IoT Architecture to Enable Intercommunication Through REST API and UPnP Using IP, ZigBee and Arduino

Hiro Gabriel Cerqueira Ferreira, Edna Dias Canedo, Rafael Timóteo de Sousa Junior
Electrical Engineering Department, University of Brasília – UnB – Campus Darcy Ribeiro – Asa Norte – Brasília – DF, Brazil, 70910-900.
E-mail: hiro.ferreira@gmail.com, ednacanedo@unb.br, desousa@unb.br.

Abstract— **To provide better life quality for mankind through the use of electrical and electronic devices that surround the world, controlling and communication between these equipments must occur and it must happen in an automatic and transparent way. This paper proposes a model based on usual technology to enable the event reading and the controlling of electrical devices, already existent and later developed, through a RESTful API or UPnP technology in order to allow the easy development of applications that can effectively improve life by transparent interaction with regular people. Furthermore, it will be proposed two direct applications of these APIs and it will be shown how to extend it.**

Keywords- IoT communication architecture; Device controlling and eventing enablement; context-awareness middleware; pervasiveness enablement architecture; RESTful API; XML; JSON; Web server; UPnP; Arduino and ZigBee.

I. INTRODUCTION

Electronic devices are essential tools in human being daily life. Mostly, these equipments perform, help or improve everyday tasks such as wake up, take a shower, eat, go to work, have fun, etc. One device, by itself, can already make a difference in our lives, imagine if they were able to talk to each other, exchange information and get pertinent information from the environment? That would bring many new ways to serve humanity. This idea started to become possible with the various computational technologies, such as the Internet, pervasive computing and mobile computing, which emerged in the later decades.

The Internet it self is a great example of the impact that communication between devices brings. This technology made possible the exchange of data between end points in an organized and standardized way trough the use of common protocols. This tool allowed programmers to develop millions of programs around the world that, somehow, use the exchange of data to serve to its very unique purpose and to deliver a service to its final users.

Thus, it arises the idea of creating a network that can take advantage of its connectivity to generate environment's pertinent information in order to better serve the users. Mark Weiser [1] was the responsible for introducing such thought at the beginning of the nineties and some current technologies, like UPnP [2], uses results of his achievement.

Weiser idea's paradigm allowed the emergence of many systems that enable the remote control of electrical devices and the exchange, between equipments, of environment's data in an event based method [2, 3, 4, 5, 6]. They succeeded in their tasks but they didn't go that further. Through this income, it can be questioned: Why hasn't this amazing technology gone around the world?

To answer that, let's take the Internet example. Making use of existing technology such as telegraph, telephone, radio and computer, Internet uses simple and common protocols to make a clear way for programmers to achieve their goals using it. These milestones set the stage for its unprecedented integration of resources [7].

Some systems that lately appeared to enable communication between objects have partially achieved the use of already existing technology. Mostly, these systems left out the use of simple protocols or ways to enable other people to extend or personalize it without great changes.

This paper proposes an Internet of Things architecture, based on UPnP, HTTPS, TCP/IP and other widely known protocol stacks, that solves these conflicts. It comes to allow its usage through two simple methods: a RESTful API and the well-documented UPnP technology.

This paper is organized in sections, for best understanding, as follows. In Section II, is presented an overview of ubiquitous computing and the UPnP architecture. In Section III is presented reviews of some related work about device controlling and eventing. In Section IV is presented the proposed architecture for device controlling, communication and eventing enablement. In Section V is presented two applications of the proposed architecture. Finally, in Section VI, is concluded with a summary of our results and directions for new research.

II. UBUIQUITOUS COMPUTING AND UPNP

Envisioned by Mark Weiser [1], ubiquitous computing, or ubiquity, is composed by mobile computing and pervasive computing and it relies on three main principles: diversity, connectivity and decentralization [8,9]. Figure 1 shows a Venn diagram of Pervasive, mobile and ubiquitous computing relationship.

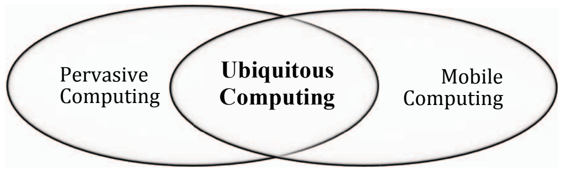

Figure 1. Venn diagram of Pervasive, mobile and ubiquitous computing relationship [8 - Adaptation].

Mobile computing contributes to ubiquity with mobility. It comes to provide independence of geographic position and ubiquitous devices connectivity.

Pervasive computing collaborates to ubiquity with its triviality of use and its context awareness. It allows the ubiquitous devices transparency and calm computing.

The principle of Diversity means that every ubiquitous device has a very unique purpose to attend its users [10]. It assumes that there are plenty devices and that each one is capable of doing a set of tasks, but they are specialized in doing only one.

The principle of Connectivity assumes that ubiquitous devices have high mobility [10]. They may change their position and/or networks and they must still be able to perform its tasks and keep its connections.

The principle of Decentralization brings the independence between devices to function, process its data and do their specific tasks. This principle is the key factor for environment reading, context awareness and pervasiveness.

A. UPnP

It is very hard to achieve ubiquitous computing's demands and propositions without the use of a common protocol. In order to permit the correct data exchange between different devices they must speak the same language, they must use the same protocols.

Universal Plug and Play (UPnP) technology defines architecture for devices interconnection. UPnP enables a network with pervasive characteristics and peer-to-peer (P2P) connectivity [11, 12]. It uses standard Internet protocols [13] such as IP, TCP, UDP, SOAP, XML, HTTP and others as Figure 2 shows.

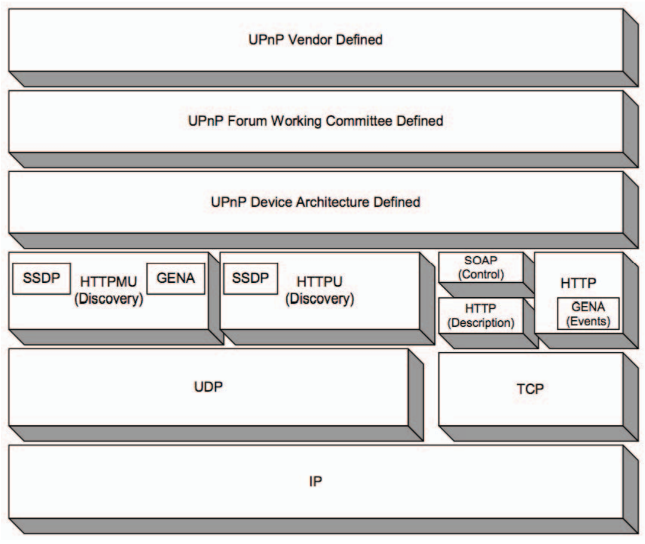

Figure 2. The UPnP Protocol Stack [13]

The basic building blocks of an UPnP network are devices, services and control points [13].

Services are the smaller units of control. They are composed by: a state table, which model its state and update its state variables when needed; A control server, to receive and execute actions requests and return responses updating the state table; An event server, which publishes changes on state variables to subscribers. An example of a service is a light service, which has the state variable power, defining the state of the device light, and two actions, turn-on and turn-off.

Devices are the holder of services and embedded devices. It may have more than one service and more than one device inside of it. Each embedded device can have its own services and its own embedded devices. An example of a device with embedded device is a multifunctional printer. It is one physical root device that contains, at the same time, a embedded printer and am embedded scanner, which have its own and different set of services, actions, control servers, event servers and state variables.

Control Points are entities that can discover and control other devices. They are entities capable of getting devices and services description, invoke actions to control services, and subscribe to service's event source. This controller, embedded to a regular device, allows the construction of pervasive and context aware UPnP network that can exchange pertinent data in order to better serve the users.

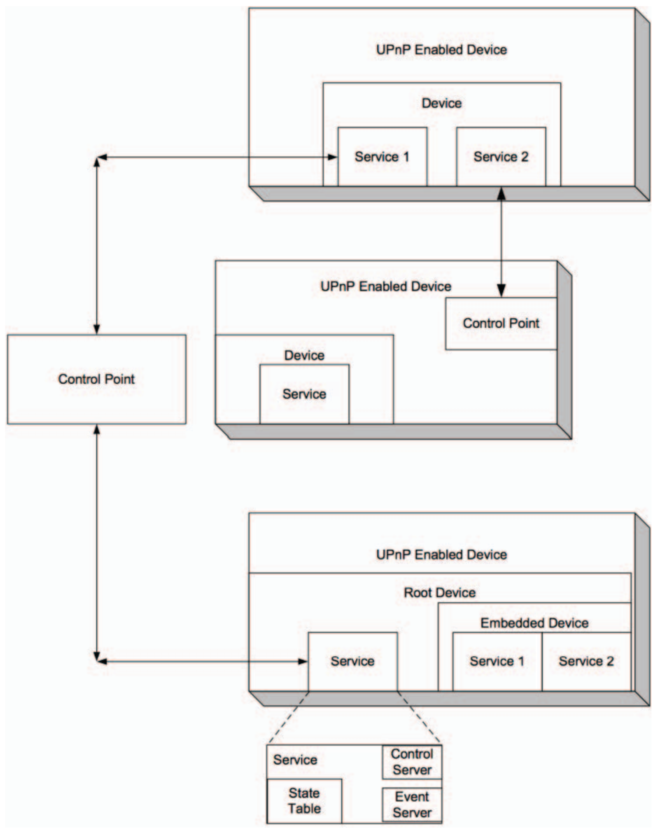

Figure 3. UPnP ControlPoints, Devices and Services[13]

UPnP has huge industry momentum, assuring success and longevity [13] and it's getting popular with DLNA for multimedia exchange [14]. But this protocol stack has the capacity to reach much more than that. This technology enables any sort of device (diversity) to have interconnectivity, enables controlling and processing in a decentralized way, and it permits the interoperability of control points. Everything is supposed to happen on UPnP's six interaction steps:

1. Addressing: Devices obtain an IP;
2. Discovery: Control points get aware of UPnP devices in the network
3. Description: Control points get aware of capacities of UPnP devices in the network through XML exchange.
4. Control: Control points execute actions on UPnP Devices trough service's control server through XML exchange
5. Eventing: Devices notify control points of its state variable changes trough its event server.
6. Presentation: Devices can display a web page for control points and/or users presenting its capabilities and, sometimes, allowing its control.

So it can be reached context awareness through step 2 and 5 using Generic Event Notification Architecture (GENA) and Simple Service Discovery Protocol (SSDP), and it can be reached controlling trough step 3 and 4 using Simple Object Access Protocol (SOAP) and GENA, which, as shown in figure 2, are already part of UPnP [13].

III. Related Work

In the past, researches have been made to enable devices controlling and networking. It can be found the extensive use of ZigBee, Arduino and UPnP technologies to enable pervasive environments and ubiquity.

A. *ZigBee*

The usage of ZigBee to interconnect reachable devices [2,3,5,6] makes the data propagation and protocoling transparent, secure and ensured. ZigBee data exchange standards uses 2.4 GHz radio frequency, the ISM band, has low-power consumption, low-cost, allows customizations for improvement and is largely used in 802.15.4 chipsets [16].

The modular controller implemented in [3] communicate users commands, sent from web applications to end devices, through ZigBee, Ethernet (UPnP) and Power Line. In to [2] it is shown a controller capable of receiving commands from DTMF, SMS or Web application and resend them to end devices using ZigBee.

B. *Arduino*

To translate controller's requests and apply them to devices it can be used Arduino. This simple and complete open-source platform (hardware and software) is intuitively extensible by the attachment of shields, including ZigBee communication (XBee shield), and can receive and send digital and analogic signals [18].

Arduino provides two kinds of pins for transmitting: Pulse Width Modulation (PWM) pins, which are a way to get analog results with digital means, and exclusive digital pins. The two types of receiving pins are onboard 6 channel A/D converter and digital pins [18].

In [15] it is implemented a hand gesture based remote control. It reads infrared (IR) hand gestures, translates then to control actions and execute then on terminal device. Arduino boards do the IR reading and final executions. [17] Implements intercommunication between Arduino boards, using ZigBee. That paper proposes a mutual reading and information transfer between boards, through a mesh network, in order to achieve a common goal: Automated monitoring of home electricity usage.

C. *UPnP*

To abstract controlling and eventing on devices and make it universal, it can be used UPnP. At [19] it is proposed a middleware that allows local and foreign network devices control using ZigBee, UPnP, DCP and IGD concepts and implementations.

IV. Proposed Architecture

According to the review and related research shown and exposed in this paper, an ideal solution for devices intercommunication and controlling must gather an easy way to setup environments with an easy way to extend the platform. It also should use well-known, accepted and documented technologies.

In this section, we introduce a new architecture to enable devices to be controlled via IP, using UPnP or RESTful API. We'll present a layered middleware that can solve controlling and eventing for distributed devices.

A. *Physical Architecture*

In order to be controlled by software, distributed devices must be able to physically communicate with others and, to solve that, 2 types of controllers compose this architecture, the master controller and the slave controllers. Thus, the four intercommunicated entities are: applications, master controller, slave controllers and devices, as figure 4 show.

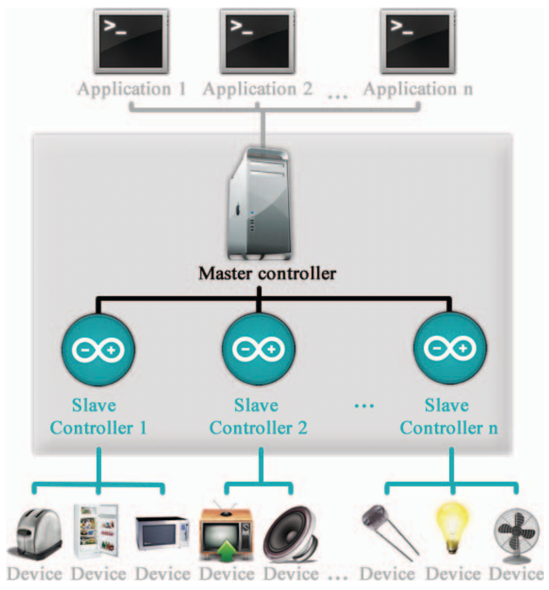

Figure 4. Physical architecture

The master controller is a Linux PC with an UPnP SDK and a LAMP server (**L**inux, **A**pache, **M**ySQL, **P**HP) holding the API. It's responsible for communication with applications and with slave controllers. Once it receives useful data, from applications, it sends reading and writing commands to slave controllers and store data in local database in an UPnP manner. Master controller is the holder of UPnP Control Points and logical UPnP devices with its own services, control servers, event servers and state tables. This controller is responsible for controlling and data store controlling, not for executing actions.

The slave controller is the circuit connected to physical devices that enables them to receive and execute a command. This controller is also able to read state from devices and send it back to master controller. Each slave controller can hold multiple physical devices and it consists of an Arduino board with attached reading and/or writing circuits, specific for each device. This controller is supposed to perform the final action into a device and verify if that action happened, for example: change a light switch to turn it off and check if the light actually turned off.

The reason for multiple slave controllers is the fact that it is not always possible or convenient to have all final devices directly attached to a main controller. Sometimes the geographical localization makes necessary the separation of the controllers. The reason for not every device necessarily has its unique slave controller is to reduce costs.

B. *Layered Architecture*

4 layers compose the logical architecture: service layer, control layer, communication layer and execution layer, as figure 5 shows:

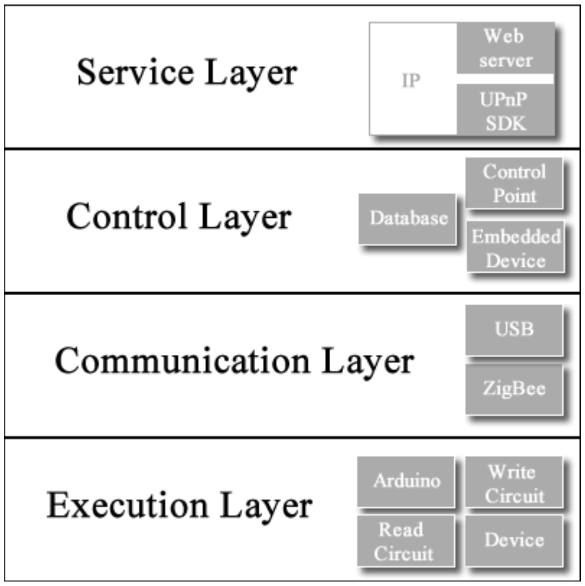

Figure 5. Layered architecture

The service layer is responsible for receiving requests and sending responses to external applications, it's the RESTful API itself and an UPnP SDK. Once it receives and processes a request, service layer sends the action to be taken to the control layer.

The control layer is responsible for keeping state of services and devices, doing multicast notifications to subscribers about states change (eventing) using service layer, discovering available devices around using communication layer, sending actions taking commands to device and sending state reading commands to communication layer.

The communication layer is responsible for exchange data from master controller and slave controllers. It interconnects execution and control layer, it interconnects separated physical devices using ZigBee.

The execution layer is responsible for receiving command from communication layer, send signals to devices and then send response back to control layer, using communication layer again.

C. *Combined Architecture*

As expected, applications run inside the service layer, main controller runs on service, control and communication layer, slave controller runs on communication and execution layer, and devices run on Execution layer. Figure 6 shows the combinations of entities, layers, layers resources and interactions resources.

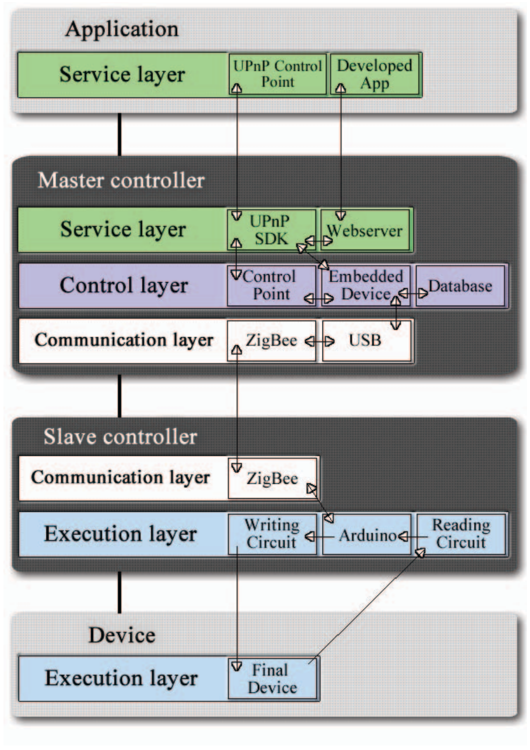

Figure 6. Combined architecture

D. *The API*

The architecture has two Application Programming Interfaces (API).

The first one is through an UPnP SDK that enables controlling and eventing of devices through the regular use of UPnP's six steps and data exchange.

The second one is the REST API. Trough requests of URLs, applications will be able to know the state of devices and to control them. For example, lets say that:

- Master controller has IP 192.168.1.2;
- Slave controller 2 has the name "room";
- Slave controller 2 is able read from, and write to a device named light;
- Device light has a service named 'power_controller'
- Service 'power_controller' has two actions named 'set_power' and 'set_power_after';
- Action 'set_power' receives variable power;
- Action 'set_power_after' receives variables time and power;
- State variable 'power' can receive string "on" or "off", and represents the state to change the light to;
- State variable 'time' receives an integer and represents the interval, in seconds, to wait before changing the light power to Variable power state;

This architecture REST API defines that controlling, invoking an action of some device's service, URL should look like:

- **http:**//*master_controller_ip*/**control**/*slave_controler_name/device_name/service_name/action_name?var1=value1&var2=value2…*

And it defines that to get the value of a state variable of a service the URL should look like:

- **http:**//*master_controller_ip*/**state**/*slave_controler_name/device_name/service_name/state_variable_name*

In our example, to turn on that light in 10 seconds from now, we would only have to access the following URL:

- **http:**//192.168.1.2/**control**/room/light/power_controller/set_power_after?power=on&time=10

And to find out if that light is on or off, we would only have to access the following URL:

- **http:**//192.168.1.2/**state**/room/light/power_controller/power

To get a list, from a master controller, with description of available slave controllers, with its embedded devices and services, or available actions, and state variables, of a service we would, respectively, access URLs as follows:

- **http:**//*master_controller_ip*/**list**/*all*
- **http:**//*master_controller_ip*/**list**/*slave_controller_name/device_name/service_name/all*

All responses from URLs requests will return a lightweight data-interchange formatted string through JSON [20] as shown in table 1.

TABLE I. JSON RESPONSE TABLE

Mode	Response

control	{
	"**status**" : "success"
	}

state	{
	"**status**" : "success",
	"**value**" : "requested state variable value"
	}

list slave_controllers	{ "**status**": "success", "**slave_controllers**": [{ "**name**" : "slave controller 0 unique name", "**description**":"slave controller 0 description", "**embedded_devices**" : [{ "**name**" : "device name", "**description**" : "device description", "**services**" : ["service 1 name","…"] }] }] }

list
service
	{
	"**status**": "success",
	"**description**": "description of the service",
	"**state_variables**": [
	{
	"**name**" : "state variable name",
	"**description**" : "state variable description",
	"**type**" : "variable type (string, number…)",
	"**minimum_value***" : "minimum allowed integer, float, number...",
	"**maximum_value***" : "maximum allowed integer, float, number...",
	"**step_size***" : "difference between consecultive values, increment value",
	"**allowed_values***" : ["value1","value2..."]
	}
	],
	"**actions**": [
	{
	"**name**" : "action name",
	"**description**" : "action description",
	"**variables**" : ["state variable n name","…"]
	}] }

a. Keys finished with * are optional and may come with null value depending on variable type.

JSON was chosen because it can automatically be converted to an array or object of many programming language [20], making interaction with this REST API even easier. Another reason is because it's lightweight and human readable. In conjunction with the description key value of controllers, devices, services, actions and state variables, it's intended to turn the API more human understandable and readable.

E. Adding new devices

Now that the architecture's operation for end applications was fully explained, it will be presented how slave controllers and end device are appended to the middleware.

To add a new slave controller, we just have to compile and load the default architecture's program code for slave controllers to the Arduino board, using regular Arduino Software available [18]. Before uploading the code to the board we must set the variable "xbee_network_id" to a free value. To get that free value, visit: http://*master_controller_ip*/zigbee_id **.**

A slave controller must be an Arduino extended with an XBee RF module (ZigBee PRO feature set) in order to communicate with master controller.

To setup a device, it has to be connected to the Arduino board. To do that, it's used writing and/or reading circuits. In an abstract way, writing and reading circuits must be able to receive and send serial data to Arduino board, respectively. So both circuits share Arduino's Ground and Vcc values and have a signal pin reserved for the serial communication, as figure 7 shows.

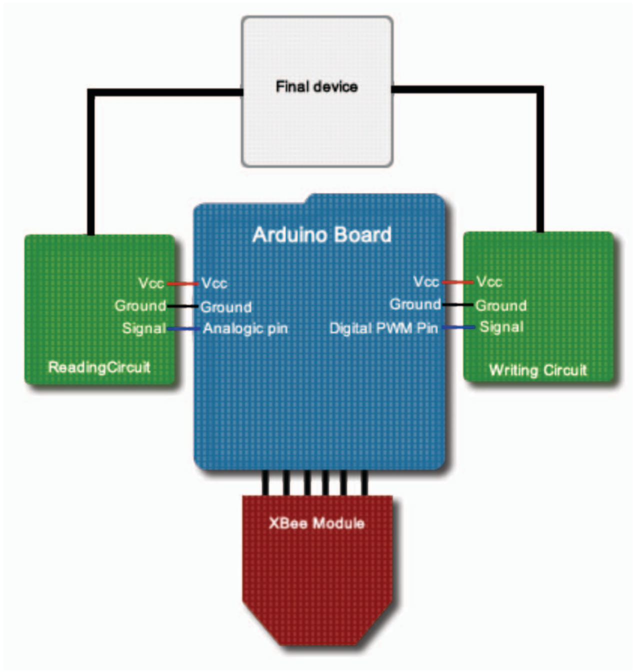

Figure 7. Example slave controller

When allowed by Arduino board, to read signals, it can be used pin's half-duplex capabilities to first request what to read. Ideally it should be used writing circuit signal pin to write to device and immediately listen to the reading circuit to obtain an answer, making the communication full duplex, but in a more delayed action that might not be possible.

Once the physical structure is all ready, it's necessary to configure de logical device. To do that, the middleware provides a Graphical User Interface (GUI) through its web server that can be accessed at:

- **http:**//*master_controller_ip*/**configure**/*slave_controller_xbee_network_id*

This GUI allows filling up html forms to **C**reate, **R**ead, **U**pdate and **D**elete (CRUD) devices, its services, actions and state variables. The correlation of slave controllers and devices, devices and services, services and state variables, services and actions, and actions and state variables are all 1:n. Table 2 below shows all fields that are gathered in GUI's CRUD.

TABLE II. GUI ENTITIES FORMS FIELDS

Entity	Field
Slave controller	Name
	Description
Slave controller > Device	Name
	Description
	Writing circuit baud rate
	Reading circuit baud rate
Slave controller > Device >Services	Name
	Description
Slave controller > Device > Service > State Variables	Name
	Description
	Type
	Minimun allowed value
	Maximum allowed value
	Step size
	Allowed values
	Reading circuit signal message
	Reading circuit pin
	Writing circuit signal message
	Writing circuit pin
	Writing circuit response length
Slave controller > Device > Service > Actions	Name
	Description

b. All writing circuit messages must start with ASCII printable character one (integer 49).

Filling in the forms will allow applications to perform actions described in table 1 and receive those responses based on the information filled in table 2.

An action, sent from Arduino to a writing circuit, is a serial transmission of a State Variable's Value (SVV) in a digital or analogic way.

F. Rules for writing and reading circuits

Action's state variables are sent at the same time when using different pins by parallel transmission. In case two or more state variables share the same digital pin, they will be transmitted in a row, one after the other, without any separation. In case two or more state variables share the same PWM pin, they will be transmitted in a row, one after the other, separated by ASCII new line char (integer 10).

Arduino's PWM pins accept values from 0 to 255. This value means the percentage, ratio between the value and 256, in a duty cycle (2 milliseconds), that the wave will be in high voltage (5v). For example, writing ASCII char "**w**" to a PWM pin is the same as writing value 127, which represents 50%, from 0 to 255. So it will be written a wave that stays half duty cycle in high voltage (5v) and half duty cycle in low voltage (0v). Due to its speed, PWM simulates voltages between 0v and 5v, also proportional to 0 to 255, and in case of char "**w**" it's represented as 2,5v constant signal to analogic devices.

Any transmitted value, in PWM or regular digital (D) pin, will stay written to the pin until a new value is written to that very pin. It's important to emphasize that an analog char is repeated for, at least one baud slot length. So, every char is repeated the ratio between baud slot and duty cycle. That forms, for analogic devices, equivalent to constant waves between 0v to 5v.

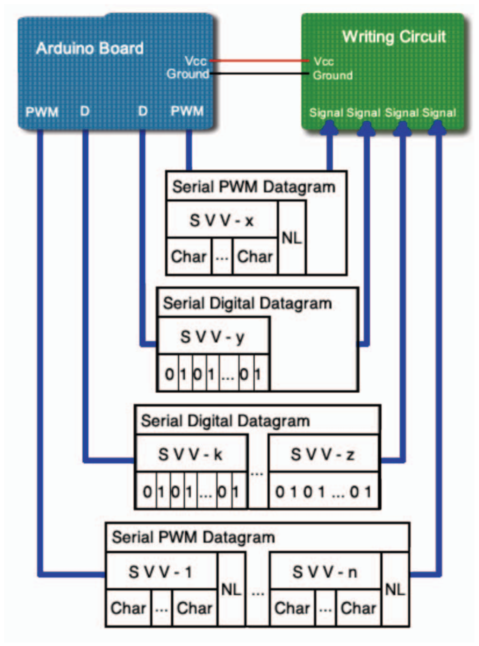

Figure 8. Serial datagrams

In the same way it should be written, by the reading circuit to the Arduino board, the state variable's value read on every baud slot. If "writing circuit signal message" isn't empty, Arduino will first write it to the pin in one baud slot, then read the value on consecutive baud slots. To do that in the right way, these values have to obey length, ranges and baud rate inserted in form fields of table 2 at GUI. Empty "writing circuit signal message" means that a pin is used exclusively for reading one and only one state variable, what is useful for common use sensors and state variables.

V. EXAMPLE USAGE SCENARIOS

With the proposed architecture we can easily develop various end applications using the vast knowledge and circuits of already existing cheap Arduino sensors. These ready to go options encompasses measurers of air temperature, air humidity, soil humidity, pressure, distance, light, smoke gas, human body touch, and others sensors such as RFID reader, flame fire detector, key switch sensor, infrared obstacle avoidance sensor, gyroscope sensor, accelerometer sensor, alcohol sensor, sound detector, magnetic detector, tilt sensor, speed detector, GPS localization, metal proximity sensor, motion detector, vibration detector, light color identification, RF link receiver, etc.

Most of above sensors need three pins: Vcc, ground and analog signal pin. So, they are ideal reading circuit for common use on context awareness.

For writing circuits, we can use 3 generic controlling approaches that handle most usual devices. The first one is the switch circuit that is used to control power only based devices, such as lights and toasters, and devices with embedded controlling knobs, buttons or lever handles. The second one is the infrared circuit that is capable of sending infrared waves and can be used to control most devices that have a remote controller such as televisions, stereos and air-conditioned. The last one is the radio frequency circuit that can handle devices with remote control that are not infra red based such as electric gates, electric shades and most long distance remote controllable machines and devices. Figures 9, 10 and 11 show sketches of these circuits.

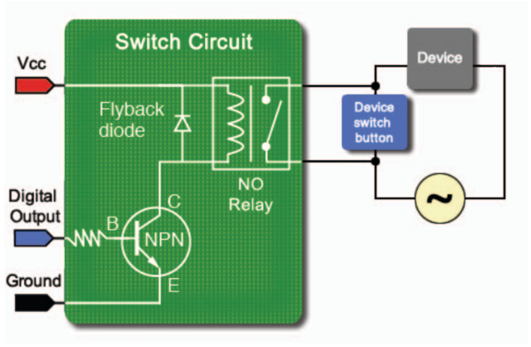

Figure 9. Writing circuit - Switch

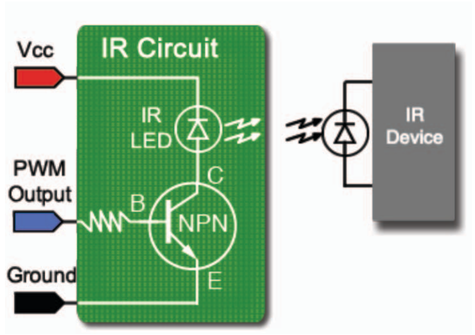

Figure 10. Writing Circuit – Infra Red

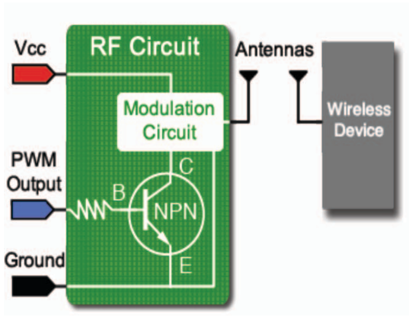

Figure 11. Writing Circuit - Radio Frequency

For home automation scenario, it could be used a slave controller for each room. These slave controller would have switch circuits to send actions to room lights and ceiling fans, IR circuits for televisions, sound systems and air-conditioned, and RF circuits for electric shades, electric gates and so on. To read state of air-conditioned, it could be used, as reading circuits, a temperature measurer with a voltage sensor. To get state of lights, it could be used either a voltage sensor or a light sensor. To read state of electric gates and shades, it could be used distance and motion sensors. To read state of televisions and sound systems, it could be used light color identification sensors.

For industry scenario, it could be used the writing switch circuit into machines knobs and lever handles and custom or default reading circuits specific for the action.

For context-awareness reading, it could be distributed many slave-controllers around ambient with only reading circuits to provide local data such as temperature, gas quantity per volume, humidity, sound noise and geolocation. All data would propagates through ZigBee to reach master controller and be accessible through centralized API.

VI. Conclusions

Software device controlling and intercommunication between devices have been focus of several recent researches, which demonstrate the importance and need of a common, simple, complete, expandable and unified architecture.

This paper came to provide that. The vast reading and writing circuits allow encompassing about every scenario and it also allows easy creation of custom circuits to extend it. Integration between end devices and end applications is transparent and uses well-known protocols and technologies.

Future improvements to this proposed architecture could solve security issues by adding SSL/TLS on service layer, authentication on control layer, AES 128 bit on ZigBee in communication layer, and slave controller secret phrase for master controllers at execution layer. Furthermore, it could be suggested more examples and detailed usage scenarios configurations, add more service interface, writing circuits and reading circuits to enable controlling by and to devices that can already communicate in other technologies such as power line and DTMF.

References

[1] Want, R., "Remembering Mark Weiser: Chief Technologist, Xerox Parc," *Personal Communications, IEEE* , vol.7, no.1, pp.8,10, Feb 2000 doi: 10.1109/MPC.2000.824564

[2] Zhenxing Wang; Zhongyuan Liu; Linxiang Shi, "The smart home controller based on zigbee," Mechanical and Electronics Engineering (ICMEE), 2010 2nd International Conference on , vol.2, no., pp.V2-300,V2-302, 1-3 Aug. 2010 doi: 10.1109/ICMEE.2010.5558422

[3] Bjelica, M.Z.; Teslic, N., "A concept and implementation of the embeddable home controller," *MIPRO, 2010 Proceedings of the 33rd International Convention* , vol., no., pp.686,690, 24-28 May 2010

[4] Erdem, H.; Uner, A., "A multi-channel remote controller for home and office appliances," Consumer Electronics, IEEE Transactions on , vol.55, no.4, pp.2184,2189, November 2009 doi: 10.1109/TCE.2009.5373786

[5] Taewan Kim; Hakjoon Lee; Yunmo Chung, "Advanced universal remote controller for home automation and security," Consumer Electronics, IEEE Transactions on , vol.56, no.4, pp.2537,2542, November 2010 doi: 10.1109/TCE.2010.5681138

[6] Il-kyu Hwang; Dae-sung Lee; Jin-wook Baek, "Home network configuring scheme for all electric appliances using ZigBee-based integrated remote controller," Consumer Electronics, IEEE Transactions on , vol.55, no.3, pp.1300,1307, August 2009 doi: 10.1109/TCE.2009.5277992

[7] Kleinrock, L., "An early history of the internet [History of Communications]," *Communications Magazine, IEEE* , vol.48, no.8, pp.26,36, August 2010

[8] Araujo, R. B., "Computação Ubíqua: Princípios, Tecnologias e Desafios". In: XXI Simpósio Brasileiro de Redes de Computadores, 2003, Natal, CE. SBRC, 2003. p 45-115.

[9] Hansmann, U.; Merk, L., Nicklous, M.S.; Stober, T. (2001) "Pervasive Computing Handbook", Ed. Springer. 409 pags

[10] Weiser, M. (2003). The Computer for the 21st Century. Scientific American., pp 94-104;

[11] Universal Plug and Play Device Architecture, v. 1.1, October 2008. Avaliable at: <http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf> acessed may 27, 2013.

[12] Miller, B.A.; Nixon, T.; Tai, C.; Wood, M.D.; , "Home networking with Universal Plug and Play,"*Communications Magazine, IEEE* , vol.39, no.12, pp.104-109, Dec 2001, doi: 10.1109/35.968819

[13] Universal Plug and Play Forum, Understanding Universal Plug and Play, 2000. Avaliable at: http://www.upnp.org/download/UPNP_understandingUPNP.doc accessed may 27, 2013.

[14] Digital Living Network Allience, DLNA, 2013. Avaliable at: < http://www.dlna.org/> accessed may 28, 2013.

[15] Solanki, U.V.; Desai, N.H., "Hand gesture based remote control for home appliances: Handmote," *Information and Communication Technologies (WICT), 2011 World Congress on* , vol., no., pp.419,423, 11-14 Dec. 2011

[16] ZigBEE Alliance, 2013. Avaliable at: http://www.zigbee.org/ accessed june 6, 2013.

[17] Anbya, M.F.B.; Salehuddin, M.; Hadisupadmo, S.; Leksono, E., "Wireless sensor network for single phase electricity monitoring system via Zigbee protocol," *Control, Systems & Industrial Informatics (ICCSII), 2012 IEEE Conference on* , vol., no., pp.261,266, 23-26 Sept. 2012

[18] Arduino, 2013. Avaliable at : <http://www.arduino.cc/> accessed at jun 7, 2013.

[19] Kim, Kuk-Se; Chanmo Park; Kyung-Sik Seo; Il-Yong Chung; Joon Lee, "ZigBee and The UPnP Expansion for Home Network Electrical Appliance Control on the Internet," *Advanced Communication Technology, The 9th International Conference on* , vol.3, no., pp.1857,1860, 12-14 Feb. 2007

[20] JSON. Avaliable at http://www.json.org, accessed june 22, 2013.