Parameter-based Mechanism for Unifying User Interaction, Applications and Communication Protocols

Jie Song, Silvia Calatrava Sierra, Jaime Caffarel Rodríguez, Jorge Martín Perandones,
Guillermo del Campo Jiménez, Jorge Olloqui Buján, Rocío Martínez García and Asunción Santamaría Galdón

***Abstract*—In the smart building control industry, creating a platform to integrate different communication protocols and ease the interaction between users and devices is becoming more and more important. BATMP is a platform conceived to archive this goal. In this paper, the authors describe a novel mechanism for information exchange, which introduces a new concept Parameter and uses it as the common object among all the BATMP components: Gateway Manager, Technology Manager, Application Manager, Model Manager and Data Warehouse. Parameter is an object which represents a physical magnitude and contains the information about its presentation, available actions, access type, etc. Each component of BATMP has a copy of the parameters. In the Technology Manager, three drivers for different communication protocols, KNX, CoAP and Modbus are implemented to convert devices into parameters. In the Gateway Manager, users can control the parameters directly or by defining a scenario. In the Application Manager, the applications can subscribe to parameters and decide the values of parameters by negotiating. Finally, a Negotiator is implemented in the Model Manager to notify other components about the changes taking place in any component. By applying this mechanism, BATMP ensures the simultaneous and concurrent communication among users, applications and devices.**

Keywords–home automation; multi-protocol; user interaction; driver; KNX; Modbus; BATNet, CoAP

I. INTRODUCTION

Due to the fast development of home automation technologies, nowadays, it is becoming more and more important to create general platforms or standards to integrate all the possible entities that could be involved in a scalable smart home control system. Basically, we consider three entities: devices, users and applications. First, devices refers to those sensors, switches, HVAC systems, etc., which could be measured or controlled. Many different communication protocols have been designed to enable communication with those devices. KNX, Modbus, LonWorks and X10 are examples of wired communication protocols, while EnOcean, ZigBee, 6LoWPAN and Z-Wave are wireless ones. Many projects have been proposed for integrating two protocols, most of which achieve the goal by making conversions between the data formats. [1] However, this mechanism is not flexible, since it does not allow a third protocol to be integrated easily. Second, users usually do not care about the protocols used by the devices as long as they can read and control them. And moreover, the configuration process should be as easy as possible. Third, the integration of different applications is proposed as a strategy to make the control system more flexible and scalable. Thanks to this way, an application can focus on a specific function, have its own algorithms and user interface. And more important, third-party developers can contribute to the platform by adding new applications. As a result, to achieve the interaction among the three most important entities mentioned above, the authors have designed a platform, BATMP, which is able to integrate different communication protocols, ease the interaction between users and devices, and allow the development of applications. BATMP consists of five components, the Gateway Manager, the Technology Manager, the Application Manager, the Model Manager and the Data Warehouse, each of which is conceived to deal with users, devices, applications and data management.

This paper is mainly contributing to solve two problems in BATMP. First, the creation of a concept which can be used and understood by users, application and devices at the same time. Second, keeping the information consistency among users, devices and applications, which means that any change made by one of these three entities should be notified to the others. By solving these two problems, the information exchange inside BATMP is assured to be simultaneous and concurrent. To solve the first problem, the authors introduce a new concept "Parameter" to encapsulate the exchanged information. A parameter encapsulates the information about a physical magnitude and the different ways to request and present it. For example, a parameter of illumination level contains the data about the value of the illumination measured, the unit which is Lux, the data type which is integer, etc. The structure and general usage of parameters will be explained in Section II. A parameter can be measured by a device, be understood by a user and be used by an

application, which is explained in detail in the Section III - V. Section III will illustrate how the Technology Manager converts a device into parameters by implementing drivers of different communication protocols. Section IV will illustrate how the users interact with the parameters, either directly or through a scenario. Section V will explain how the applications subscribe to the parameters and in addition, how they negotiate when several applications subscribe to the same parameter. Section VI will solve the second problem mentioned above, which is how to use a Negotiator to keep the information consistency among these three components. Finally, conclusions are presented in Section VII.

II. Parameter in BATMP - Introduction

To carry out the information exchange among users, devices and applications, a new concept "Parameter" is introduced in BATMP. A Parameter is an object which represents a physical magnitude and in addition contains the information about its presentation (such as unit and data type), available actions (such as modifiable or read only), access type, request frequency, etc. It is used as the common object for communication among all the BATMP components.

A. Parameter in BATMP components

In general, BATMP comprises five components: the Gateway Manager (GM), the Technology Manager (TM), the Application Manager (AM), the Model Manager (MM) and the Data Warehouse (DW). The function of each component is explained in detail in [3]. As illustrated in Fig.1, GM, TM and AM hold a Parameter Pool for their own use. TM first uses several drivers to convert the physical devices to parameters and then uses the Parameter Reader and the Parameter Listener to ask or change their values to accomplish the interaction with devices. GM uses the parameters to accomplish the interaction with users and the Rule Engine checks the values of each parameter constantly to decide if a scenario or rule should be triggered. AM allows the applications to subscribe to the parameters and decide the value of a parameter after a negotiation among several applications. Each Parameter Pool contains a copy of all the parameters in the BATMP and each parameter is identified by a given id. Parameters with the same id should be consistent in all these three components. Therefore, any change of the parameter in one of the components should be notified to the others through the Negotiator in the Model Manager.

B. Parameter structure and related classes

The parameter is designed to be capable of handling all kinds of use cases of information exchange in BATMP. Fig.2 shows the class UML of the parameter and its related classes. This structure of parameter makes it possible to reach an agreement on the following issues among users, devices and applications:

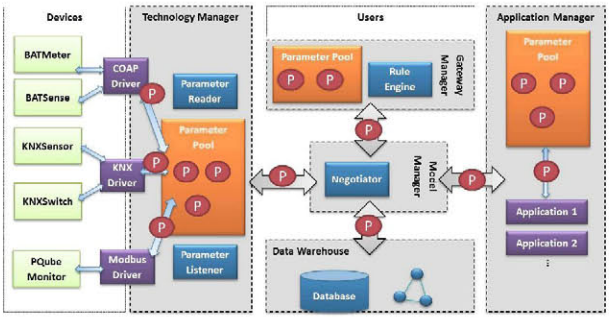

Figure 1. Illustration of use of Parameter in BATMP components

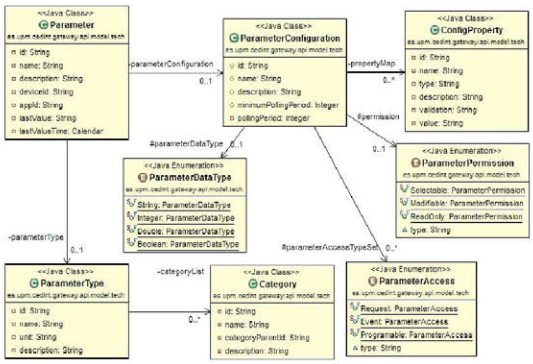

Figure 2. Class UML of parameter and the related classes

1) Basic Information: Each parameter includes a series of basic information, such as name, category, unit, etc. The information is shown to users in the User Interface and allows users to sort the resources according to different criteria. Meanwhile, an application can get a set of parameters in which it is interested. For instance, a "Consumption monitoring" application would be interested in retrieving just those parameters within category "Consumption".

2) Status Information: This information includes Last Value and Last Value Time. Thanks to them, users can know the latest information of the parameter, the Rule Engine can trigger a rule if the given conditions are satisfied, the application can make decisions basing on several parameter values, and the driver can decide if a new request should be sent to the devices for retrieving a new value.

3) The actions: The attribute ParameterPermission (Selectable, Modifiable and ReadOnly) declares that whether the parameter value can be changed, either by a user or an application. And the attribute ParameterDataType (String, Integer, Double, Boolean, etc.) informs the users and the applications about the valid value format that should be passed to the driver to control the device. For instance, to change the status of a switch, a boolean should be passed as the value, while to change the set temperature of an air conditioning, an integer should be passed.

4) Access Type: The attribute ParameterAccessType (Request, Event, Programmable) decides if the user/application is able to access the real time parameter value. Also, TM uses different manners according to this attribute to communicate with the drivers to retrieve parameter values from the devices. The details will be explained in Section III-D.

5) Request Frequency: For each parameter that defines Programmable as its ParameterAccessType, it is mandatory for the driver to set "Minimum Polling Period", which indicates the shortest time interval that can be set between two requests, to prevent the potential problem of abusing of requests. On the basis of this, the user can define the "Polling Period" of each parameter, which should be larger than the "Minimum Polling Period" of this parameter. This value will be used finally as the real time interval between two requests.

6) Validation: The Validation defines the regular expression and possible range of a parameter value in order to assure that all the values set by the user/application/driver follow the valid format and are in the correct range. For example, a parameter "Power consumption" can only use numbers as its value and be within the range [0, ∞).

III. Control of Devices through Parameters

As mentioned above, a communication protocol defines a list of rules and the data format for exchanging messages among devices. The device can be a sensor, an actuator, a computer, etc. In home automation, each device sends or receives messages following one or several communication protocols. In BATMP, for connecting to devices through different communication protocols, several drivers are implemented. The function of a driver is to convert a physical device into a logical device that contains a set of parameters. In this section, we consider drivers of three protocols, KNX, 6LoWPAN(BATNet) and Modbus as examples to illustrate how this parameter-based mechanism enables the devices using different communication protocols working together.

A. KNX devices and KNX Driver

KNX is a standardized OSI-based network communication protocol for intelligent buildings. The most common form of KNX installation is connecting various devices together by a two-wire bus. The logical topology or sub network structure of KNX allows 256 devices on one line. Each function of the device will be assigned to a group address and one or several flags. The flags control the communication possibilities of an object on the bus, which could be R(Read), W(Write), C(Communication), T(Transmit) or U(Update). [8] Fig.3 shows a part of the KNX network configuration in ETS4 (the KNX software tool) and how the KNX Driver converts a KNX device into a set of parameters. As we can see, Blind1 is a blind which can be opened at a specified percentage. Two group addresses are used

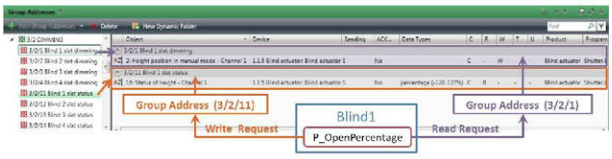

Figure 3. Parameter creation of KNX Device in KNX Driver

for handling the percentage status of Blind1: "3/2/1 blind1 dimming" with flag W for setting its percentage and "3/2/11 blind1 status" with flag R to read its percentage. In the KNX Driver, corresponding to the physical blind installed in a home, a logical device Blind1 is created, which contains a parameter named P_OpenPersentage. Therefore, if a user wants to know the status of Blind1, which is namely, the last value of the parameter P_OpenPersentage, TM will send a READ request through the KNX Driver to the group address "3/2/1" for retrieving the value. And if the user wants to change the status of Blind1, which is actually in this case, to change the parameter P_OpenPersentage, the driver will send a WRITE request containing the target percentage to the group address "3/2/11". In both cases, the user does not need to know what is a group address and which one is used.

B. BATNet and CoAP Driver

BATNet is a set of wireless IoT/6LoWPAN devices such as smart meter (BATMeter) [2], smart plug (BATPlug), ambient sensor (BATSense)[4], lighting control (BATStreetLight), etc. All BATNet devices communicate with BATMP through 6LoWPAN and CoAP protocols. 6LoWPAN stands for IPv6 over Low power Wireless Personal Area Networks, which is used for the communication at network layer. And CoAP (Constrained Application Protocol) is an application layer protocol used in very simple electronics devices, allowing them to communicate interactively over the Internet. To achieve the communication between BATNet devices and BATMP, a CoAP Driver is implemented. Fig.4 illustrates how a CoAP Driver converts a physical device BATMeter into a set of parameters.

BATMeter is a power meter sensor designed to be installed in an electrical board to measure the real time voltage, current, power, energy consumption, etc., of the electric lines. Each BATMeter can measure a maximum of six lines and for each lines, it measures six properties: Voltage, Current, Real Power, Apparent Power, Real Energy and Apparent Energy. As each current line works independently, the CoAP Driver will create a logical device for each line and each logical device includes six parameters. For querying the value measured by a BATMeter, the CoAP Driver needs to send a request containing the command "/values GET" to it. As long as a query request is received, the BATMeter will send a message which contains the values of all properties of all the current lines connected, as shown

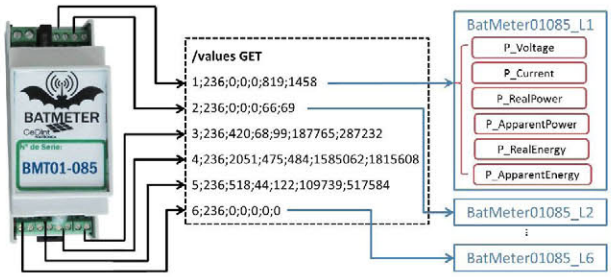

Figure 4. Parameter creation of BATMeter in CoAP Driver

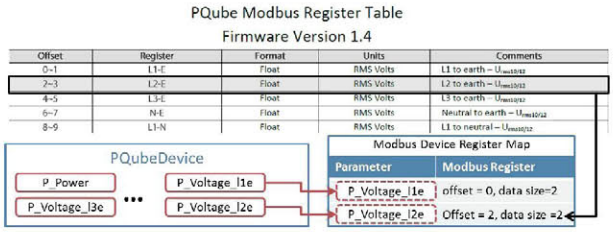

Figure 5. Parameter creation in Modbus driver

in Fig.4. Each line of the message has the following format: line_number;Vrms;mArms;W;VA;Wh;VAh, which corresponds to Voltage, Current, Real Power, Apparent Power, Real Energy and Apparent Energy. The CoAP Driver will parse the message and get the value corresponding to each parameter accordingly.

C. Modbus devices and Modbus Driver

Modbus is a serial communication protocol for connecting industrial electronic devices. Modbus allows the connection of approximately 240 devices in the same network and sends the measurements to a supervisory computer. In BATMP, we have connected a PQube device, a energy monitor manufactured by Power Standards Lab, to record the power consumption of the building. [6] Data collected from PQube is sent to BATMP through a Modbus Driver.

Each Modbus device holds a register map and each register corresponds to a value which can be measured, such as temperature, voltage, humidity, etc. Fig.5 shows a part of the PQube Modbus Register Table and the process of parameter creation. The data sheet can be downloaded at [5] after specifying the device model. As shown in the table, each register corresponds to a different offset and a data size of one or two words. As a result, the Modbus Driver creates parameters of Modbus devices according to the offset and the data size of each register. In this example, the register with offset 2 and data size 2 corresponds to the voltage of line 2 to earth. Therefore, a parameter P_Voltage_l2e is created for this register. When a Modbus message is received, the Modbus Driver will take the 2 bits value at the offset 2 and assign it to the parameter P_Voltage_l2e.

D. Measuring parameters' values

All the parameters created by the drivers are kept in the Parameter Pool and according to the AccessType of the parameters, TM handles them in different manners.

1) Request: indicates that the value of this parameter is measured or changed when a user requests. For handling this kind of parameter, TM keeps listening to GM. As soon as a request is received, it will ask the driver to measure or change the parameter value immediately. For example, the parameter P_OpenPercentage created in section III-A has Request as its AccessType, and when a user tries to change its value through GM, TM will receive the order and execute the action accordingly.

2) Programmable: indicates that the value of this parameter is measured according to a fixed frequency. In BATMP, this frequency is configured by defining the "polling period" in the Parameter Configuration. For handling the parameters with this attribute, TM uses a Parameter Reader to ask the drivers about the values of the parameters every polling period. For example, all the parameters created for the BATMeter in section III-B have Programmable as their AccessType and a user can define "query the BATMeter every 15 minutes" by setting polling period to 900.

3) Event: indicates that the value of this parameter will be sent whenever a change happens. The sensor will send a signal spontaneously to the driver when a motion is detected and the driver should notify this change to TM. For handling these type of parameters, TM uses a listener to listen to all the drivers and accepts the information when a new parameter value is passed by a driver. For example, a motion sensor can detect the presence of people in a room, which is represented as a parameter "Presence" in BATMP.

IV. User Interaction with BATMP through Parameters

The user interaction is accomplished in the Gateway Manager. A user can manage the parameters in two ways: read or change a parameter value directly or define a scenario and a set of rules to control the parameters according to some conditions.

A. Direct control over parameters

Each BATMP user can have two kinds of permissions over a parameter: reading permission, which allows the user to read the parameter value, or writing permission, which allows the user to read and change the parameter value. The parameter value can be changed only if two conditions are fulfilled: first, the parameter has to be modifiable or selectable. And second, the user who made the request should have been granted writing permission over this parameter by the administrator of BATMP.

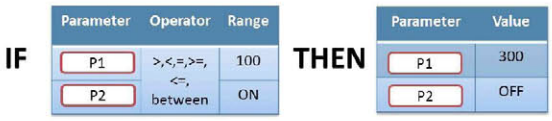

Figure 6. Rule example

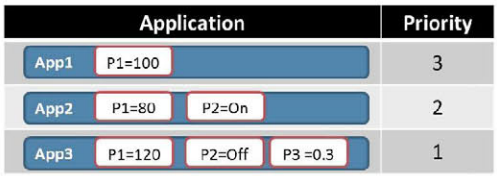

Figure 7. Application Negotiation

B. Control parameters by scenarios

BATMP allows a user to define various scenarios when the user wants to control different devices and parameters automatically under certain conditions. The user can activate a scenario manually or configure a trigger to activate a scenario at fixed days of a week or on some specified dates. Each scenario has a set of rules, which indicates that the actions should be carried out when the scenario is active. All the rules follow a IF-THEN form, as shown in Fig. 6. If one and/or several parameters satisfy the conditions, then the parameters in the result should be set to the certain values. In GM, a Rule Engine has been designed to check constantly about which scenario should be active at a given moment and if some actions should be carried out according to the value of each parameter from the Parameter Pool.

V. Working with Applications through Parameters

In BATMP, an application is a program designed for achieving certain functions on its own consideration. An application can take advantage of all the basic configuration of BATMP such as user, device, parameters, but also have their functions, such as an optimized algorithm to manage the data, a different interface for displaying, etc. So far, several applications have been developed such as GreenLab, an application for monitoring a Green House and BATStreet-Light, an application for controlling the lighting system in a campus according to the illumination and presence of people and cars.

A. Parameter subscription

Each application can subscribe to a set of parameters. Once the subscription is performed, the application can read the value and change the value of the parameter. Besides, if the value of the parameter is changed by others, which could be an application, a user or a driver, all the applications who have subscribed to this parameter will receive a notification about this change.

B. Negotiation among applications

Sometimes, there may be more than one application subscripting to the same parameter. In this case, a negotiation will be carried out to decide the final parameter value. When a user creates a new scenario, it is mandatory to define the priority of each application under this scenario. As a result, in different scenarios, the applications are assigned different priorities. For example, in the scenario "Out Of Home", the application "Security" will be assigned the highest priority and "HVAC Comfort" will be assigned the lowest one as nobody is at home. On the contrary, in the scenario "At home", the priority of "Security" will be set lower than in the previous case and "HVAC Comfort" will be assigned the highest priority. When a parameter can be changed by several applications, AM has to coordinate among these applications according to their current priorities and the one with the highest priority will get the right to decide the parameter value. Fig.7 illustrates the negotiation process among the applications. As we can see, APP1, APP2 and APP3 all subscribe to P1, and both of APP2 and APP3 subscribe to P2. With the priorities as APP1 > APP2 > APP3, the final result of the negotiation turns out to be P1=100, P2=ON and P3=0.3.

VI. Negotiator

As explained above, in BATMP, parameters are used as common objects for communications among different components. Each component keeps a copy of the parameter set for its own use. As a result, any change in a certain component of a parameter should be communicated to the other components. In general, the value of one parameter could be changed in three components, GM, TM and AM. For resolving potential conflicts, a Negotiator is created in MM to notify other BATMP components about the changes taking place in one of the components so that the others can make proper decisions accordingly.

A. Gateway Manager (GM)

The changes happening in GM come from the decision of a user or the rule engine. For instance, when a user wants to change the illumination percentage of a dimmer. As illustrated in Fig. 8(a), after the user selects the value:

1) One message, containing the parameter id and target percentage, arrives at GM through REST.
2) GM informs the Negotiator about the intentions of the user
3) The Negotiator sends the parameter id and target percentage to TM, asking if the value of this parameter can be changed. As a result, inside TM, the driver of this parameter is located and the value in the physical world (the illumination percentage of the light) is changed.

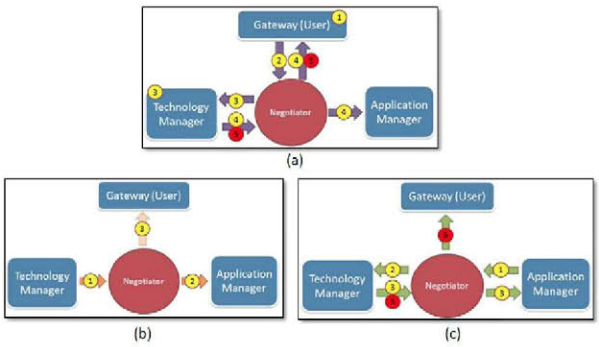

Figure 8. Negotiator

4) If it succeeds, TM will send a success message containing the set value and time-stamp to the Negotiator, which then sends the success message to AM to update its parameter value and to GM to show the result to user.
5) If it fails, TM will send an error message containing the cause to the Negotiator and then the Negotiator sends the error message to GM to show it to the user.

B. Technology Manager (TM)

The changes happening in TM are those values input by each driver. For instance, when TM sends GET request to a BATMeter, it will then send a message containing the measured value back to TM through the CoAP driver, illustrated in Fig. 8(b):

1) TM sends a message containing the new value, the parameter id and the time-stamp to the Negotiator.
2) The Negotiator sends the information to AM and then AM updates its parameter value.
3) There is no need to send the information back to GM as the user gets the information on request, namely, the data come from DW directly.

C. Application Manager (AM)

The changes taking place in AM are those negotiated among applications. For instance, at 9:00 in the morning, an application of illumination control wants to turn on a light as it detects someone entering. At the same time, an application of consumption wants to maintain the space at minimum consumption. As a result, due to the negotiation between these two applications, AM decides to open the blinds instead of turning on the light. As illustrated in Fig. 8(c):

1) AM sends a message containing the parameter id and the target value to the Negotiator.
2) The Negotiator tells TM to change the parameter value.
3) If it succeeds, TM will send a success message containing the set value and the time-stamp to the Negotiator. Then the Negotiator will send the success message to AM to update its parameter value.
4) If it fails, TM will send an error message containing the cause to the Negotiator and then the Negotiator will send the error message to AM.

VII. Conclusion

In this paper, the authors introduce a novel mechanism for information exchange in an intelligent building control platform, BATMP. A new concept "Parameter" is introduced, which is used as the common object among different components of BATMP. In the Technology Manager, three drivers are implemented for dealing with different communication protocols, KNX, CoAP and Modbus. They accomplish the connection with the devices by carrying out the device-parameter conversion. Through the Gateway Manager, the user is able to interact with BATMP by directly controlling a parameter or by defining a scenario to enable various parameters working collaboratively. In the Application Manager, the applications can subscribe to the parameters for reading or writing to them and if a parameter is subscribed by several applications, a negotiation will be carried out to decide the final parameter value according to the priorities of each application. Finally, to ensure the consistency among the three components mentioned above, a Negotiator is implemented in the Model Manager to handle and notify the changes happening in any of the three components. In a word, the use of Parameter enables the simultaneous and concurrent interaction among users, applications and devices.

References

[1] Woo Suk Lee; Seung Ho Hong, "Implementation of a KNX-ZigBee gateway for home automation", Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on , vol., no., pp.545,549, 25-28 May 2009

[2] Guillermo del Campo, Eduardo Montoya, Jorge Martín, Igor Gómez, Asunción Santamaría: BatNet: A 6LoWPAN-Based Sensors and Actuators Network. Ubiquitous Computing and Ambient Intelligence. 7656, 58–65 (2012)

[3] Jaime Caffarel, Guillermo del Campo-Jiménez, Jorge M. Perandones, César Gomez-Otero, Rocío Martínez and Asunción Santamaría: Open Multi-Technology Building Energy Management System. In: Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress, pp.397–404. St. Petersburg (2012)

[4] Jorge Martín, Igor Gómez, Eduardo Montoya, Jie Song, Jorge Olloqui, Rocío Martínez. BatNet: An Implementation of a 6LoWPAN Sensor and Actuator Network. ACM e-Energy. May 2013

[5] Power Standards Lab - Downloads, http://www.powerstandards.com/downloads.php [Accessed on 8th October 2014]

[6] Hodgson, S., "Power quality monitoring case study", Power in Unity: a Whole System Approach, IET Conference on , pp.1,20, 16-17 Oct. 2013

[7] Communication Reliability in the Intelligent Building Systems, http://www.knx.org/fileadmin/downloads/05

[8] Flags-KNX Association, http://www.knx.org/fileadmin/template/documents/downloads_support_ menu/KNX_tutor_seminar_page/Advanced_documentation/02_Flags_E1008a.pdf [Accessed on 8th October 2014]