets server to store these modules. On the other hand, there will be the devices that already manage websockets to be used in the local network and, these, do not need to be housed in any of the nodes although it is necessary to have the meta-information of their capabilities.

Anyway, in all cases, we need to know in which direction it is and its port (your websocket) in order to access the interaction information that can generate us.

This is one of the most important functions of WIAS, this server is responsible for maintaining the information of the devices available for interaction and the modules accessible on the network as well as their connections.

It is important to have a fixed domain name as web pages are not allowed by default to access the local network in which they are running although it can make requests to servers that are on it.

2.6 The WIAS server (Web Interface Address Server)

The WIAS server (our I-o-I server) keeps remote peripheral information available at all times. It is a concept similar to the cloud but in a local environment, on a LAN. When a user (web client) wishes to use a remote peripheral, it consults WIAS about the actual availability. It answers the user with all the features of such peripheral and cluster processing modules, or also the virtual interfaces necessary to establish and maintain a connection. With them, the client simply has to connect and start using the device, as seen in the Fig. 6.

In terms of architecture there are two logical levels: the level of maintenance and exploitation of the availability information of peripherals and, the level of real connections to different peripherals.

The WIAS server, therefore, is responsible for performing the directory tasks of the interfaces that are available and already installed on the local network (LAN).

The main task of WIAS is to provide client web applications with the information they need to get in touch (ie create the communication link) with the desired peripherals

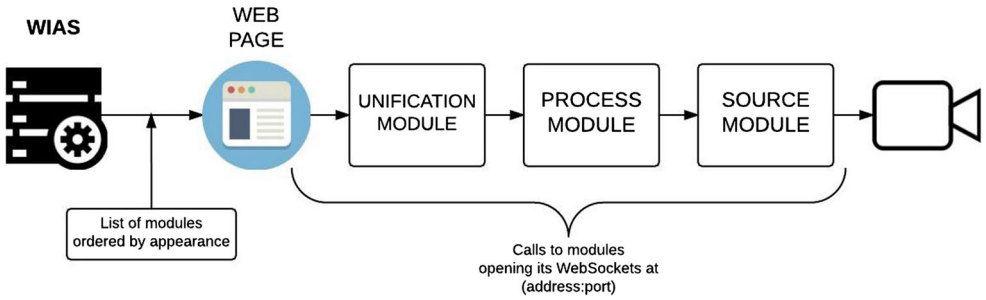

Fig. 6 The connection information provided by WAIS to a client page and the activation of the modules

that are distributed in the LAN. It is equivalent to a DNS in the network, which is able to associate names with physical addresses. The following is how Web applications create links to peripherals via WIAS addressing.

2.7 Creating the peripheral-web client link

When a web application starts, it can be in two possible situations from the point of view of the knowledge of the interface devices to use.

1. Never run before: in this case the web application does not have the necessary information to connect to the peripherals, so you must obtain it by making a request to WIAS.
2. The application has already been executed other times: in this second scenario the client application already has the connection information to the peripherals, so it can choose to use this or request new data from WIAS. In this case the client application makes use of the new features of HTML5 that allow to save information in a local repository of the client machine.

2.8 Request information from WIAS

The WIAS server located in the same LAN has a fixed name in the local network which is accessible and known by any client of the LAN that wants to use its services.

When a web application starts up and is in the "1" situation described in the previous point, or if it is in the "2" situation and you want to change physical (peripheral) devices, you must make a request for information about available devices through a SOAP query. The WIAS responds with a list of available web peripherals that are compatible with the features that the client will have specified in the query. These characteristics specify the client's requirements regarding the interfaces. For example, when the client queries the availability it will indicate that it needs two devices type range of values and a button.

Once the client receives the list of peripherals of the WIAS, it selects the ones that you want to use and establish with them a communication via websocket (Socket-D), that is to say, a direct link with them where they will receive the information of the use of the peripheral. From that moment, the peripherals can already be used by the client application. All this protocol can be seen graphically in Fig. 7 or more simplified in Fig. 8.

It does not matter where hardware devices are physically installed. The only requirement is that they are accessible on the local network and registered with both their characteristics and their connection information in the WIAS database.

2.9 The MVCI design pattern

The development described in this article actually represents a new gear in the Model–view–controller, MVC design pattern.

What is proposed in this work is to free the programmer and the user of web applications from all the details concerning the physical aspects of the peripherals and configurations of the operating system, so that it only has to pay attention to the possibilities that it offers the established standard.

For instance, if a web client running an Arkanoid type game (The Wall) wants to use an interface-type peripheral to govern X-axis motion, it will simply ask for the devices available to the server (range type) and once one is chosen, a connection will be established between the two of them via websocket. In this way, from the point of view of the web application, all the peripherals of the same type are the same, whereas from the physical point of view, a peripheral that provides values for a displacement in x can be a keyboard, a mouse, a joystick, a motion detection camera, etc.

We believe that our work presented here is placing a new branch in the MVC model, the concept I (Interface). A new world of possibilities in the field of interaction of users with web applications is opened with this new paradigm.

2.10 Implementing the WIAS server

To carry out the development of the WIAS server we have chosen to make use of a service-oriented architecture (SOA).

The server is in an application container, capable of performing the functions of web server which will enable the

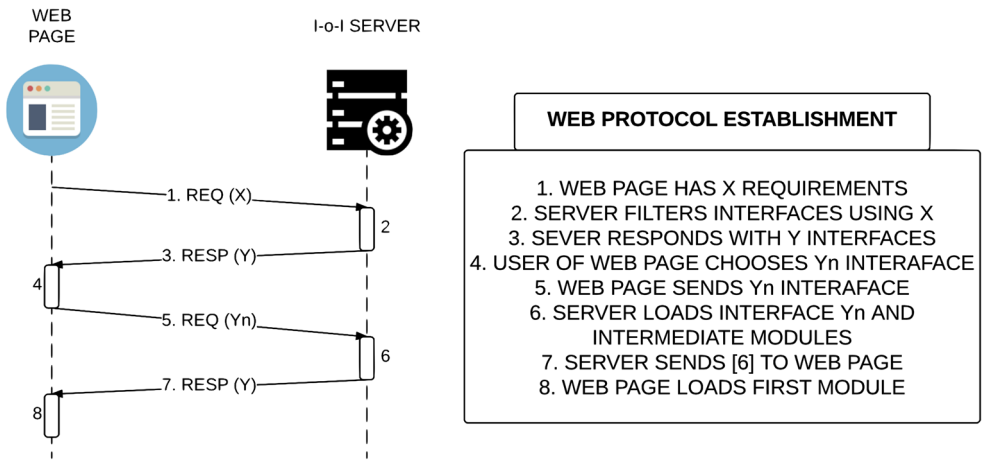

Fig. 7 Process of a client page requesting an interface

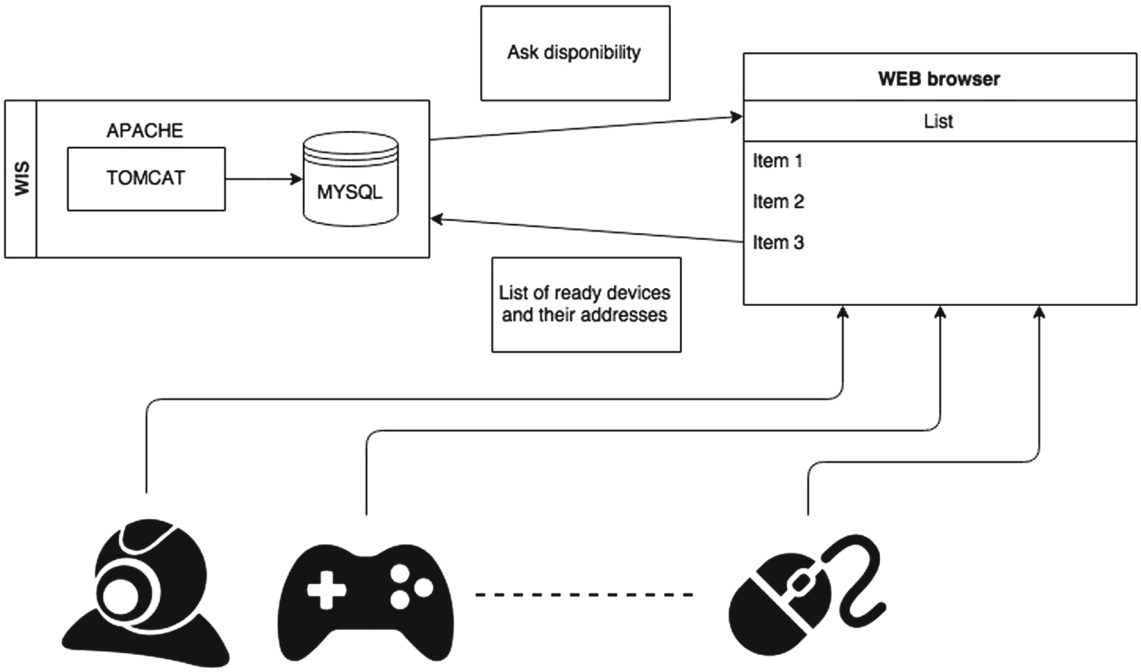

Fig. 8 Sequence of an availability request to WIAS

creation of an intranet that will facilitate the maintenance of the database of the WIAS server, and on the other hand, to provide information by making available of the LAN a series of web services.

The database, situated in a MySQL DBMS, contains the necessary information to carry out the deployment of the new defined development pattern MVCI. This database is accessible via Java links for the purposes of the server itself and via web services for potential clients who request it from the local network. Such clients can be both web applications and interface control modules. It is possible to use other servers although in the initial prototype we use this one. See Fig. 9 for more details.

2.11 Logical interface layer

The I of the new MVCI pattern takes shape from the creation of a logical scheme of types of interface information that the user can receive. This set of inputs is sufficiently formable so that, in the future, it could allow new extensions of new types of information. In the beginning, the following types of interface inputs are established:

- range: returns the value of a dimensional component, which can vary between a maximum value and a minimum value (min and max attributes).

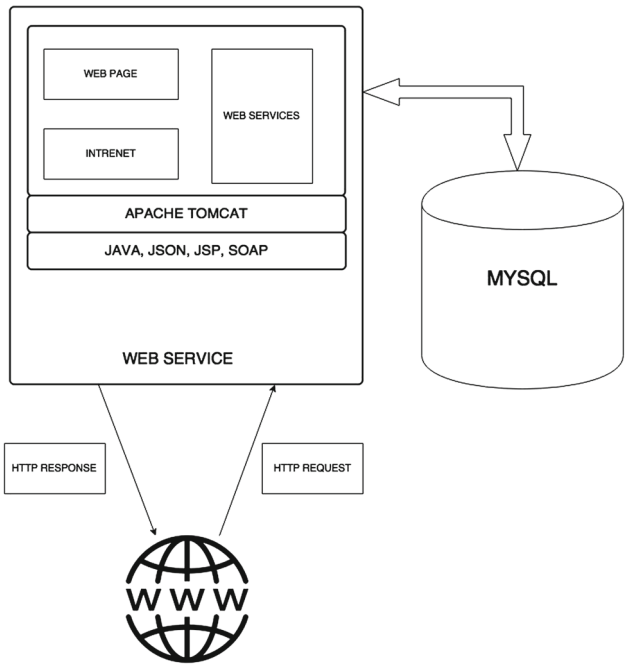

Fig. 9 General scheme of WAIS

- streaming: informs about the address from which audio and/or video transmission data are obtained (URL attribute).
- img: sending image type information (null attribute).
- keyboard: string information (null attribute).
- button: discrete type information set capable of sending the following events: press, release, click, double-click.
- time: timestamp.

According to this set of definitions of information types we establish the following types of virtual interfaces:

1. range1: provides a range-type value.
2. range2: provides two range type values.
3. range3: provides three range type values.
4. img: provides image information.
5. streaming: provides audio information.
6. streamingv: provides video information.
7. keyboard: provides character frames (string).
8. button1: provides button type information.
9. button2: provides information for two buttons.
10. mouse1: provides the information corresponding to a button and two ranges of values.
11. mouse-wheel: provides the information corresponding to three buttons and three ranges.
12. time: timestamp.

According to the above, you can check the ease with which you can declare new types of inputs and combine these into new definitions of web interfaces. Thus, we can define a Database schema that can persistently contain and maintain a malleable structure of sets of interfaces and their characteristics.

The final DATABASE contains much more information, but in terms of the logical layer of web interface management two “infotype” and “device” tables are defined. The first one contains everything necessary to satisfy the requirements of the web clients that want to use the peripherals of the LAN, such as their access address, interaction characteristics offered (button, range, …) and their current status (busy/available).

All this information will serve to offer a collection of skillful peripherals when the user requests a type of interaction device. In the second of these tables is the information regarding the connection parameters needed by the different logic modules. These parameters are used to establish the physical link and the processes that control the input of data and direct it from the physical peripheral to the web application.

In addition to these two tables, other tables for the management of connections are in the database, since at all times the system must know the actual state of the connections, who uses each physical peripheral and what modules are connected and with whom it is connected. In addition, it also has information in a history log, as well as the interaction data with the device type, in order to collect statistical and control data.

The process for requesting a LAN device is as follows: when a web application requests information from WIAS about the availability of a series of devices that meet its needs, it responds with a list of available devices that conform to those requirements. Thus, in this scenario we can establish the communication protocol that will be explained in the following session.

In Fig. 10, the client webpage can perform page requests over the entire web. If it is in the local mode, it can only ask for the devices defined in the local environment.

2.12 WIAS HTTP protocol

There are two states in the HTTP protocol that can be established which are “Request” and “Response” according to the state. There are two different information frames corresponding to these two states.

“Request” operation launches the client web to the WIAS through a soap request to a Web Service:

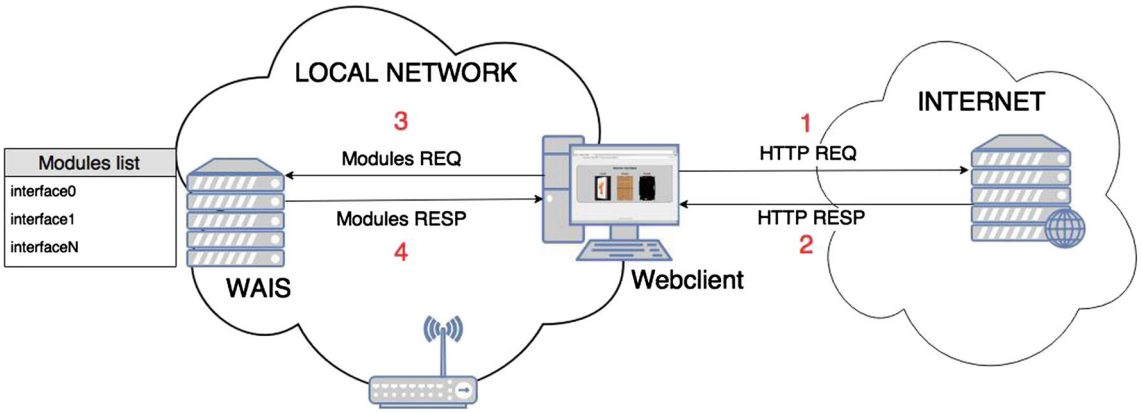

Fig. 10 Overview of the system taking into account the local network with the WWW

```
{
    "REQUEST" : {
        "ASK-FOR" : {
            "RANGE" : 2,
            "BUTTON" : 1
            .
            .
            .
        }
    }
}
```

Corresponding to this request the WIAS server responds with the following JSON frame:

```
{
    "RESPONSE" : {
        "I-HAVE" : {
            "DEVICE" : {
                "ID" : 1,
                "URL" : "SOCKET URL",
                "DEF" : "DEFINCIÓN JSON"
            },
            .
            .
            .
             "DEVICE" : {
                "ID" : N,
                "URL" : "SOCKET URL",
                "DEF" : "DEFINCIÓN JSON"
            }
        }
    }
}
```

When a page needs to connect to a server, it first queries the WIAS to know which ones are available and chooses the one the user needs. Then, the chosen server will give the service address and the page access to the socket server that connects to the desired device.

The server maintains the meta information of all the interface servers scattered over the local network.

2.13 Overview of the system

Figure 11 shows a global view of how the system works in an sample process. It passes all the major components and the connection processes from step 1 to 4.

Step 1.- A web page is loaded. It requests a list of available physical and virtual interfaces.
Step 2.- The user chooses an interface. The WIAS returns the entire path of the websockets.
Step 3.- The client page activates the process module A which in turn, activates the source module B.
Step 4.- The process of sending information from the WEB-CAM7 to the application web is started through all the intermediate modules.

2.14 Examples for the proof of the design concept

This section presents some of the interaction interfaces we implemented through the websockets that illustrate the feasibility and potential of our framework. Except for the control example of an operating system, all the others are web pages that code the games in JS. These games were picked up from the internet and adapted to our interface via websockets using our concept.

2.14.1 First example: Pong set with analog potentiometers

Using our user interface concept, this example shows much more precise action than the mouse or keyboard with a

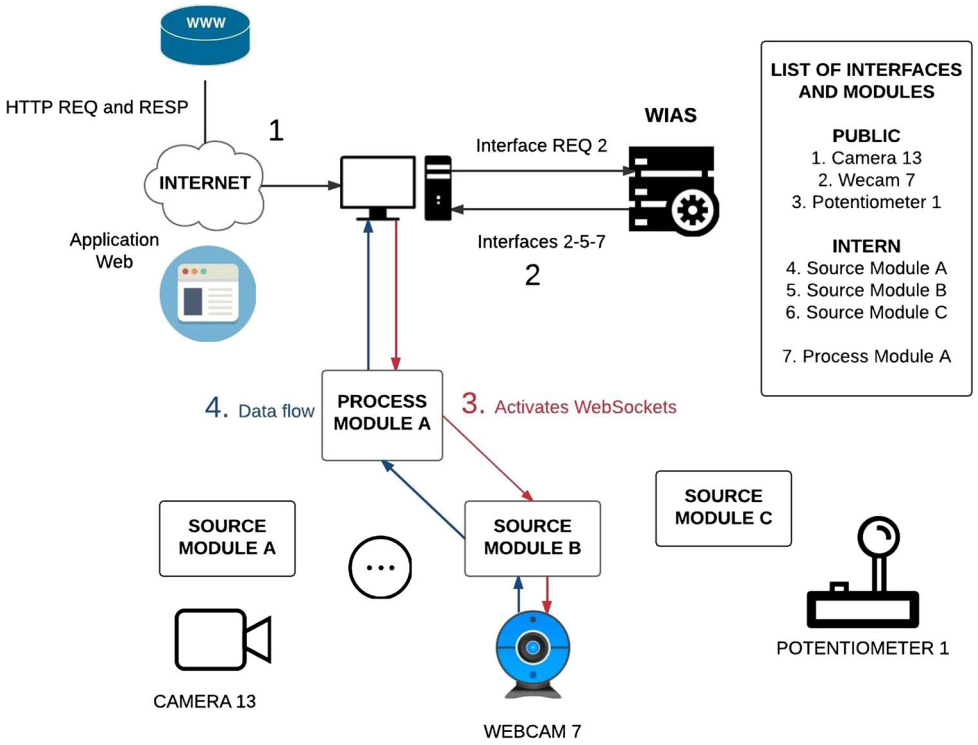

Fig. 11 A global view of the system process with all its components and the connection

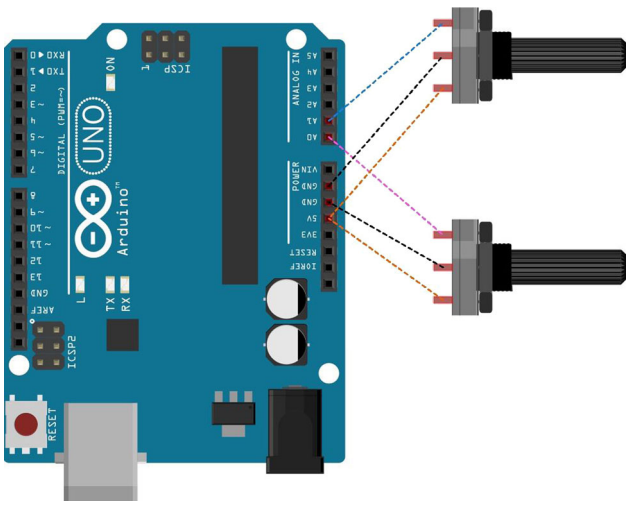

Fig. 12 Connection of the potentiometers with the Arduino unit

smoother response. The game consists of a webpage to play the old and well-known pong game. When loading the game page in the browser, the page is connected to the given websocket and interaction occurs through the potentiometers. In the server interface system there is an open websocket that supplies the position information of each potentiometer. The connection scheme of the websockets can be seen in Fig. 3. The sensor module is an Arduino unit connected by USB to the system that sends the position of each potentiometer at regular intervals. The position information is then offered to the websocket clients. The connection diagram of the potentiometers with the Arduino unit is illustrated in Fig. 12.

In Fig. 12, the analog values A0 and A1 are read, with a range of 0-1024 that are sent in pairs via the serial port to the source module.

2.14.2 Second example: the balance game

The game was adapted from a web page to play with the scale. An old scale was used taking advantage of its sensors. We designed a small signal conditioning module for the amplification of the signal in the platform before connecting with the Arduino unit. In this case, the Arduino unit sends 4 values corresponding to the weight readings of each of the sensors. And, through these four values, the user's center of gravity used by the game can be successfully calculated as shown in Fig. 13.

2.14.3 Third example: Pong game using the center of gravity of a player

To control this pong game we use the interaction device that we developed in example 2 with four weight sensors. It uses the same client page as example 2 to play, but connecting to another service of the same server which shows the reusability of the interface concept. In this case, it measures the center of gravity of the player to move the paddle in the game. To make this pong game works, it requires an intermediate processing module that translates the information to a single vertical variable. This corresponds to the vertical center of gravity of the player (see Fig. 14).

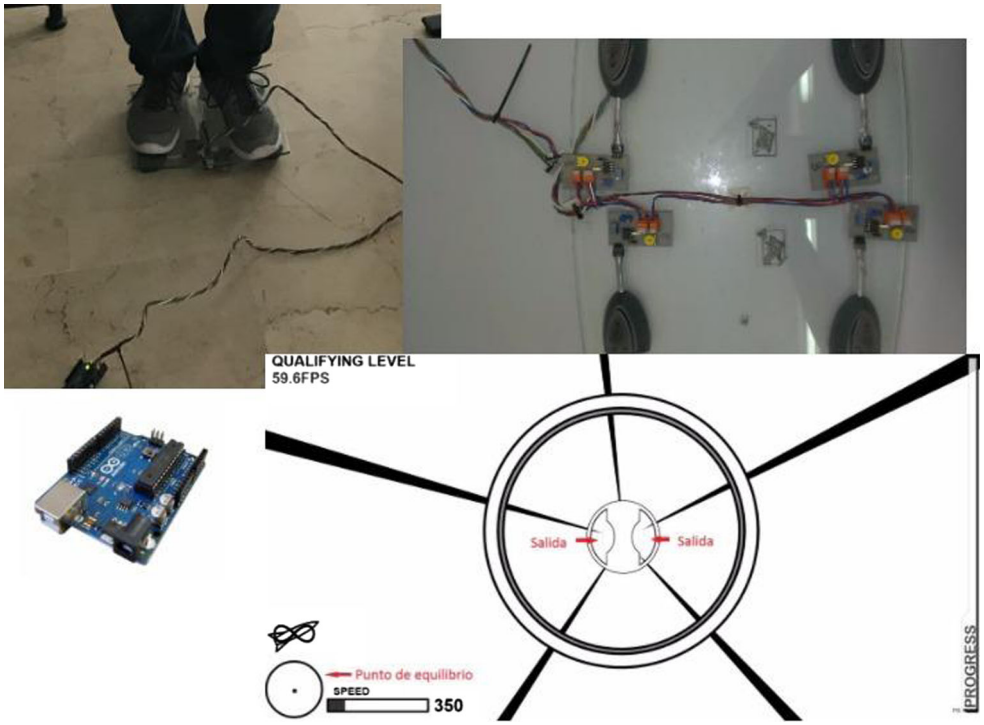

Fig. 13 Components of the balance game using the center of gravity of a player

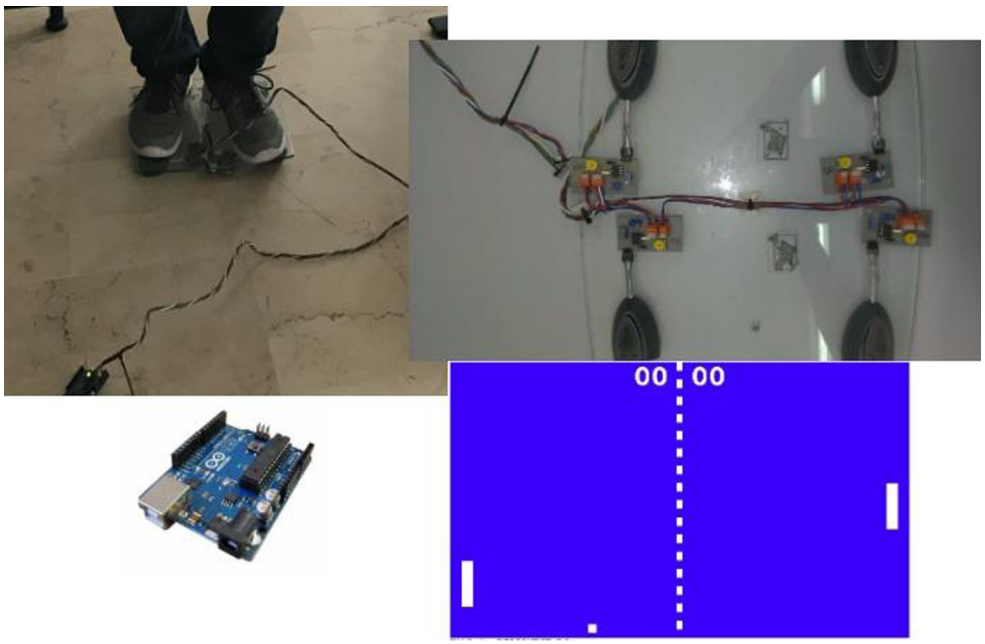

Fig. 14 Components of the pong game using the center of gravity of a player

2.14.4 Fourth example: Pong game with mobile phones for 4 players

This is a game scheme which was illustrated earlier in this article and corresponds to Fig. 3. Four players can play together using their mobile phones as the paddles in front of a projector screen. The web page corresponding to the game requests a virtual interface for four players and connects to the virtual interface server. In addition, it shows the address in the web page of the virtual interface for the mobile phones that

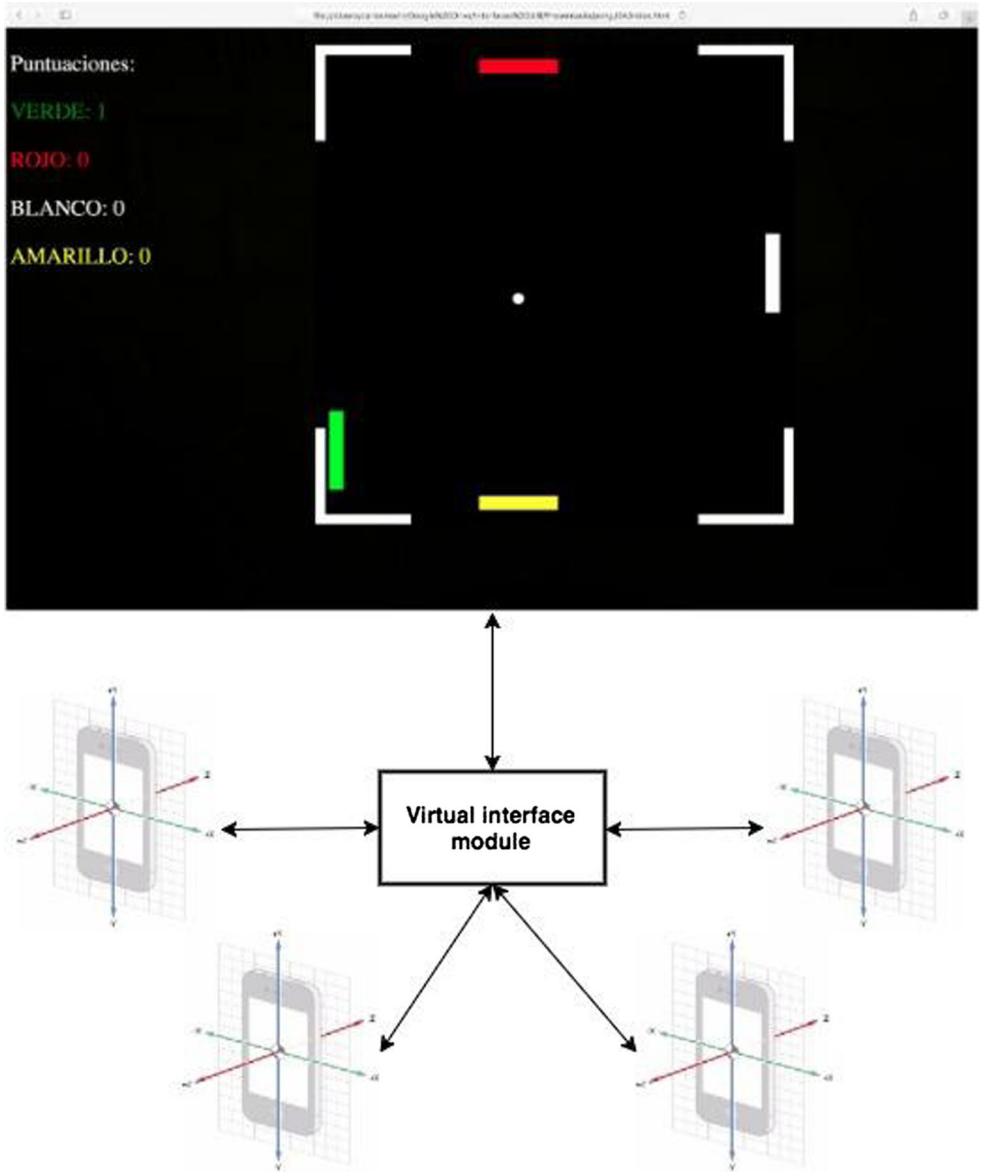

Fig. 15 The connection between the virtual interfaces hosted in the 4 mobile phones and the four-player pong game snapshot on a large screen

is nothing more than some Java Script code. The code reads one of the orientation axes of the mobile phone and sends the information to the virtual interface server. Finally, the virtual interface server sends the parameters of the 4 players to the four-player pong game customer pages. A complicated 4 player game becomes so easy to be implemented and the players enjoyed playing the game together. A screenshot of the game can be seen in Fig. 15.

As shown in Fig. 15, the web pages housed in the mobile phones read from their sensor one of the components of the position sensor and send this value through the web-socket to the virtual interface module. After that, the virtual interface module groups the four read values to the pong game of pong that translates the values to paddle movements.

2.14.5 Fifth example: Pong game with infrared light

This is an illustrative example of using input devices converted into interaction devices. In this case, we use a computer's camera to turn it into an interaction device. We limit the light entering the camera by blocking it using the surface of a used 3.5" floppy disk. This facilitates the processing for computers with little power. We built a pointer with a button and an infrared led on the tip. The acquisition module takes control of the computer's camera and analyzes each image for bright points generated by the infrared pointers in each player's hands. Players move the paddle with the movement of the pencil and the button pressed. The connection diagram is very similar to that of Fig. 4, only one coordinate point corresponding to the **x** of the

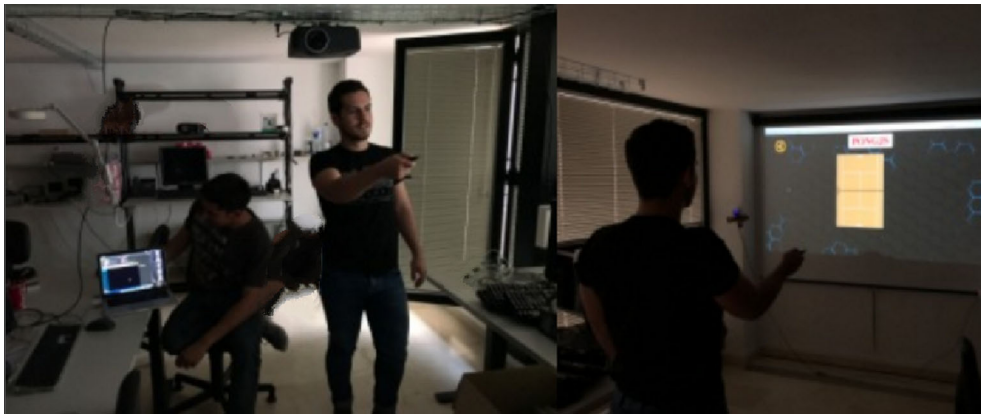

Fig. 16 The pong game with infrared light

detected point is sent. Figure 16 shows the components of the game.

The two images in Fig. 16 taken from the back and front illustrating the components involved in the pong game with infrared light. The computer on the left shows the image that the module receives from the computation of its position. On the right, the player holding the infrared pencil. In the bottom left of the image you can see the webcam that detects the infrared light.

2.14.6 Sixth example: using smart phone to control the mouse and keyboard of a computer

This example is mainly oriented to the field of teaching, with which a teacher can give a class from any point of the classroom using only his smartphone without having to be in front of his computer. However, it is not only tied to that use since the remote use of the computer favors the accessibility of the systems. There are similar applications already available today [6], but we were interested in having control of the system in order to easily adapt it to our framework.

At the time of writing the article, it had not yet been fully integrated into our framework since the server used was running on the client computer and the mobile was connected to it. But, we have proved that it can be transformed into a web client.

It should be noted, however, the great flexibility of having a terminal client (it can also be a service, or a graphical application) that is coupled to the guest operating system. Since it works with node.js and robot.js [9], any OS that supports them is plausible to be fully integrated in our framework and any interface or combination of them can interact directly with the operating system. It currently works with Mac OS X, Windows, Linux, BSD, Unix, ChromeOS. It will be true for other devices such as smartv, etc. This opens the door to the use of interfaces easily elaborated and adapted to the people with special needs at the price of a conventional interaction device (our previous examples already confirmed it). Interfaces can be generated as easily as web pages without a great knowledge of their internal operation.

The current operating mode of this example has been implemented. Using the same WebSocket infrastructure within a local network, the instructor's smartphone (also called VIRTUAL INTERFACE in our system) will connect to the computer you want to control, allowing you to carry out any action you could perform using your mouse or keyboard. Any action performed on the smartphone will communicate with the open service on the computer, which will act accordingly to the message received. See Fig. 17 for how the interaction works.

In the Fig. 17 you can see the process of connecting the virtual interface to the virtual interface server. In the image on the right the user has the screen for the introduction of their credentials. Once validated and authenticated, access to the home screen in the left image is established. In the home screen the user has the possibility to adjust some interaction parameters if the default ones are not to his liking and, most importantly, reset the position of the mobile to the relative position of the projector to maintain an ergonomic position in the interaction. This is done by clicking on the icon of the mobile that is represented in the virtual page.

In Fig. 18. the two images are the two interaction screens with the virtual web interfaces on the computer (it is a web page). In the image on the left, the user interacts with the operating system in a pointer mode. The mobile phone movements are interpreted as the cursor movement for the computer. In the image on the right, the user inputs commands in the mobile phone as if it is a unix terminal. The system also supports control commands ctrl-c, ctrl-z, etc…also for Windows system.

On the other hand, the computer will start the process of listening for connection requests. Once the request is heard the new client will be accepted and will wait for messages. For each message received, this service will receive corre-

Fig. 17 The process of connecting the virtual interface (smart phones) to the virtual interface server

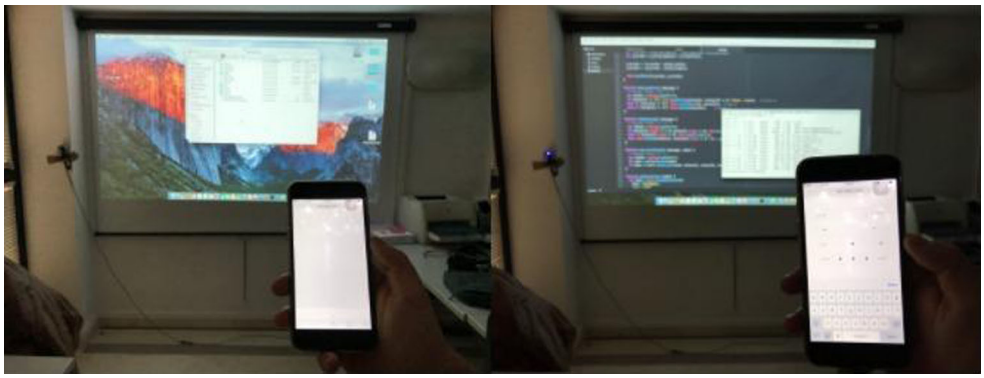

Fig. 18 The two interaction screens with the computer virtual web interfaces

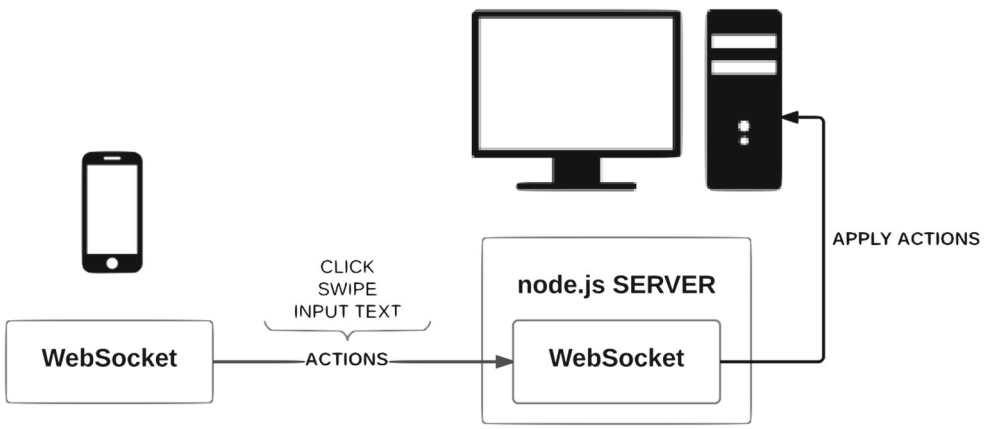

Fig. 19 Interaction between the virtual interface hosted on the mobile phone and the operating system

sponding type of action and execute it, using a library called RobotJS that has facilitated direct interaction with the OS.

Figure 19 shows the interaction between the virtual interface which is hosted on the mobile and the operating system. The virtual interface server and the webmouse system are hosted in the operating system.

2.14.7 Seventh example: the MagiCap

This is an example to illustrate the potential use of this technology in the development of interfaces adapted to people with special needs. We call it the MagiCap. Its aim is to control the mouse pointer by moving the head. For its

Fig. 20 It shows all the components of the MagiCap. At the front, we can observe the ir led, it is covered by a small piece of a red balloon (in order to dim the ir light). On the right, both the battery and the switch have been affixed. A piece of the negative film shown at the top right corner must be adhered to the laptop webcam

implementation we have needed a cap, a 1.5 volts battery, a commutator (for on/off), an infrared led, and a piece of negative film (as light filter). The total amount of the required material was less than €5. In Fig. 20, we illustrate all the components assembled.

It is important to use the negative film for the proper functioning of the system. The film acts as a filter blocking the normal light and letting the ir light pass so that the webcam is able to capture mainly the ir light.

Assembling the components to the cap is very simple we just have to insert the ir led in the front part of the visor and solder two wires which are connected to the battery and the commutator, being these affixed on the right side of the cap, as it is shown in Fig. 20.

Now that the physical part of the system has already been described, we are proceeding to explain the software components.

As in the fifth example, we make use of a webcam as an input device and we use the same software module to detect the brightest point in every image, the pointsdetector. The point detected in every image is sent to the client application by means of a websocket. This module in action can be observed in Fig. 21.

The client application (developed in nodejs with RobotJS), simply connects to the websocket, reads the point sequence, applies a smoothing filter, converts it into screen coordinates and moves the cursor to that particular position.

The magiCap has been tested successfully on different laptops (MacBookPro 2012, Acer Aspire E5) and operating systems such as Windows 10, Mac OS X and Linux.

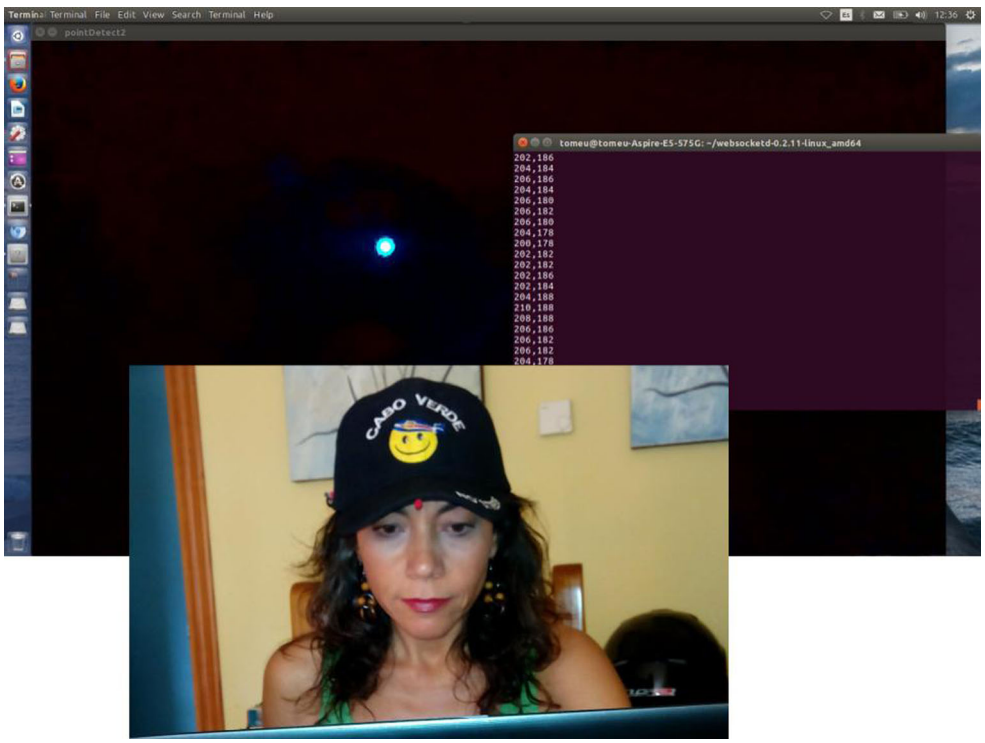

Fig. 21 Shows a screenshot of the system in action. On the right, we can see a terminal printing the detected points. In the background, for illustrative purposes, we see the image detected by the camera and a blue circle marking the detected point. In the foreground, we show the user with the magiCap

Certainly, the precision of the movement depends on the webcam features, such as the image resolution or the ir sensitivity (the better resolution, the better movement precision). Concerning the ir sensitivity, we observed that we are able to adapt the system to the ir sensitivity of the camera removing the piece of the red balloon from the ir led on the MacBook or placing it on the Acer. Anyway, it is always possible to improve the movements precision by reducing the screen resolution.

Although an accurate study of this interface on users hasn't been carried out, the tests on ourselves have allowed us to interact successfully with the already mentioned operating systems.

2.15 Potential applications

In this pilot project, the development has been focused on the use of virtual devices located in a local network to games, but the applications offered by the model can be very diverse. Above all, it gives the idea of how open and scalable the model is, as well as the multimodal use of any type of devices (for example, a simple ring with a red ball, a Wi-Fi camera and a projector connected to the MVCI model can be easily converted into a virtual whiteboard):

- Support for teaching. It can be easily implemented environments supporting the teaching task, such as virtual pointers, interactive whiteboards, collective rapid response tests where each user can, for instance, make use of his mobile (previous registration in the classroom LAN), tests of accessibility, etc.
- Systems accessible to people with special needs. It is also possible to adapt the same application to different individuals, each of them with their own mobility characteristics and their corresponding hardware peripherals. For example, you can play a game of pong using a microphone and two paddles manipulated by two players respectively.
- Data flow control. Flow and presence control sensor environments can be easily assembled. To control the traffic, the noise level, the amount of use of areas in shops, etc.

3 Conclusions and future work

We believe that the framework has achieved its objectives, that is, with few changes, the client web pages are able to handle different interaction devices regardless of the type of information they initially provide.

With our model, web pages do not need to worry about managing different interfaces with their mode of information. We proved this both by playing pong with gestures, potentiometers and virtual interfaces. This works well thanks to the fact of disconnecting the interaction devices from the system and putting them in the cloud. Even they are anchored in a concrete system, through the websockets we can make them transparent to the client applications that can use them. In our scheme, except for the source module, the type of information they provide (video, audio, biosignals, etc.) is not important, it is processed before reaching the client application and adapts to the needs of each client application.

In addition, the architecture works well because of the processing modules which are the ones that actually perform the transformation process of the interaction information. By using an SOA model for implementation and with the help of websockets, the development of new modules and the reuse of them in other schemes is realized in a simple and orderly way. Modules are simple programs that read from stdin and write to stdout in any language available on the node.

With this paradigm, you can open a new development field in the cloud. Just as there are thousands of developers working on web development today, it is also possible to work on web interfaces. Creating an interaction interface becomes a matter of design and imagination. In fact, with the virtual interfaces is already blurred the concept of web page or interface because some are the interaction devices of others. But we must say that the merit is not our framework but the possibilities offered by HTML5 with the use of websockets, access to sensors, WebRTC, WebGL, etc.

With the examples proved, we have seen how easy it is to integrate a new interface into our framework, which, with the development of the webmouse, also allows to interact with almost any operating system. In fact, we are working on the integration of a device that we developed in the past and because of the cumbersome work of developing drivers and adapting it to different operating systems, we had to put it aside [4].

Another important aspect to note is the possibility of relocating the modules according to the calculation power requirements.

In addition to analyzing the functionality of the modules, we can understand the connection schemas as functional programming schemas, where each module represents a function applied to the previous module and the source module is a time function of type f(t). A connection scheme is then of the form h(g(f(t))), so one could probably explore the possibility of programs in cloud.

We believe that a great deal of development work is still necessary in order to adapt this simple prototype to a real application situation. But it is also true that, since the scheme is simple, it is easy for a large community of users to carry solutions that in the beginning had not been tried. Therefore, one of our first objectives in the future work will be to provide

a version available to the community of users of the network in the form of open source.

References

1. opensoundcontrol.org an enabling encoding for media applications. In: opensoundcontrol.org an enabling encoding for media applications. http://opensoundcontrol.org/introduction-osc. Accessed 29 Mar 2017
2. Axenopoulos A, Daras P, Malassiotis S et al. (2012) I-SEARCH: a unified framework for multimodal search and retrieval. The future internet lecture notes in computer science 130–141. https://doi.org/10.1007/978-3-642-30241-1_12
3. Doug Engelbart 1968 Demo. In: Doug Engelbart 1968 Demo. http://web.stanford.edu/dept/SUL/library/extra4/sloan/MouseSite/1968Demo.html. Accessed 29 Mar 2017
4. Estrany B, Fuster P, Garcia A, Luo Y (2008) Human computer interface by EOG tracking. In: Proceedings of the 1st ACM international conference on PErvasive technologies related to assistive environments—PETRA '08. https://doi.org/10.1145/1389586.1389694
5. Etzold J, Brousseau A, Grimm P, Steiner T (2012) Context-aware querying for multimodal search engines. Lecture notes in computer science advances in multimedia modeling 728–739. https://doi.org/10.1007/978-3-642-27355-1_77
6. Keyboard, Mouse and Touchpad. In: Turn iPhone, iPad and Android into wireless mobile mouse/trackpad/keyboard with Remote Mouse. http://www.remotemouse.net/. Accessed 29 Mar 2017
7. Multimodal Interaction Working Group Charter. In: Multimodal interaction working group. https://www.w3.org/2013/10/mmi-charter.html. Accessed 29 Mar 2017
8. Reithinger N, Streit M, Tschernomas V, et al. (2003) SmartKom. In: Proceedings of the 5th international conference on Multimodal interfaces—ICMI '03. https://doi.org/10.1145/958432.958454
9. RobotJS–Node.js desktop automation. In: RobotJS - Node.js desktop automation. http://robotjs.io/. Accessed 29 Mar 2017
10. Spacebrew. In: Spacebrew. http://docs.spacebrew.cc/. Accessed 29 Mar 2017
11. TUIO.org. In: TUIO. http://www.tuio.org/. Accessed 29 Mar 2017
12. W3C. In: W3C multimodal interaction working group. https://www.w3.org/2002/mmi/. Accessed 29 Mar 2017
13. WebSocketsthe UNIX way. In: websocketd. http://websocketd.com/. Accessed 29 Mar 2017
14. OpenCV library. In: OpenCV library. http://opencv.org/. Accessed 29 Mar 2017
15. Arduino–ArduinoYun. In: Arduino–ArduinoYun. https://www.arduino.cc/en/Guide/ArduinoYun. Accessed 29 Mar 2017
16. Arduino.org. In: Arduino open source hardware and software for electronic projects. http://www.arduino.org/. Accessed 29 Mar 2017
17. The Internet of Things with ESP32. In: The Internet of Things with ESP32. http://esp32.net/. Accessed 29 Mar 2017
18. FFmpeg. In: FFmpeg. http://ffmpeg.org/. Accessed 29 Mar 2017