Offering Web-of-Things Connectivity to Building Networks

Gérôme Bovet
Telecom ParisTech
46 Avenue Barrault
Paris, 75013 France
gerome.bovet@telecom-paristech.fr

Jean Hennebert
University of Applied Sciences
Fribourg
Bd. de Pérolles 80
Fribourg, 1700 Switzerland
jean.hennebert@hefr.ch

http://dx.doi.org/10.1145/2494091.2497590

Abstract

Building management systems (BMS) are nowadays present in new and renovated buildings, relying on dedicated networks. The presence of various building networks leads to problems of heterogeneity, especially for developing BMS. In this paper, we propose to leverage on the Web-of-Things (WoT) framework, using well-known standard technologies of the Web like *HTTP* and *RESTful APIs* for standardizing the access to devices seen from an application point of view. We present the implementation of two gateways using the WoT approach for exposing KNX and EnOcean device capabilities as Web services, allowing a fast integration in existing and new management systems.

Author Keywords

Building networks, Web-of-Things, EnOcean, KNX, Gateways

ACM Classification Keywords

D.2.11 [Software Engineering]: Software Architectures.; H.4.3 [Information Systems Applications]: Communications Applications.

Introduction

Building management systems (BMS) are nowadays very common in different types of buildings such as offices, manufactures or even private households. Those BMS rely on a variety of sensors and actuators linked with each other forming a dedicated building network. At origin, only the temperature of a building was controlled in a very simple way over global or room-wise thermostats, targeting a threshold temperature value. Since then, motivated by raising energy costs and by the growing importance of the comfort, new strategies involving other kind of devices and being much more complex have been developed. New generation of BMS are taking advantage on many kinds of sensing and actuating devices, for managing the HVAC (Heating, Ventilation and Air Conditioning), the lightening, doors opening, windows and blinds control, and also for restricting access. This constellation of devices has led buildings to become "smart" and are now relying on information systems connected to building networks like KNX, BACnet or LonWorks. The most installed network in Europe is KNX. The EnOcean network is gaining importance in buildings, based on energy harvesting wireless technologies [?].

Unfortunately, those building management networks do not propose any standardization from an application point of view to dialogue with interconnected devices. Because of this situation, BMS combining multiple networks are uncommon. Buildings where the network should evolve with new devices that are not compatible with the actual one, or where extending the wiring is not feasible because of physical constraints are good examples of the resulting network heterogeneity [?], as illustrated in figure 1. Even if gateways encapsulating the specific building telegrams in IP packets are existing, it actually exists no standard at the application level, having as consequence every building network protocol to be understood and implemented by the BMS.

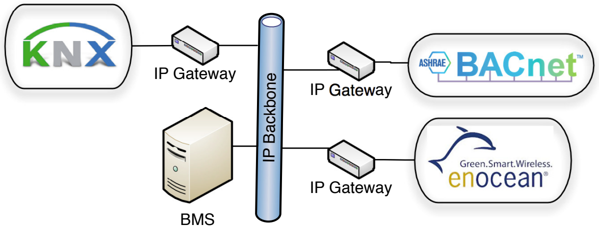

Figure 1: Network heterogeneity in smart buildings

Looking now at Internet and Web technologies, *Web services* are nowadays trendy and widespread, allowing interoperability between heterogeneous information systems (IS). Their key benefits are in being platform independent and using well-known standards for structured exchange. The Simple Object Access Protocol (SOAP) is an example of Web service protocol specification relying on Extensible Markup Language (XML) for its message format and Hypertext Transfer Protocol (HTTP) for message negotiation and transmission [?]. Unfortunately, because of the large overhead of XML and the complexity of the service description language WSDL, SOAP is not well suited for accessing sensors and actuators considered as *things*. On the other hand, the resource oriented architecture (ROA) part of the emerging *Web-of-Things* (WoT) concept, offers new ways for accessing things, which are more suited for BMS [?].

In this paper, we propose an approach for KNX and EnOcean networks to become homogeneous, allowing access to devices in a Web-of-Things manner. By taking advantage of the bests practices of the WoT, we guarantee a fast integration of both building networks in every control system. In addition to this, we put importance on the fact that our gateways must be simple to use, low-cost and easily integrable in an existing environment.

This paper is organised as follows. The next section refers and summarizes related work. The application of the Web-of-Things to the KNX and EnOcean networks is discussed in the third section. The fourth describes the implementation of the gateways. Performance tests in a existing buildings are shown in the fifth section. The sixth section concludes our paper and provides insights on further research.

Related work

Cooltown [?] is one of the early projects considering people, places and things as Web resources. A new interaction approach using HTTP GET and POST for manipulating things was introduced. Lately, with the evolution made in embedded systems, it was possible to embed Web servers directly on sensors and actuators. The *WebPlug* [?] framework introduce so-called *mashups*, relying on the Web-of-Things paradigm, where sensors and actuators play a central role.

Trying to ease the development of applications using KNX devices has been explored in different works. A first attempt was realized with the BCU SDK [?], which consists of a script generating C++ classes representing devices capabilities. A more Web oriented approach has been realized in [?]. The principle was to expose KNX functionalities as Web services by using the oBIX (Open Building Information Exchange) standard, which is a special XML schema for representing building data and operations. Unfortunately, oBIX is not at all widespread in BMS, probably because of its relatively complex XML schema. In addition to this, the proposed implementation does not allow an easy integration of the gateway in an existing environment, requiring an important configuration effort for large networks. The openhab [?] project is a BMS system offering interfaces to various building networks, including KNX. An integration of EnOcean is actually under development. Although being a complete solution, it does not allow for importing a KNX configuration nor exposing stored data through REST services. Additionally the project is quite complex and asks for having a deep knowledge of its working. Our approach tackles these limitations by taking advantage of the WoTs simplicity and by being highly integrable in existing KNX infrastructures.

Mapping building networks to RESTful APIs

As required when using the WoTparadigm, every object is expected to embed a REST server offering an API located through self-descriptive URLs. This vision can unfortunately not be applied as such to building networks. Devices connected to KNX or EnOcean networks are not IP-enabled and will therefore not be accessible through URLs. In addition to this, those devices are very constrained and task oriented, which makes it impossible for most of them to embed a Web server. We propose to fill this gap by introducing gateways exposing devices functionalities in the form of RESTful APIs. The role of those gateways is to hide the complexity of each building network and to allow clients interacting with attached devices in a Web-of-Things manner. In other words, the devices will appear to other participants of the WoT as they would be embedding the API on themselves.

From KNX

KNX interworking, the KNX application layer, was thought to ensure the interoperability between devices from various manufacturers. It standardizes the structure and interpretation of telegrams payload data. As it is the case for many automation systems, system functionality is described by so-called functional blocks which ensures the interworking [?]. Logical parts of a device, such as a specific function are symbolized by those Functional Blocks (FBs). This principle can be illustrated with an example involving a light switch FB that is a logical function of a four channels relay. A functional block is always attached to only one device.

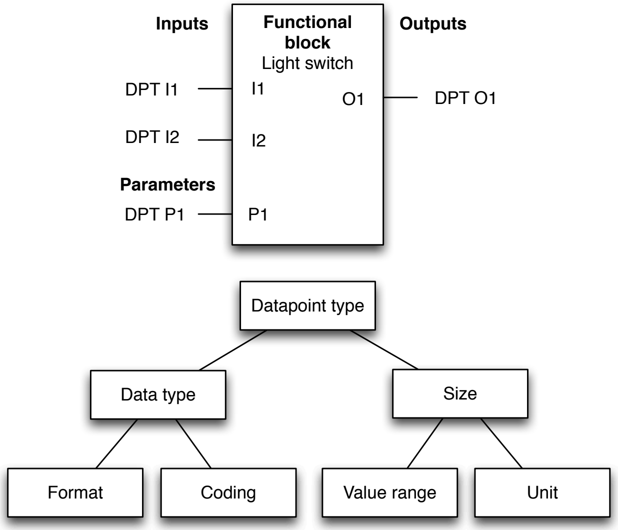

Figure 2: Functional block structure and datapoint type composition

As illustrated on figure 2, FBs are composed of a set of datapoints (DPs). Those datapoints are communication endpoints of devices allowing access to the functions of a block [?]. The inputs (DPT I) stand for states that can be altered by other devices. The parameters (DPT P) are configured by administrators or engineers for changing the behaviour of the FB. At last, the outputs (DPT O) give insights of the actual state of the block. Datapoints are also standardized in terms of syntax and semantics as visible in the lower part of figure 2, and are also organized in several categories depending on their FBs purposes. For example, our light switch provides the DP switch on off allowing to turn the light on or off. By knowing the datapoint type, one can find in the KNX specification all information regarding the datapoint, including the format, coding, value range and unit. The KNX protocol identifies two categories of DPs: group objects (GOs) and interface object properties (IOPs). The GOs are endpoints involved in group communications between producers and consumers basing on a multicast approach. This type of DP is used by sensors, actuators and control devices for exchanging information. On the other hand, IOPs are only for configuration and management purposes. One can address an IOP only with the physical address of the device.

The *Engineering Tool Software* (ETS) was developed by the KNX association for configuring a KNX infrastructure. With this software, administrators and engineers have the possibility to create the building hierarchy, the network topology, and finally to create group objects that will represent functionalities between devices. At the time of writing this article, ETS is the most used tool for KNX configuration in professional installations. As shown in figure 3, ETS exports projects in an archive composed of multiple XML files. The *knx_master.xml* file contains the description of all the DP types. The network topology, building organization and group addresses are stored in the *0.xml* file. Finally, there is a folder for every manufacturer, containing a XML file for each device type composing the network. The available DPs on a device are contained in the device file. Luckily, this archive is zipped without any security and contains easily understandable XML files, which allows developers to import all the network knowledge inside third-party applications. As illustrated in figure 3, we suggest here to perform a XSL transformation on the different files included in the archive to aggregate all useful information for our gateway into one single XML file.

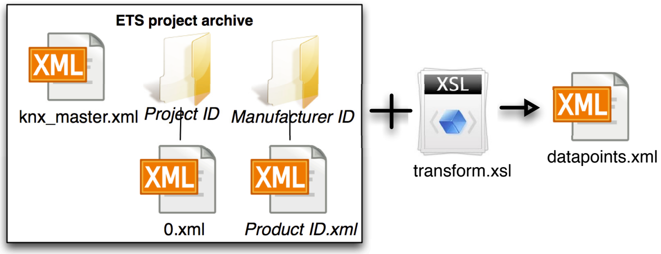

Figure 3: ETS project archive structure

As we previously pointed out, access to functionalities in a KNX network is achieved via group objects, which are a combination of datapoints and a group address. For offering interaction with KNX devices, we match group objects to REST services. This is obtained by using the XSLT output issued from the XSL transformation, which allow us to to compose an URL identifying a specific group object. For example, following URL would be linked to the data point shown in listing 1: *http://heating.office005.ground.leso.epfl.ch/dpt_switch*. The domain part is composed of the physical location of the device inside the building, completed by the domain name of the organization. The last part of the URL that represents the action to perform is the datapoint type name. We can now easily link group objects to URLs by following this rule:

http://<GROUP_NAME>.<LOCATION>.<ORGANIZATION_DOMAIN>/<DATAPOINT>.

Listing 1: Datapoint XML representation after XSL transformation

```
<datapoint stateBased="true"
name="Heating" desc="Status"
mainNumber="1"
 priority="Low" actionName="DPT_Switch"
 actionDesc="on/off" dptDesc="1-bit"
 dptBitsSize="1"
 location="Office005.ground.LESO">
     <knxAddress type="group">
          6195
     </knxAddress>
</datapoint>
```

Since now, clients can interact with the KNX devices through the gateway by sending HTTP requests. A GET request will result in the gateway returning the actual state of the DP. On the other side a POST request containing the new state inside the payload data will result in the KNX device changing its state. Representing the structural organization of the building inside the domain part of the URL opens a new dimension. By acting this way, we can hide the fact that the device is actually in a KNX network and not directly connected to an IP network. For users of the system, the device seems to be an IP one with its own DNS entry directly pointing to it. However, this brings a certain complexity for the DNS system as it musts contain entries matching with KNX groups redirecting requests to the gateway.

From EnOcean

EnOcean also works with its own application layer ensuring compatibility between devices. It relies on the

EnOcean Equipment Profile (EEP) providing information about the type of telegram and the representation of the data. Four types of telegrams are defined: RPS for Repeated Switch telegrams sent by push buttons, 1BS for One Byte telegrams sent by contacts and switches, 4BS for Four Byte telegrams sent by various types of sensors, and the VLD for Variable Length telegrams. EEPs are specified in an XML file provided by the EnOcean alliance that holds the description of all profiles, including frame composition, structure of data and meaning of data. Furthermore, every field of data of an EEP is identified by a so-called shortcut, such as TMP for temperature or HUM for humidity. The listing 2 shows the XML description of a datafield.

Listing 2: Datafield XML representation of an EEP

```
<datafield>
  <data>Temperature</data>
  <shortcut>TMP</shortcut>
  <description>
    Temperature (linear)
  </description>
  <info>
    DB_1 Temperature (8 bit) −40...0 C ,
    linear n=255...0
  </info>
  <bitoffs>16</bitoffs>
  <bitsize>8</bitsize>
  <range><min>255</min><max>0</max></range>
  <scale><min>−40</min><max>0</max></scale>
  <unit> C</unit>
</datafield>
```

Mapping EnOcean capabilities is much more complicated than KNX, even if the configuration of the network is very simple by just pairing devices together. Indeed there is no central management of the network like with ETS. Forming groups of functionalities by pairing devices is achieved by the user putting the actuator in a teach-in mode, and triggering a learn telegram on the sensor that have to drive the actuator. All the knowledge is distributed in the actuators and it exists unfortunately no way to retrieve it, because of the sensors sending broadcast telegrams and most actuators being only listeners. The major drawback of this working is the fact that it is therefore not possible to automate the mapping of the EnOcean network capabilities to REST APIs. An alternative is to let the user reproduce the pairing of devices by using a Web application hosted on the gateway. However the process can be simplified for the user by the gateway listening to teach-in telegrams. Those telegrams hold the EEP information, so that the gateway is now able to parse and interpret the telegrams. The user can then add the detected devices to the gateway by furnishing additional information such as a name and the location. This information is then used to map URLs with shortcuts of EEPs by following this rule: http://<SENSOR_NAME>.<LOCATION>.<ORGANIZATION_DOMAIN>/<SHORTCUT>. This allows clients to retrieve values of sensors by sending a HTTP GET request. However, as sensors are only senders and can not listen to command telegrams, the gateway will return the last captured data. As there is no way to discover actuators, their properties, like name, description, location and compatible EEPs have to be entirely filled in by the user. From now, it is possible for clients sending a HTTP POST request to actuate either by giving all shortcuts values of the EEP inside a JSON payload, or by specifying the shortcut inside the URL as follows: http://<ACTUATOR_NAME>.<LOCATION>.<ORGANIZATION_DOMAIN>/[<SHORTCUT>].

Common functions

Each gateway is extended with common APIs, especially interesting for reactive and proactive BMS. Developers can discover what devices are accessible and what are their capabilities by sending GET requests to URLs representing locations. The gateway will answer with a JSON indicating all sub-locations or devices inside the specified location. One can also gather knowledge about the available datapoints/shortcuts of a device by putting the *.../** placeholder at the end of the URL.

For reactive BMS, we offer a notification mechanism, notifying them as soon as the value of a sensor changes. Clients must first register themselves on the interesting resource and indicate the callback that will have to be called by the gateway. This is achieved by putting the *.../[un]register* keyword at the end of an URL pointing to an endpoint (e.g. *http://air.office005.ground.leso/hum/register*).

Finally, and dedicated to proactive BMS, we offer to them the possibility to announce their needs for storing history data on the gateway, and retrieving it later. For doing this, a client will interact with the *.../storage* sub-resource of an endpoint. By putting the *add/remove* keywords (e.g. *http://air.office005.ground.leso/tmp/storage/add*), and by indicating the number of days data should be stored in the case of an add command, clients have the possibility to start or end storing data. For retrieving the stored data, clients can either indicate the number of days one wants to go back in the history (i.e *.../storage?days=X*), or by specifying a period of time with a start and an end date (i.e. *.../storage?from=X&to=Y*).

Implementation

In this section, we provide further insights on the implementation of both gateways by following the principle described in this article. For running our gateways, we decided to use the *Raspberry Pi model B* platform that is low-cost and offers sufficient computing power for our purposes [**?**].

EnOcean modes

Due to the pairing principle of EnOcean, where sensors are learned on actuators, our gateway has to provide two modes of working: hybrid and bridge. Those behaviours are illustrated in figure 4. In the *bridge mode* it is the gateway's role to drive actuators, according to virtual groups created through the Web application. This simplifies the physical pairing of devices by needing only to learn the gateway on actuators. In this mode, the gateway will listen to telegrams sent by sensors, and if the sender of an incoming telegram is configured in bridge mode and is part of a group, the gateway will then retransmit the original telegram by replacing the sender ID by its own. The drawback of this mode is at adding some latency and essentially to be critical in case of the gateway failing (e.g. hardware problem, power outage, etc.), in which case users will no more be able to actuate (e.g switching lights, moving blinds, etc.) because of sensors being not directly paired with actuators.

In order to avoid those problems, we implemented a second mode of working, the *hybrid mode*. This time, not only the gateway is learned in actuators, but also the sensors. This allows to both the gateway and sensors to drive actuators. By having a direct link between sensors and actuators, we can provide a better user experience. As for the *hybrid mode*, the gateway will store each captured telegram for providing state values to clients.

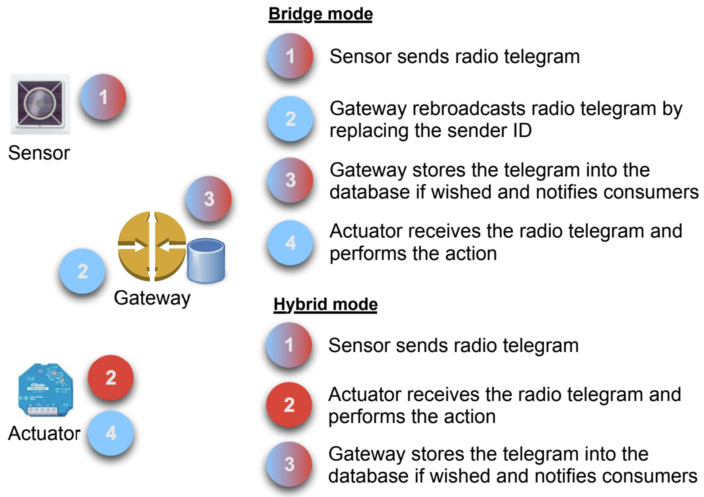
Bridge mode
1 Sensor sends radio telegram
2 Gateway rebroadcasts radio telegram by replacing the sender ID
3 Gateway stores the telegram into the database if wished and notifies consumers
4 Actuator receives the radio telegram and performs the action
Hybrid mode
1 Sensor sends radio telegram
2 Actuator receives the radio telegram and performs the action
3 Gateway stores the telegram into the database if wished and notifies consumers
Sensor
1
Gateway
2
3
Actuator
2
4

Figure 4: EnOcean gateway hybrid and bridge modes

Architecture

The KNX gateway relies on the *Calimero 2.0* Java library [**?**]. This library provides Java classes and methods for KNXnet/IP tunnel communications, and datapoint object representation allowing developers to build applications dedicated to KNX infrastructures. For the EnOcean gateway, as it currently exists no similar library, we had to develop our own library for capturing, parsing and sending telegrams over an USB dongle. The Web part of our implementation is based on Java servlets running on a *Jetty* server [**?**]. We argue our choice for Jetty because it is easily integrable on low-resources equipments thanks to its lightweight implementation, compared to other Web servers like Tomcat, Glassfish or Grizzly. MySQL was chosen as database engine. All components of the implementation are open source and free.

As illustrated on figure 5, our implementation relies on several logical modules very similar for each gateway.

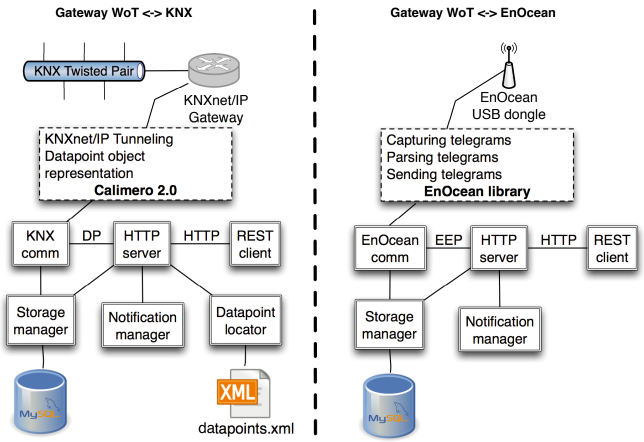

Figure 5: Architecture of the KNX and EnOcean gateways

Evaluation

The performance and limitations of the implementations were evaluated by several tests. For having realistic feedbacks we performed our tests on two buildings already installed with KNX and EnOcean networks. Starting from the evaluation results, we also propose some improvements of the gateway that could valuable for BMS.

KNX performance

For determining the gateway's performance, we selected to measure some key-values, like maximum number of requests per second, maximum simultaneous requests, notifications reaction time (i.e from the action on the KNX device until notification) and processing time of the ETS project archive during configuration. We base our measurements on an existing KNX installation of the four floors office LESO building located on the EPFL campus in Lausanne, Switzerland. This installation features 265 devices, distributed in 795 group objects and represents an average installation that can be found in several buildings.

Measure type	Result
ETS archive processing time	30 [min]
Maximum HTTP requests per second	45
Maximum simultaneous HTTP requests	620
Average event reaction time	33 [ms]

Table 1: KNX gateway performance measured on a real-life KNX installation running 265 devices

EnOcean performance

For the EnOcean gateway, we measured the same key-values as for KNX apart the ETS archive processing file. The gateway was installed at the College of engineering, Fribourg, Switzerland, where an office room of 20m^2 is equipped with 7 sensors, 3 lights/blinds switches and 4 actuators. This lab office is occupied by three people and represents a typical room that can be found in office buildings.

Measure type	Result
Maximum HTTP requests per second	82
Maximum simultaneous HTTP requests	631
Average event reaction time	29 [ms]

Table 2: EnOcean Gateway performance measured on a real-life EnOcean installation running 14 devices

Discussion

Table 1 summarises the performance of the KNX gateway running on a RaspberryPi. The XSLT being extremely resource consuming, this is reflected when processing the ETS archive file that needs quite a long time. However, as this operation has only to be performed during the configuration of the gateway, we can assume that it is not an important issue. The maximum HTTP requests per seconds is actually bottlenecked by the twisted pair of the KNX network offering only 9600b/s. The maximum simultaneous HTTP requests is limited by the Raspberry Pi. Nonetheless, we believe that the measured value is largely sufficient for common BMS operation. Finally, we observed a fast event response time that would typically allow a BMS to function in reactive mode. From table 2, we can clearly see that all results are also satisfying for the EnOcean. All limitations are due to the Raspberry Pi reaching its limits of processing capabilities. Anyway, the Raspberry Pi is largely sufficient for such applications and has to be considered as an alternative to classical PCs running gateways os serving as middleware.

Our approach implying access to the DNS server of the host network can be considered as a hurdle. The access to the DNS may be restricted by security policies in which case a dedicated DNS server has to be set up for the gateways.

Some developers where asked to build prototypical BMS applications using KNX or EnOcean devices, and this in various development languages. We obtained very positive feedbacks showing the benefits of leveraging on standardized and well-accepted protocols for reducing integration time.

Conclusions

Taking inspiration on Web-of-Things paradigms, we inspected the feasibility and benefits of using well-known Web standards like HTTP and RESTful APIs for interfacing building networks and building management systems. Our concepts and implementation were proven by being used on real installations, resulting in many positive feedbacks coming from developers pointing to the simplicity of use of WoT APIs.

From a global point of view, we believe that the integration of heterogeneous networks can be much simplified using WoT approaches. We also believe that in a near future, BMS will have to handle various networks based on different technologies, where smart gateways will prove their usefulness.

Our future works will cover security aspects of the gateway through authentication and encryption of data to prevent misuse. A mechanism for distributing the logical rules coming from proactive BMS, potentially distributed in the cloud, will also be explored.

Acknowledgements

The authors are grateful to the Swiss Hasler Foundation and to the RCSO grants from the HES-SO financing our research in this exciting area of smart buildings.