A ubiquitous communication architecture integrating transparent UPnP and REST APIs

Hiro Gabriel Cerqueira Ferreira*, Edna Dias Canedo and Rafael Timóteo de Sousa Junior

Electrical Engineering Department,
University of Brasília,
UnB, Campus Darcy Ribeiro, Asa Norte,
Brasília, DF, 70.910-900, Brazil
E-mail: hiro.ferreira@gmail.com
E-mail: ednacanedo@unb.br
E-mail: desousa@unb.br
*Corresponding author

Abstract: This paper proposes a ubiquitous middleware architecture enabling end devices to be easily used through software services. These services are available to end applications in a transparent and extensible manner using UPnP or a REST API. Internal modules are interconnected by ZigBee, which allows a mesh and auto reconfigurable internal network. Arduino is used to interconnect end devices to the middleware, allowing easy integration of devices by reading their state and executing actions on them.

Keywords: pervasiveness enablement architecture; context-aware middleware; event monitoring; device controlling; representational state transfer; REST; universal plug and play; UPnP.

Reference to this paper should be made as follows: Ferreira, H.G.C., Canedo, E.D. and de Sousa Jr., R.T. (2014) 'A ubiquitous communication architecture integrating transparent UPnP and REST APIs', *Int. J. Embedded Systems*, Vol. 6, Nos. 2/3, pp.188–197.

Biographical notes: Hiro Gabriel Cerqueira Ferreira graduated in Communication Networks Engineering from Universidade de Brasília, Distrito Federal, Brazil in 2012. Since the second half of 2012, he is a Master's degree student in Electrical Engineering at Universidade de Brasília with major domain in telecommunication and specific area on ubiquitous computing. He also is co-founder of Green Tecnologia, Brazilian Enterprise of Software Development.

Edna Dias Canedo graduated in Systems Analysis from Universidade Salgado de Oliveira, Goiás in 1999. She received her Masters degree in the Domain of Software Engineering from Universidade Federal da Paraíba UFPB in 2002, and doctorate degree in Telecommunications from Universidade de Brasília (UNB). She is a Professor of the Software Engineer course in the Gama Institute of the Universidade de Brasília – UNB.

Rafael Timóteo de Sousa Jr. graduated in Electrical Engineering from Universidade Federal da Paraíba, Campina Grande, Brazil in 1984. He received his Masters degree (DEA) in Telematics and Information Systems from École Supérieure d'Electricité – SUPELEC, France in 1985) and doctorate degree in Signal Processing and Telecommunications from Université de Rennes I, France in 1988. He performed his sabbatical year research on computational trust in ad hoc networks at Ecole Supérieure d'Electricité – SUPELEC (2006–2007). He is an Associate Professor in the domains of Computer Networks Engineering, Information Technology and Information Security at the Universidade de Brasília.

1 Introduction

Electronic devices are essential tools in human beings' daily life. Indeed, these standalone devices perform, help or improve everyday tasks. Thus, it is interesting to imagine a scenario in which devices become 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 computational technologies that emerged in recent decades such as the internet, pervasive computing and mobile computing.

A great example of the impact arising from communicating devices is the internet. This technology made possible the exchange of data between end points in an organised and standardised way through the use of common protocols. This tool allowed the development of millions of programmes around the world using the exchange of data for delivering services to final users.

Thus, a consequent evolution was to the idea of creating a network that can take advantage of its connectivity to gather and process pertinent information from the environment in order to better serve users. Mark Weiser (Want, 2000) was the responsible for introducing such thought at the beginning of the nineties and some current technologies, like universal plug and play (UPnP) (Wang et al., 2010), uses results of his achievement.

Weiser's paradigm allowed the emergence of many systems that enable the remote control of electrical devices and allow the equipment to exchange environment data in an event-based method (Wang et al., 2010; Bjelica and Teslic, 2010; Erdem and Uner, 2009; Kim et al., 2010; Hwang et al., 2009). Most of these works use proprietary protocols and exclusive methods to control devices. They do not allow expansion and have no APIs. That turns them into black boxes that cannot be improved nor expanded by others but their authors.

This paper is motivated by the principle of proposing an open architecture that uses already known protocols, technologies and common open APIs in order to allow more transparent interactions, improvements and easy expansion by anyone.

This paper proposal follows the internet example, which used existing technology, such as telegraph, telephone, radio and existing computes, in order to put forward a simpler and clear way for programmers to achieve their goals using it. Internet milestones set the stage for its unprecedented integration of resources (Kleinrock, 2010) that allowed new study and continuous latter improvements.

Consequently, this paper proposal is aimed at reuniting already known technological resources to provide a new architecture that allows simple expansion and interaction between programmes and devices. Once a device is integrated within the architecture through Arduino codes and circuits, it can be controlled accessing URLs or sending XML messages, enabling communications independent of operational system, programming language or hardware.

This work is organised in sections, as follows. In Section 2 is presented an overview of ubiquitous computing and the UPnP architecture. Section 3 reviews related work about device controlling and event processing. Section 4 describes the proposed architecture for device controlling, ubiquitous enablement and event processing. Section 5 presents system prototypes based on the proposed architecture. Section 6 discusses potential applications of the proposed architecture and prototypes. Finally, in Sections 7, the conclusion is presented with a summary of our results and directions for new research.

2 Ubiquitous computing and UPnP

To better describe the proposed architecture, it is important to review some topics about ubiquitous computing and UPnP. As envisioned by Mark Weiser (Want, 2000), ubiquitous computing, or ubiquity, is composed by mobile and pervasive computing and is founded on three main principles: diversity, connectivity and decentralisation (Araujo, 2003; Hansmann et al., 2001).

Mobile computing is related to the maintenance of connectivity regardless of geographical position. Pervasive computing comprehends triviality of use and context awareness. It allows the ubiquitous devices to proceed with transparency and calm computing.

The principle of diversity means that every ubiquitous device has a very unique purpose to attend its users (Weiser, 1991). It assumes that there are plenty devices and that each one is capable of doing a set of tasks, but they are specialised in doing only one.

The principle of connectivity assumes that ubiquitous devices have high mobility (Weiser, 1991). They may change their position and/or access networks and they must still be able to perform their tasks and keep their connections.

The principle of decentralisation concerns the independence between devices to function, process their data and do their specific tasks. This principle is the key factor for environment monitoring, context awareness and pervasiveness.

To achieve a middleware with ubiquitous characteristics, it has to cover all of ubiquity principles. That requests a common protocol to allow the correct communications between all involved parts.

TCP/IP control and management protocols, namely ICMP and SNMP, are scope limited protocols if considered for device controlling in a ubiquitous manner. These technologies focus on monitoring communication and computational characteristics of devices, but not on their available tasks and services.

UPnP defines architecture for devices interconnection that enables a network with pervasive characteristics and peer-to-peer (P2P) connectivity (Universal Plug and Play Device Architecture, 2008; Miller et al., 2001). It uses standard internet protocols (Universal Plug and Play Forum, 2000) such as IP, TCP, UDP, SOAP, SSDP, GENA XML and HTTP, to allow transparent communication between its parts. Using the UPnP stack, any entity capable of reading from and writing XML messages to another entity is ready to control, be controlled, update its status for its subscribes on the network and to be updated by other entities about their status when being their subscriber.

The basic building blocks of an UPnP network are devices, services and control points (Universal Plug and Play Forum, 2000). Devices are holders of services and contain embedded devices. A device may have more than one service and more than one internal device. Each embedded device can have its own services and its own embedded devices.

Services are units of control composed by a state table, a control server and an event server. An example of a service is a lighting service, which has the state variable power (defining the state of a light device) and two actions, turn-on and turn-off.

Control points are entities that can discover and control other devices. They are capable of getting devices and

services descriptions, invoke actions to control services, and subscribe to service's event sources. While embedded to regular devices, this kind of controller allows the construction of pervasive and context aware UPnP networks that can exchange pertinent data in order to better serve their purpose.

Within these networks, UPnP operations include 6 main steps, as follows:

1 addressing: devices obtain IP addresses

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 exchanges

4 control: control points execute actions on UPnP devices by exchanging XML messages with their services control server

5 eventing: devices notify their state variable changes to control points event servers

6 presentation: devices can display a web page for control points and/or to users, thus presenting their capabilities and, sometimes, allowing their control.

These operations support context awareness through steps 2 and 5, by using the generic event notification architecture – GENA, and the simple service discovery protocol – SSDP. Controlling operations comprise steps 3 and 4, using the simple object access protocol – SOAP, and GENA.

UPnP has huge industry momentum (Universal Plug and Play Forum, 2000) and is becoming popular with DLNA certified products for multimedia exchange (DLNA, 2013). As described above this technology enables any sort of device to have interconnectivity (diversity). It enables device controlling and service processing in a decentralised way, and it permits the interoperability among control points. Besides, UPnP technology includes the concept of discovery and description of devices and services, thus allowing a device to be dynamically found and fully understood, in functionality, by XML descriptions. These characteristics motivate the choice of the UPnP protocol stack for supporting the ubiquitous architecture proposed in this paper.

3 Related work

Various research works have been made regarding the questions of devices controlling and networking. In general, it is common to these works the extensive usage of ZigBee, Arduino and UPnP technologies to enable pervasive environments and ubiquity.

3.1 ZigBee

The usage of ZigBee to interconnect reachable devices (Wang et al., 2010; Bjelica and Teslic, 2010; Kim et al., 2010; Hwang et al., 2009) makes the data propagation and protocoling transparent, secure and ensured. ZigBee data exchange standards uses 2.4 GHz radio frequency (ISM band), has low-power consumption, low-cost, allows customisations for improvement and is largely used in 802.15.4 chipsets (ZigBEE Alliance, 2013). Besides that, it can be easily configured to generate mesh and auto reconfigurable networks, something that other wireless technologies usually do not.

For instance, the modular controller implemented in Bjelica and Teslic (2010) communicate users commands, sent from web applications to end devices, through ZigBee, Ethernet (UPnP) as well over power lines. Wang et al. (2010) present a controller capable of receiving commands from DTMF, SMS or web applications and then resending these commands to end devices using ZigBee. But, these proposals are based on specific intercommunication protocols, which makes then difficult to be extended and fully studied so as to be further improved. A contribution of our proposal consists of designing an architecture that is open to be extended and improved due the choice of its standard protocols.

3.2 Arduino

Arduino (2013) is a simple though complete open-source platform comprising hardware and software that is used in our proposal to translate controller requests and apply them to devices. This platform is intuitively extensible by the attachment of shields, including one for ZigBee communications (XBee shield). Arduino can receive and send digital and analogic signals using PWM, digital pins and onboard channel A/D converters.

For instance, Arduino is used in Solanki and Desai (2011) to implement a hand gesture-based remote control that uses infrared (IR) signals to monitor hand gestures, translating these gestures to control actions and execute them on a terminal device. Another example (Anbya et al., 2012), based on the utilisation of ZigBee for intercommunication between Arduino boards, proposes a mutual reading and information transfer between boards, through a mesh network, in order to achieve the common goal of automated monitoring of home electricity usage.

Again, these proposals are not meant to be expanded or reused for other purposes.

3.3 UPnP

As described above, UPnP is used to abstract controlling and eventing on devices and make these operations openly available.

Kim et al. (2007) propose a middleware that allows local and external control on network devices using ZigBee, UPnP, DCP and IGD concepts and implementations. This proposed middleware aims to control already existent devices, but this paper (Kim et al., 2007) does not clarify how to add devices and which operations can be performed beyond communications. Another contribution of our

proposal is a layered architecture that clarifies functions for device and services discovery, utilisation and extension.

3.4 REST

In order to offer a transparent API, the same controlling and eventing available throughout UPnP should be usable through a distributed computing programming interface based on the representational state transfer – REST (Stirbu, 2008). This kind of API can be exercised via clients such as cURL (Stenberg, 2013) or standard libraries for generating HTTP requests in many programming languages. That allows even older programming languages to access devices without sending and receiving XMLs or specific protocol messages beyond HTTP GET. REST APIs are based on the idea that the access to a URL through a web browser can be bound to an action to be executed or a state to be retrieved. Any software accessing that URL would trigger the same event.

Gupta et al. (2010) present a sensor network that sends and receives data using REST and turning this data viewable into a dynamic web application.

4 The proposed architecture

As discussed before an ideal solution for ubiquity and device controlling must gather an easy way to setup environments as well as means to extend the platform capabilities. To attain this goal, the middleware should use well-known, accepted and documented technologies in order to allow its improvement and understanding by other people.

In this section, we introduce our proposed architecture extending a previous work (Ferreira et al., 2013). This architecture enables devices to be controlled via IP, using both UPnP and a RESTful API. It consists of a layered middleware that can support controlling and eventing for distributed devices through ZigBee and Arduino. Its main components are illustrated in Figure 1.

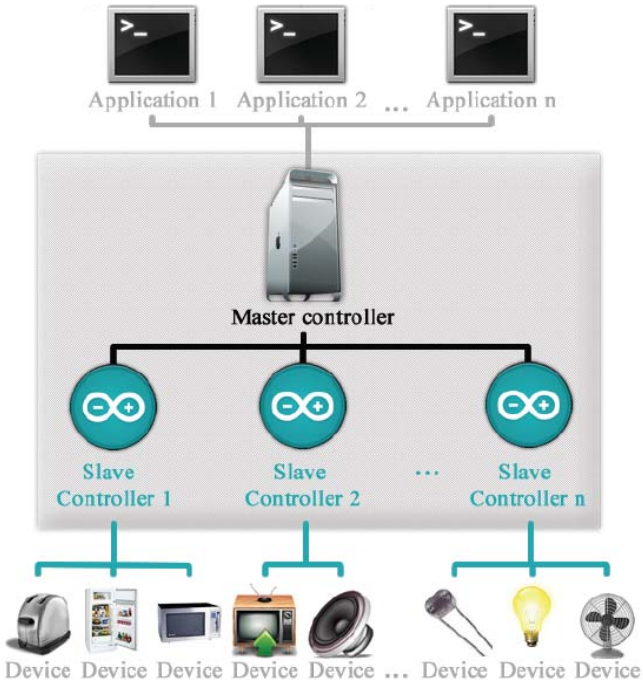
4.1 Physical architecture

The idea is to interconnect end devices with applications through a middleware. Since the devices are distributed in space and the intention is to unify communications for applications and these devices, the middleware is based on a main controller that interacts with applications while slave controllers interoperate with nearby devices.

The applications and the master controller exchange data using a REST API or UPnP, both over a TCP/IP stack. The master controller and the slave controllers exchange data through ZigBee mesh networks using serial data transmissions. Slave controllers and end devices exchange data using Arduino and, when needed, specifically modelled reading and writing circuits for some device.

The master controller contains the entire middleware database and is the ZigBee coordinator. Thus, the master controller keeps state data regarding all devices controlled by slave controllers. It also sends control requests to slave controllers and treats their responses according to the sent request. Consequently, the master controller is responsible for broadcasting and managing all UPnP protocol stack rules, packages and for providing the REST API.

Figure 1 Physical architecture (see online version for colours)



Source: Ferreira et al. (2013)

A slave controller is a circuit connected to physical devices enabling them to receive and execute commands. It communicates through ZigBee with master controllers. Such a controller is also able to read state data from devices and send this data to the master controller. Each slave controller 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 succeeded. Support to more complex end devices can be assured if slave controllers use a specific system such as ARTK (Schimpf, 2013).

5 Proposed services and devices integration

Aimed at illustrating the organisation of the proposed architecture, this section presents seven simplified services that can be generated using this ubiquitous architecture. All necessary state variables, actions, circuits and device configuration are presented in text and tables, referencing the diagram of components displayed in Figure 2.

Tables 1 to 7 contain the necessary abstraction of services in a device and corresponding state variable. It also specifies reading and writing circuits that manage the presented state variable and the values that this variable can assume.

The reading circuit of a device state variable will only return the allowed values. Thus, writing circuits will only be

able to set state variables to their allowed values. A table line without any specified reading or writing circuits means that the respective state variable is read only or write only.

5.1 Services description

Table 1 represents the first application service which is a humidity monitoring system that requires humidity sensors distributed over desired perimeters.

Table 1 Humidity monitor system

Device name	*State variable (SV)*	*SV allowed values*	*Reading circuit (RC)*	*Writing circuit (WC)*
Hygrometer	Humidity	0 to 1,023	Humidity sensor	-

The second service (Table 2) is an irrigation system that requires a switch circuit and voltage sensors attached to every water valve to respectively change and read the state of the valves (on/off). Soil humidity sensor can be used to monitor when to turn the valves on and off.

The third service is a temperature system that requires a thermometer to read the ambient temperature, and an IR emitter LED to send commands to air conditioners and heaters in order to change the temperature (Table 3).

Table 2 Irrigation system

Device name	*State variable (SV)*	*SV allowed values*	*Reading circuit (RC)*	*Writing circuit (WC)*
Soil hygrometer	Humidity	0 to 1,023	Soil humidity sensor	-
Water sprinkler	Power	On, off	Voltage sensor	Switch circuit

Table 3 Temperature system

Device name	*State variable (SV)*	*SV allowed values*	*Reading circuit (RC)*	*Writing circuit (WC)*
Air conditioner and heater	Temperature	15 to 30	Air temperature measurer	IR circuit
	Power	On, off	-	IR circuit
	Mode	Heater, cooling	-	IR circuit
Thermometer	Temperature	–55 to 125	Air temperature measurer	-

Table 4 represents the fourth service which is a lighting system where a switch circuit turns lights on and off and a luminosity sensor can check if the light is turned on or off.

Figure 2 Services, circuits, models and average prices (see online version for colours)

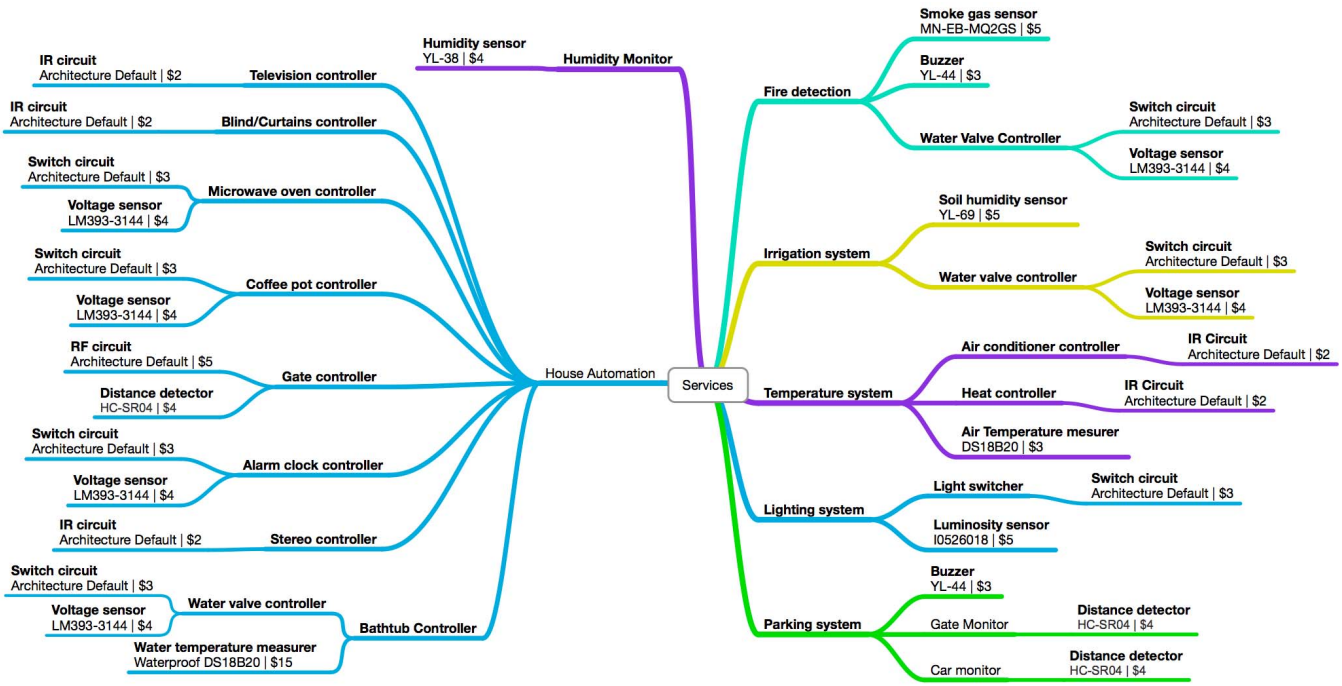


Table 4 Lighting system

Device name	*State variable (SV)*	*SV allowed values*	*Reading circuit (RC)*	*Writing circuit (WC)*
Light	Power	On, off	Luminosity sensor	Switch circuit

Table 5 Parking system

Device name	*State variable (SV)*	*SV allowed values*	*Reading circuit (RC)*	*Writing circuit (WC)*
Crash alarm	Sound	On, off	-	Buzzer
Gate	Distance	2 to 700	Distance detector	-
Car	Distance	2 to 700	Distance detector	-

Table 6 Fire detection system

Device name	*State variable (SV)*	*SV allowed values*	*Reading circuit (RC)*	*Writing circuit (WC)*
Fire detector	Smoke	True, false	Smoke gas sensor	-
Fire alarm	Sound	On, off	-	Buzzer
Fire sprinkler	Power	On, off	Voltage sensor	Switch circuit

Table 5 concerns the service of a parking system. It has a gate monitor to check if it is opened or closed, a car proximity detector to check if, in case the gate is not closed, the car is about to collide, and a buzzer to notify the car being about to collide.

A fire detection system (Table 6) requires smoke gas detectors distributed over desired perimeters. A buzzer can be used to notify nearby people about fire and a switch circuit can be used to open water valves to stop the fire, in case it is detected, and voltage sensor to check the state of valves.

Finally, a house automation system includes an IR emitter LED for controlling televisions, stereos, curtains and binds, as well as switch circuits with voltage sensors to control microwave ovens, coffee pots and alarm clocks. Bathtubs can be controlled using switch circuits to command water pipes actuators, voltage sensor to check which pipes are opened and a water temperature measurer to check which pipe should be opened to change water temperature to a desired level. Besides that, a RF circuit and a distance detector can be used to control garage doors.

5.2 Service-related actions

The UPnP protocol stack requires devices with internal services that have their own state variables and possible actions to execute. Table 8 shows the actions necessary to read or write state variables in the allowed range.

Table 7 Home automation system

Device name	*State variable (SV)*	*SV allowed values*	*Reading circuit (RC)*	*Writing circuit (WC)*
Television	Channel	1 to 1,000	-	IR circuit
	Volume	1 to 100	-	IR circuit
	Power	On, off	-	IR circuit
Curtain	Movement	Open, close	-	IR circuit
Microwave oven	Power	On, off	Voltage sensor	Switch circuit
Coffee pot	Power	On, off	Voltage sensor	Switch circuit
Gate	Movement	Open, close, stop	-	RF circuit
	Distance	2 to 700	Distance detector	-
Alarm clock	Power	On, off	Voltage sensor	Switch circuit
Stereo	Power	On, off	-	IR circuit
	Volume	1 to 100	-	IR circuit
	Station	1 to 6	-	IR circuit
Bathtub	Hot_valve_ actuator	Open, close	Voltage sensor	Switch circuit
	Cold_valve_ actuator	Open, close	Voltage sensor	Switch circuit
	Water_ temperature	–55 to 125	Water temperature measurer	-

5.3 Overall view of the proposed architecture functioning

To describe the architecture operating aspects we consider the situation arising when a master controller receives an UPNP SOAP XML message ordering it to execute the action ‘set_power’ into the device ‘light’, managed by a slave controller nearby. Indeed, the message commands to set that service’s ‘Power’ state variable to the value ‘off’. The master controller will search its database for the zigbee_id of that slave controller, as well as the writing pins corresponding to the ‘power’ state variable of the ‘light’ device, and the writing circuit signal message to send to that pin, in this case. Then, the master controller will send the serial ZigBee message to the slave controller containing the pin and value to be written. The slave controller receives the message and writes that value to the specified pin, which has a switch circuit connected to the light. This switch circuit, when receives zero, opens the relay that turns the light off, thus completing the SOAP action. After sending

the message to slave controller, the master returns an OK response to the SOAP request.

Table 8 Services actions

Action name	*Related state variable*	*Allowed values*
fire_detected	Smoke	True, false
set_sound	Sound	On, off
set_power	Power	On, off
get_power	Power	On, off
get_humidity	Humidity	0 to 1023
get_air_temperature	Temperature	–55 to 125
set_air_temperature	Temperature	15 to 30
get_water_temperature	Water_temperature	–55 to 125
set_operation_mode	Mode	Fan, cooling
	Mode	Fan, heater
set_movement	Movement	Open, close, stop
get_distance	Distance	2 to 700
set_channel	Channel	1 to 1,000
set_volume	Volume	1 to 100
set_station	Station	1 to 6
set_actuator	Hot_valve_actuator	Open, close
	Cold_valve_actuator	Open, close
get_actuator	Hot_valve_actuator	Open, close
	Cold_valve_actuator	Open, close

Moreover, when the light has its state changed, the slave controller detects that the luminosity sensor reading pin value has changed. When that happens, the slave controller notifies, through ZigBee, the master controller informing that the pin has changed to that new value. The master controller receives the notification, updates its database and sends GENA packages, through TCP/IP, to all UPnP 'light' devices subscribers notifying about the state variable change and its new value.

6 Services usage scenarios

All the reading and writing circuit models shown in Figure 2 can be directly plugged into an Arduino board and commanded to start performing their specific functions which means, in other words, that they are ready to be integrated to the slave controllers.

The UPnP environment comprises the devices, with their specific services, their state variables and allowed values, as well as the possible actions to be executed by each of them. Configured at master controller, the UPnP environment, together with connected slave controllers, enables smart devices capable of notifying their state and supporting operations on these states, setting the stage for unprecedented integration.

We described usage scenarios by proposing four applications that can make use of these ready to go devices, the architecture services and the integrated devices.

In all applications, an action request means sending an UPnP SOAP XML message or accessing the corresponding REST API URL. Both methods are provided and directed to the master controller through control URLs whose general format is http://master_controller_ip/control/slave_controller_name/device_name/service_name/ (Ferreira et al., 2013).

6.1 Routines app

The first application is the routines app which consists of two main functions: scheduled routines and reaction routines.

Scheduled routines are meant to execute actions into devices, changing state variables to desired values in a specified time and defined periodicity. For example, a routine can be created that turns off all 'light' devices at 8 AM every day. To run that routine, the app, daily at 8 AM, should only send action requests to each device control URL, with the action 'set_power' and state variable 'power' with value 'off'.

Reaction routines are meant to execute control commands triggered by receiving event notification. For example, a routine sounds the fire alarms and opens the fire sprinklers when fire is detected. To be able to operate, the app should subscribe, through UPnP, to the fire detector device. From that point on, the app will receive notifications of any sate variable change from the fire detector through the master controller GENA. Whenever the app receives a change on the state variable 'smoke' with new value 'true', it should send to each fire alarm control URL an action request with action 'set_sound' and the value 'on' for the state variable 'sound'. It should also send to each fire sprinkler device the action 'set_power' and the value 'on' to the state variable 'power'.

In a similar way, a reaction routine is created to perform interaction between parking system devices to sound the alarm when the gate is opened and the car is close to the device. It can also control the irrigation system, turning on the water sprinklers when the soil is dry and oppositely turning them off when the soil is soaked.

6.2 Controlling app

The second example is the controlling app which is responsible for sending control requests to one or multiple devices in order execute one or a group of actions per device. For every action to send, a delay time must be set.

This app can be used for instance to prepare a breakfast. When the person wakes up, she tells the controlling app to prepare the breakfast. It will turn on the microwave oven (which supposedly has bread with ham and cheese left prepared the night before) and the coffee pot. After a while, the controlling app will turn off both devices and the

breakfast will be ready. This scenario requires a command named 'breakfast' that sends, with delay 0, a control request to the devices coffee pot and microwave oven with action 'set_power' and state variable 'power' set to 'on'. The breakfast command should also send control request to device coffee pot, with delay 5 minutes, and microwave oven, with delay 1 minute, containing the action 'set_power' and state variable 'power' set to 'off'.

This app can be used to control every device that has state variables capable of having their state changed by a slave controller. It can be used to control temperature, television, stereo and all the other devices

6.3 Notifier app

A notifier app is conceived to sends SMS or e-mail notifications in case one or a range of state variables from devices assume a determined value. This app also can notify if a state variable exceeds, or drops below, a defined threshold value or keeps in the same state for more than a defined time interval.

Usage examples include notifications of fire detection, garage doors open for long periods (meaning that they were probably forgotten open), lights turned on for a long time, water temperature too high and improper for bath, and so on.

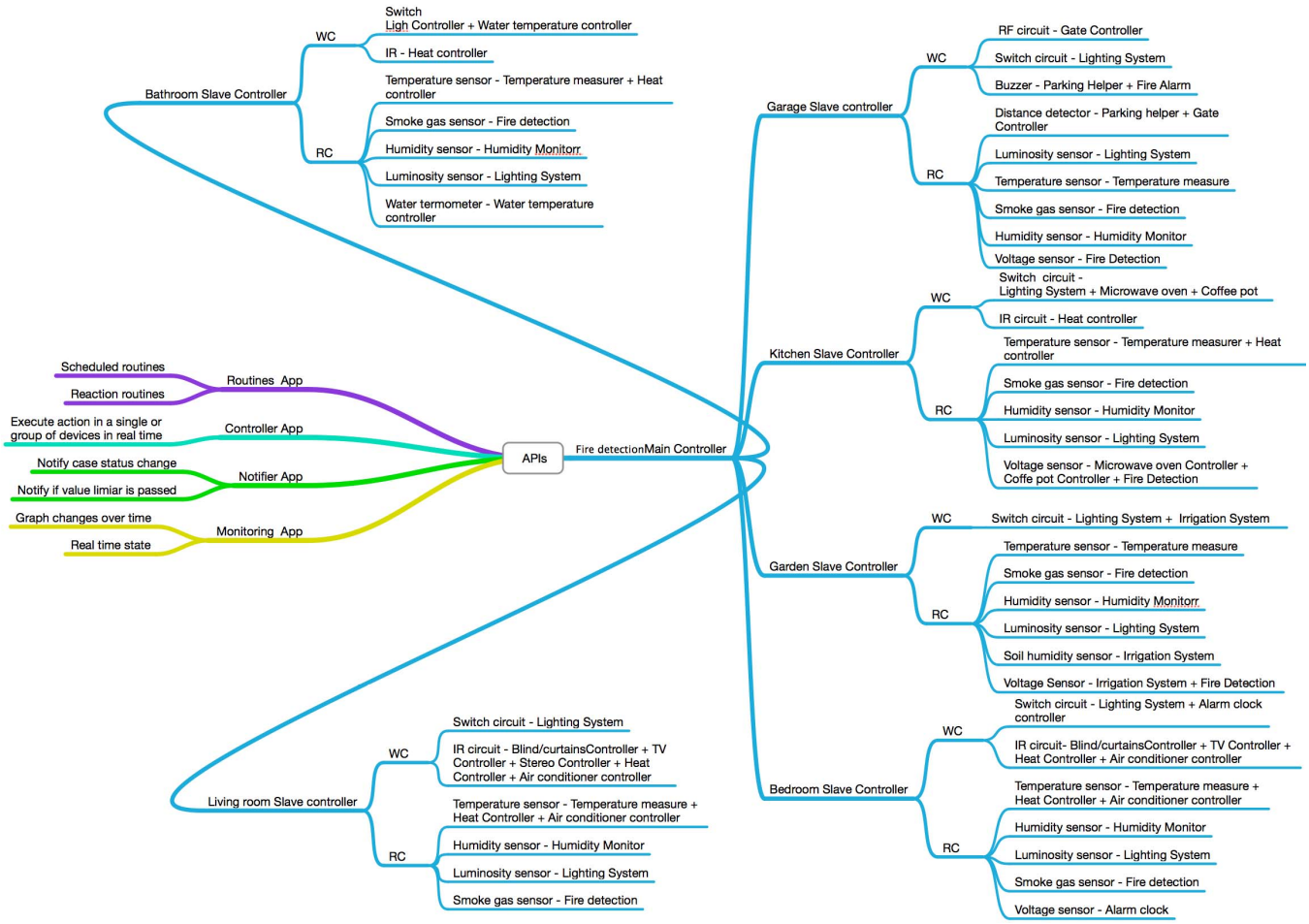
This app has to subscribe to all desired devices associated to UPnP event servers in order to receive GENA packages. Once it receives the events, it checks notifications metrics and decides to send the notifications.

6.4 Monitoring app

The fourth and last proposed application is the monitoring app. It keeps track of devices state variable values timely and can display this data in charts. This app can also check for a state variable value in real-time.

This app has a database that saves the slave controller name, as well as device name, service name, state variable name and value, and the gathering time when the values have been retrieved. To keep this tracking process, the app has to subscribe to all desired devices associated to UPnP event servers in order to receive GENA packages. To display the actual value of a state variable, it just checks its database for the latest value retrieved.

Figure 3 Detailed house scenario (see online version for colours)



6.5 House scenario

As an example to conclude the analysis of the proposed architecture, the services and the preceding applications are applied together into a house, comprising the integration of four different applications, seven services, and more than 20 different devices communicating over more than 30 circuits. The example is a house with one garage, one garden, one living room, one bathroom, one bedroom and one kitchen. The master controller is placed at the living room and for each of the six rooms there is a slave controller (Figure 3).

Figure 3 shows all the reading and writing circuits that should be attached to room devices and slave controllers in order to provide the services.

Once everything is in place, it is necessary to configure the master controller using the forms described in Table 2 of Ferreira et al. (2013), so as to specify the right slave controller and pin where each device reading and writing circuit is located and to which state variable they are attached to.

With this configuration in place, all devices are now available to be controlled via UPnP and the REST API. Any app can now be activated, such as the proposed apps in Section 4, using devices to provide different services.

Besides this home scenario, the proposed architecture can be applied to agriculture or forests, since ZigBee allows data to be passed from one slave controller to another until it reaches the correct destination. It is possible to irrigate large areas or check for soil humidity, temperature conditions, air humidity and light incidence in slave controllers in large areas. Also, mall parking systems aware of unoccupied parking spots and alarming collision situations for every spot can also be easily made, with similar configurations.

7 Conclusions

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

This paper contributes to setup an architecture responding to these requirements. The variety of reading and writing circuits and services allow encompassing most applications scenarios and the architecture itself allows easy creation of custom circuits to further extend it. Integration between end devices and end applications is transparent and uses well-known protocols and technologies.

Future improvements to this proposed model could solve security issues by adding authentication and confidentiality to the architecture layers. Furthermore, other application scenarios can be designed with public configurations that could add other writing and reading circuits. Devices that can already communicate in other technologies such as power line and DTMF should be studied to be integrated in the same APIs to make it more transparent for end applications.

Acknowledgements

The authors wish to thank the Brazilian Ministry of Planning, Budget and Management for its support to this work.

References

Anbya, M.F.B., Salehuddin, M., Hadisupadmo, S. and Leksono, E. (2012) 'Wireless sensor network for single phase electricity monitoring system via Zigbee protocol', *2012 IEEE Conference on Control, Systems & Industrial Informatics (ICCSII)*, 23–26 September, pp.261–266.

Araujo, R.B. (2003) 'Computação Ubíqua: Princípios, Tecnologias e Desafios'. in *XXI Simpósio Brasileiro de Redes de Computadores*, Natal, CE, SBRC, pp.45–115.

Arduino (2013) [online] http://www.arduino.cc/ (accessed 7 June 2013).

Bjelica, M.Z. and Teslic, N. (2010) 'A concept and implementation of the embeddable home controller', *MIPRO, 2010 Proceedings of the 33rd International Convention*, 24–28 May, pp.686–690.

Digital Living Network Alliance (DLNA) (2013) [online] http://www.dlna.org/ (accessed 28 May 2013).

Erdem, H. and Uner, A. (2009) 'A multi-channel remote controller for home and office appliances', *IEEE Transactions on Consumer Electronics*, November, Vol. 55, No. 4, pp.2184–2189, DOI: 10.1109/TCE.2009.5373786.

Ferreira, H.G.C., Canedo Jr., E.D. and Rafael, T.S. (2013) 'IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and Arduino', *The 9th International Conference on Wireless and Mobile Computing, Networking and Communications*, 7–9 October.

Gupta, V., Poursohi, A. and Udupi, P. (2010) 'Sensor.Network: an open data exchange for the web of things', *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 29 March–2 April, pp.753–755, DOI: 10.1109/PERCOMW.2010.5470533.

Hansmann, U., Merk, L., Nicklous, M.S. and Stober, T. (2001) *Pervasive Computing Handbook*, Springer.

Hwang, I-k., Lee, D-s., and Baek, J-w. (2009) 'Home network configuring scheme for all electric appliances using ZigBee-based integrated remote controller', *IEEE Transactions on Consumer Electronics*, August, Vol. 55, No. 3, pp.1300–1307, DOI: 10.1109/TCE.2009.5277992.

Kim, K-S., Park, C., Seo, K-S., Chung, I-Y. and Lee, J. (2007) 'ZigBee and the UPnP expansion for home network electrical appliance control on the internet', *The 9th International Conference on Advanced Communication Technology*, 12–14 February, Vol. 3, pp.1857–1860.

Kim, T., Lee, H. and Chung, Y. (2010) 'Advanced universal remote controller for home automation and security', *IEEE Transactions on Consumer Electronics*, November, Vol. 56, No. 4, pp.2537–2542, DOI: 10.1109/TCE.2010.5681138.

Kleinrock, L. (2010) 'An early history of the internet [history of communications]', *IEEE Communications Magazine*, August, Vol. 48, No. 8, pp.26–36.

Miller, B.A., Nixon, T., Tai, C. and Wood, M.D. (2001) 'Home networking with universal plug and play', *IEEE Communications Magazine*, December, Vol. 39, No. 12, pp.104–109, DOI: 10.1109/35.968819.

Schimpf, P.H. (2013) 'ARTK: a compact real–time kernel for Arduino', *International Journal of Embedded Systems*, February, Vol. 5, Nos. 1–2, pp.106–113, DOI: 200710.1504/IJES.2013.052176.

Solanki, U.V. and Desai, N.H. (2011) 'Hand gesture based remote control for home appliances: handmote', *2011 World Congress on Information and Communication Technologies (WICT)*, 11–14 December, pp.419–423.

Stenberg, D. (2013) *cURL and libcurl* [online] http://curl.haxx.se/ (accessed 12 August 2013).

Stirbu, V. (2008) 'Towards a RESTful plug and play experience in the web of things', *2008 IEEE International Conference on Semantic Computing*, 4–7 August, pp.512–517, DOI: 10.1109/ICSC.2008.51.

Universal Plug and Play Device Architecture (2008) v. 1.1, October [online] http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf (accessed 27 May 2013).

Universal Plug and Play Forum (2000) *Understanding Universal Plug and Play* [online] http://www.upnp.org/download/UPNP_understandingUPNP.doc (accessed 27 May 2013).

Wang, Z., Liu, Z. and Shi, L. (2010) 'The smart home controller based on zigbee', *2010 2nd International Conference on Mechanical and Electronics Engineering (ICMEE)*, Vol. 2, pp.V2-300–V2-302, 1–3 August, DOI: 10.1109/ICMEE.2010.5558422.

Want, R. (2000) 'Remembering Mark Weiser: Chief Technologist, Xerox Parc', *IEEE Personal Communications*, February, Vol. 7, No. 1, pp.8–10, DOI: 10.1109/MPC.2000.824564.

Weiser, M. (1991) 'The computer for the 21st century', *Scientific American*, September, Vol. 265, No. 3, pp.94–104.

ZigBEE Alliance (2013) [online] http://www.zigbee.org/ (accessed 8 August 2013).