HAL
open science

A Web-of-Things Gateway for KNX Networks

Gérôme Bovet, Jean Hennebert

To cite this version:

Gérôme Bovet, Jean Hennebert. A Web-of-Things Gateway for KNX Networks. Smart SysTech 2013 European Conference on Smart Objects, Systems and Technologies, Jun 2013, Erlangen, Germany. hal-00824735

HAL Id: hal-00824735

https://hal.science/hal-00824735v1

Submitted on 22 May 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Web-of-Things Gateway for KNX Networks

Gérôme Bovet (1,2)
(1) LTCI
Telecom ParisTech ENST
46 Rue Barrault, 75013 Paris, France
gerome.bovet@telecom-paristech.fr

Jean Hennebert (2)
(2) HES-SO, EIA-FR
University of Applied Sciences of Western Switzerland
Boulevard de Pérolles 80, 1705 Fribourg, Switzerland
jean.hennebert@hefr.ch

***Abstract*—Smart buildings tend to democratize both in new and renovated constructions aiming at minimizing energy consumption and maximizing comfort. They rely on dedicated networks of sensors and actuators orchestrated by management systems. Building networks are often heterogeneous, leading to complex management systems having to implement all the available protocols and resulting in low system integration and heavy maintenance efforts. Typical building networks offer no common standardized application layer to build applications. We propose in this paper to leverage on the *Web-of-Things* (WoT) framework, using well-known technologies like *HTTP* and *RESTful APIs* to offer a simple and homogeneous application layer. We outline the implementation of a gateway using the principles of the WoT to expose capabilities of the KNX building network as Web services, allowing a fast integration in management systems.**

I. Introduction

In recent years, building management systems (BMS) have become very common in various types of buildings, such as offices, manufactures or even private households. They rely on a variety of sensors and actuators composing together a dedicated building network. Initially, the heating control of a building was very simple, relying on global or room-wise thermostats, targeting a threshold temperature value. Motivated by raising energy costs and by the importance of the comfort, more complicated strategies have since then been developed. Modern BMS include many kinds of sensing and actuating devices, managing the HVAC (Heating, Ventilation and Air Conditioning), the lighting, doors opening, windows and blinds control, and also security access systems. Buildings have become ”smart” and are now including complete information systems using dedicated building management networks for communication, as for example KNX, BACnet, or LonWorks. KNX is actually the most used network in Europe. One can find almost every kind of device compatible with this network. The physical support of the KNX network is most of the time based on the 29V powered twisted pair TP1 inherited from the EIB standard [1]. KNX supports also different types of physical connections like Ethernet, RF or power line. Another emerging standard for interconnecting sensors and actuators in buildings is Enocean, principally based on energy harvesting wireless technologies [2].

Unfortunately, such building management networks do not offer a standardised way to interact with devices connected to them from an application point of view. Due to this, it becomes difficult to build BMS combining multiple networks. This situation can be found in 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 [3]. This is leading to heterogeneous building management networks such as the one illustrated in figure 1. While it exists gateways encapsulating the specific telegrams of the building management network in IP packets, there is actually no standard at the application level, resulting in the BMS having to understand and to implement every network protocol.

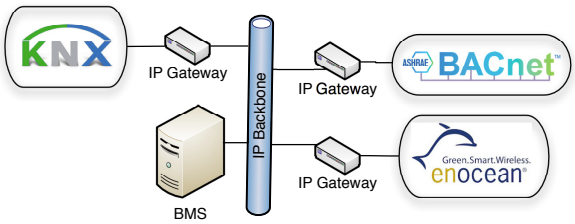

Fig. 1. Network heterogeneity in smart buildings

Looking now at Internet and Web technologies, so called *Web services* are nowadays widespread, able to make heterogeneous information systems (IS) interoperable. They are platform independent and use well-known standards for structured data 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 [4]. Unfortunately SOAP is not well suited for accessing sensors and actuators considered as *things*, because of the large overhead of XML and the complexity of the service description language WSDL. On the other hand, the emerging *Web-of-Things* framework (WoT), offers new ways for accessing things in a resource oriented architecture (ROA), which are more suited for BMS [5].

In this paper, we present the implementation of a gateway allowing access to devices connected to a *KNX* network in a Web-of-Things manner. By taking advantage of the bests practices of the WoT, we guarantee a fast integration of KNX in every control system. In addition to this, we put importance on the fact that our gateway 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. In Sect. 3, we provide an overview of the Web-of-Things paradigm. Its application to the KNX network is discussed in Sect. 4. Sect. 5 describes the implementation of the gateway. Performance tests in a real building are shown in Sect. 6. Sect. 7 concludes our paper and provides insights on further research.

II. Related work

One of the early projects considering people, places and things as Web resources is *Cooltown* [6]. This project introduced a new interaction approach by using HTTP GET and POST requests to manipulate things. Then, with the progress made in embedded systems, it was possible to integrate Web servers directly on sensors and actuators. So-called *mashups* were introduced in the *WebPlug* framework relying on the Web-of-Things paradigm, where sensors and actuators play a central role [7].

Due to the intrinsic architecture of things leveraging on Web services, problems related to performance and memory usage were quickly discovered. Another source of problems is related to the energy consumption of things that needs to be minimised. In other words, optimisations are sought at every level when dealing with things. Some works have started tackling optimisations at the communication layers [8]. Other works have proposed to simplify the way things expose their services through the web using RESTful APIs instead of heavyweight SOAP protocol [9], [10], [11].

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* [12], which consists of a script generating C++ classes representing devices capabilities. A more Web oriented approach has been realized in [13]. 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* [14] project is a BMS system offering interfaces to various building networks, including KNX. 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 WoT's simplicity and by being highly integrable in existing KNX infrastructures.

III. The WoT framework

The Web-of-Things framework fills the gap left by the Internet-of-Things regarding the application layer [15]. The IoT actually only covers the IP connectivity of everyday objects, resolving problems linked to the Internet access and network topologies. The WoT is completing the paradigm by leveraging on well-accepted standards of the Web to build Application Programming Interfaces (APIs) to things. In this framework, things are representing resources identified using URLs and manageable using the verbs of the HTTP protocol to form the so-called RESTful APIs. Based on the widespread standards of Web applications, the WoT approach has good potentials to become the de facto standard to communicate and program small and constrained interconnected objects. In this chapter we will provide insights of the main concepts underlying the WoT.

A. Resource identification

In the WoT, every capability or property of a device is considered as a resource. For example, a temperature sensor could return the measured value both in Celsius and in Fahrenheit. This would give us two resources that can be read. More precisely, some resources can allow multiple operations as read and write. So, we first need to be able to identify and address those resources in a simple way before we can interact with them. This is realized by using *URLs* in the same way as for retrieving Web pages on servers. An advantage of this approach is in its hierarchical way to organize resources reflecting the physical world. This principle is shown in figure 2. For accessing the Celsius temperature value, one would use the following URL: *http://<DOMAIN>:<PORT>/genericnodes/1/sensors/temperature/celsius*.

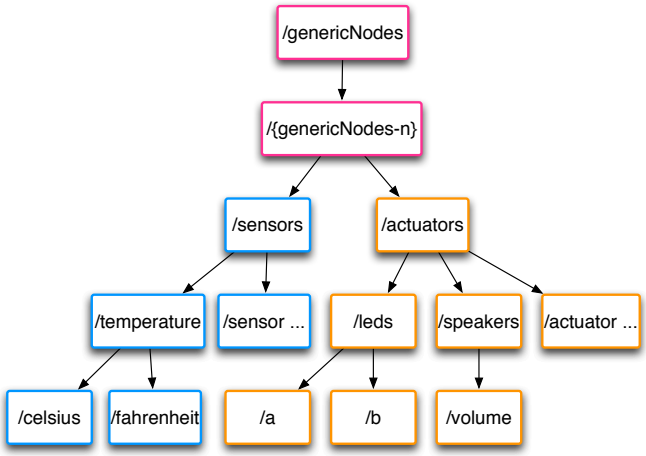

Fig. 2. Resource hierarchy example of an abstract node

The domain part of the URL also allows to be hierarchically structured to match a virtual abstract structure or a real physical organization. In the case of buildings, URLs can give insights on the location of devices inside the organization by decomposing it into sub-domains according to buildings, floors and rooms.

B. RESTful APIs

RESTful APIs are really the communication and application layers in the WoT by leveraging the HTTP protocol. Unlike SOAP, HTTP is used as application protocol and not only for transport. REST has several advantages over SOAP by having less overhead and being resource oriented, which fits naturally with physical objects. With the WoT paradigm, every object or thing is embedding a Web server exposing an API for acting with its sensing, actuating and configuration capabilities. Those services are located through the URLs as explained previously. The interaction with the resources is achieved by sending HTTP requests containing a so-called *HTTP verb* that can be one as follows: GET, POST, PUT and DELETE. These verbs reflect actions that can be performed on resources. The GET is for retrieving information, POST to modify information, PUT to add information, and DELETE for removing information on a resource. The GET and POST verbs are in the context of WoT the most used ones. For example,

one will use the GET verb to read a sensor's value, and the POST one for actuating a relay.

A HTTP communication always consist of a request to a specific resource and a response, this for any kind of operation. HTTP responses contain in the header a code value expressing errors, exceptions and successes. Each code has a well-known meaning listed in the HTTP 1.1 specifications.

C. Events notifications

In many scenarios systems want to be informed as soon as something happen on a monitored resource. Instead of using a polling technique that is resource consuming, the WoT relies on *callbacks*. In an event-based system, the first step is the registration of the consumer at the producer. Working with things embedding a REST Web server, we can expand the API with methods dedicated to registration. A system interested to be notified about a change of state on a resource will announce itself by providing the callback, instanciated by a URL representing a REST service on the consumer side. We demonstrate this mechanism with a simple example involving a BMS and a motion detector as illustrated in figure 3.

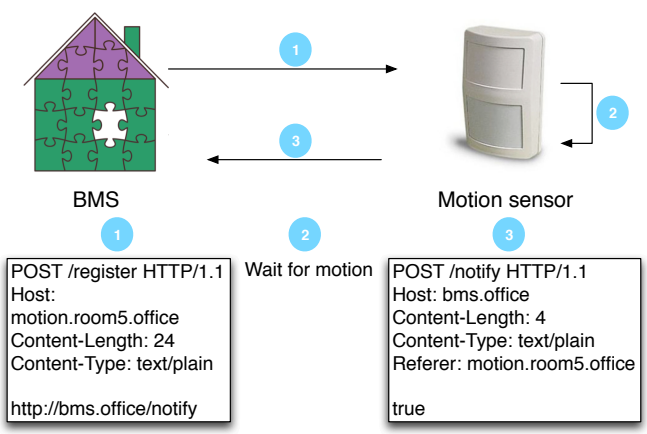

Fig. 3. Event notification mechanism with (1) consumer registration step with callback notification, (2) producer value monitoring, (3) producer notification to the consumer.

1 - The BMS will register at the motion sensor in order to be notified when someone enters or leaves a room. This is achieved by the BMS sending a HTTP POST request to *http://motion.room5.office/register* containing the callback URL. **2 -** The producer will then internally watch its resource. **3 -** Every time the value changes the motion sensor will do a HTTP POST request to the callback URL. The steps 2 and 3 are repeated until the consumer unregisters.

IV. From KNX to RESTful API

As previously outlined with WoT paradigms, every object is expected to embed a REST server offering an API located through URLs for interaction. Unfortunately this approach can not be applied as such to a KNX network. Devices connected to the KNX network have no IP address and therefore will not be accessible by using URLs. In addition to this, KNX devices are very constrained and task oriented, which makes it impossible for most of them to embed a Web server.

A way of filling this gap is to propose a gateway exposing devices functionalities in the form of RESTful APIs. The gateway will hide the complexity of the KNX network and allow clients to interact with KNX 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. Clients will therefore be able to retrieve information from a KNX network (for example read a temperature value) or to influence the behaviour of the installation (for example to move blinds).

Though the gateway approach is seducing, open questions remain about the automatic mapping of devices functionalities to URLs and RESTful services. In the next sections, we propose a discovery approach allowing clients to identify which devices are accessible by using RESTful services and what are their capabilities.

A. KNX application layer

The KNX application layer, also known as *KNX interworking*, was thought to ensure interoperability between devices of various manufacturers. It standardizes the way how payload data inside telegrams have to be structured and interpreted. Like many systems dedicated to automation, the interworking is based on so-called functional blocks to describe system functionality [16]. Logical parts of a device, such as a specific function are symbolized by those Functional Blocks (FBs). We can illustrate this principle with an example of a light switch FB that is a logical function of a four channels relay. A functional block is always attached only to one device.

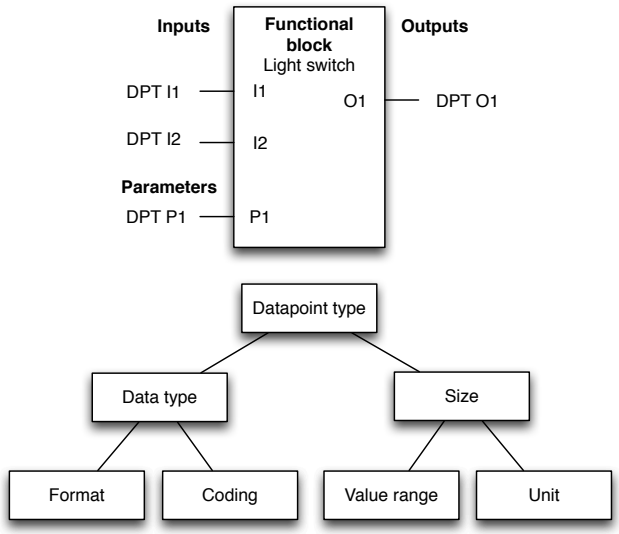

Fig. 4. Functional block structure and datapoint type composition

As visible in the upper part of figure 4, FBs are composed of a set of datapoints (DPs). Those datapoints are communication endpoints of devices allowing access to the functions of a block [13]. 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. KNX administrators simply link

outputs and inputs together for associating devices. Datapoints are also standardized in terms of syntax and semantics as visible in the lower part of figure 4, and 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.

B. ETS export archive

The KNX association has developed the *Engineering Tool Software* (ETS) to configure 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 5, 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 device file informs about the available DPs on the device. This archive, being zipped without security, containing XML files easily understandable allows to import all the network knowledge into other applications. As illustrated in figure 5, we propose here to apply a XSL transformation to the different XML files inside the archive in order to centralize all necessary information for our gateway into one single XML file.

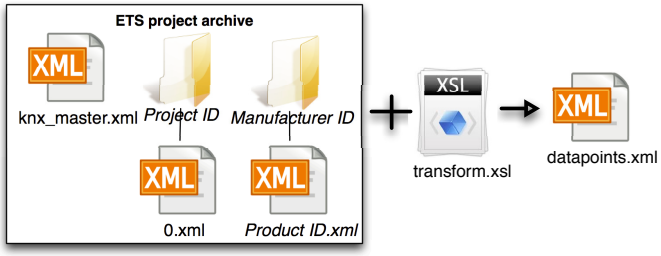

Fig. 5. ETS project archive structure

C. Datapoints to REST

As previously explained, access to functionalities in a KNX network is done via group objects, which are compositions of datapoints and a group address. In the elaboration of the gateway, we match group objects to REST services for allowing interaction with KNX devices. For doing this, we use the XSLT output that was created from the XSL transformation. By taking into consideration the corresponding fields of the XML, we are able to compose a URL identifying a specific group object. For example, the datapoint shown in listing 1 would result in the following URL: *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>
```

Our objective is here to allow the BMS to pull state values and to perform actions on the KNX network by sending HTTP requests to the gateway. A HTTP GET request will result in the HTTP response containing the actual value of the group object inside the payload data. For changing a state, one will send a HTTP POST request containing the new value inside the payload data. 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. For example, the DNS equivalence of the group *heating* located in room *office005* of the *ground* floor in the *leso* building, giving the DNS entry *heating.office005.ground.leso* has to redirect requests to the gateway.

D. Events

To avoid control systems being forced to implement a polling strategy for observing changes of states, our gateway offers a notification mechanism allowing to observe every group object. Here, the events notifications principle of the Web-of-Things is applied. Every URL identifying a group object is extended with two sub-resources for registration and unregistration. The *register* and *unregister* key-words are placed after the datapoint type as sub-resource. In our previous example, the URLs for registration and unregistration will be as follows: *http://heating.office005.ground.leso.epfl.ch/dpt_switch/[un]register*. The gateway holds internally a list of all the consumers registered for every group object. When a change of state of a group object occurs, the gateway checks its list of listeners, and perform the notification through the callback of the consumers.

E. Discovery

Before communicating with KNX devices, a system has first to discover what group objects are available on the gateway. A very simple manner would be to provide a single list informing about all group objects. Such an approach is obviously not well adapted due to its poor structure and the need to transmit the whole list for every discovery request. Here, we propose to expand our concept of building-composition structured DNS. In addition of keeping entries for groups, it does also have entries for the sub-domains composing the URLs. For example, it will store entries like *ground.leso.epfl.ch*. By calling *http://ground.leso.epfl.ch*, the DNS server redirects the request to the gateway. The gateway will perform a lookup in the XML file to find all children and will respond with a JSON structured message.

Once a group has been located, the available datapoints can be known by adding the placeholder * instead of the datapoint identifier. The gateway will answer with a JSON payload describing the datapoints one can interact with. An example of this procedure is given in listing 2. The resulting URL structure is as follows: http://<GROUP_NAME>.<LOCATION>.<ORGANIZATION_DOMAIN>/*.

Listing 2. JSON message structure example of a datapoint description for http://heating.office005.ground.leso.epfl.ch/*

```
{"datapoint_info":"1-bit",
"datapoint_type":"DPT_Switch",
"description":"on/off",
"bits_size":1,
"datapoint_number":"1.001",
"url":"http://heating.office005.
ground.leso.epfl.ch/dpt_switch"}
```

F. Storage

BMS are nowadays evolving from simple reactive to proactive controls where the user behaviour and habits play a key role. Proactive BMS use past data to build more and more complex mathematical models of the building's use, for example to anticipate the control of heating or to provide energy consumption feedback [17], [18]. A possibility would be to let the BMS store all sensor data into a local database. The BMS would then have to register callbacks on all sensors. This would result in a lot of event notifications, eventually causing some congestions. Therefore, we propose that our gateway acts as decentralized storage.

We introduce here the concept of storage registration, where the BMS announce their interest of storing some sensor data on the gateway. This is achieved by the client calling the *.../storage/add* sub-resource of a group object with a HTTP POST request. This request must contain the *Referer* header used for identifying the client. The maximum number of days data have to be stored has to be provided inside the payload as parameter. Once having a client registered for storage, the gateway will start to store the measurements for this datapoint. When the client does not need the storage anymore, it can unregister by POSTing to *.../storage/remove*, providing its ID in the *Referer* header. For retrieving the data, one has to make GET requests with some URL parameters delimiting the range of data. In our design, we propose two ways of filtering the data. The first is by indicating the number of days one wants to go back in the history with the URL: *.../storage?days=X*. The second one is by specifying a period of time with a start and end date as follows: *.../storage?from=X&to=Y*. In both cases the gateway will respond with a JSON array containing the stored data, composed of values and timestamps. An example is given in listing 3.

Listing 3. JSON message structure example of a datapoint storage for http://lighting.office005.ground.leso.epfl.ch/dpt_switch

```
{
"storage": [
{ "value":"on",
"timestamp":"2013-01-10 08:12:34" },
{ "value":"off",
"timestamp":"2013-01-10 09:05:57" },
{ "value":"on",
"timestamp":"2013-01-10 13:40:03" },
{ "value":"off",
"timestamp":"2013-01-10 17:33:11" }]
}
```

As the gateway has also a limit to its storage space, we propose to include rules for the management of the data history:

- In the case of multiple clients registered to the same group object with different maximum days to be stored, the gateway will store data according to the highest value.
- In the case of multiple clients registered to the same group object with different maximum days to be stored, when the client with the highest maximum days unregisters, the data pool is shrunk to the new highest one.
- When an unique client of a group object unregisters, the stored data for this group object will be dropped.
- When the stored data of a group object is not read during three times the maximum number of days of this group object (maximal days value between all clients of the GO), the data is dropped and all clients of the group object are automatically unregistered.

Our storage concept brings many advantages compared to classical "logger" approaches where usually everything is stored using a circular buffer. First, we only store data that are effectively needed by the BMS. Clients can choose what kind of data they need and for how long it has to be available. Second, the rules listed before allow using small storage media like SD cards on constrained devices.

V. Implementation

We provide here more details on a practical implementation of the gateway according to the principles exposed in Section IV. We voluntary restricted the implementation to only state based group objects which are used by BMS.

A. Platform

We selected as hardware platform the *Raspberry Pi model B* which is low-cost and offers enough computing power for running our gateway [19] . This tiny micro-computer (85.6mm

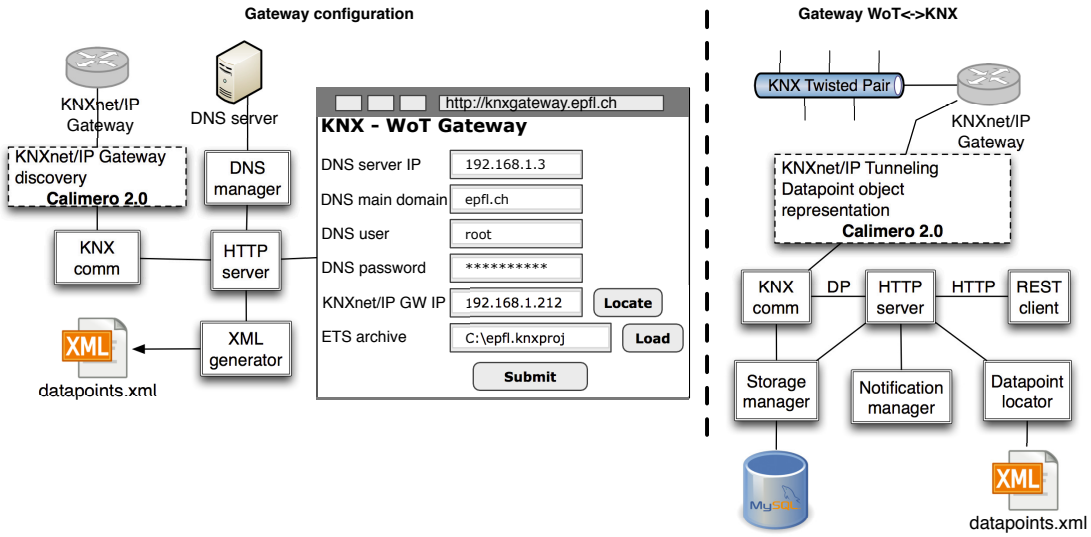

Fig. 6. Overall KNX-WoT gateway architecture illustrating the logical modules for two scenarios: gateway configuration (left part) and gateway normal operation (right part).

x 56mm x 21mm) embeds an ARM1176JZFS CPU running at 700MHz with 512MB of RAM. The model B offers an Ethernet connectivity with an RJ45 port. It is composed of a SD card reader, two USB ports, one HDMI video port and a RCA video output. This module can be considered as low-power with about 4W consumption, powered at 5V over a micro USB port. The Raspberry Pi can be operated by many Linux systems installed on a SD card plugged into the device. Any kind of software compatible with the ARM processor can be installed on it, like Java, MySQL and many others.

B. Architecture

Our implementation relies on the *Calimero 2.0* Java library [20]. 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. The Web part of our implementation is composed of a Java servlet running on a *Jetty* server [21]. We opted for Jetty instead of other common Java Web servers like Tomcat, Glassfish, JBoss or Grizzly because of Jetty being easily embeddable on low-resources hardware thanks to its lightweight implementation. The database is running on MySQL. All components of the implementation are open source and free.

As shown in figure 6, we based our implementation on several logical modules shared in different scenarios of use. The first one is the configuration of the gateway, where the administrator will provide all the necessary information for proper running. Once configured, the gateway enters in its normal operation where it can serve requests for manipulating group objects. We provide here more details about the role of the different logical modules.

The **Datapoints file** acts as database even if it only consists of XML tags. This file is at the heart of the application and holds all the mandatory information for communication with the KNX devices. It contains a description of all group objects reachable on the network, indicating the datapoint type, the group address, and other kind data about the group object. This file also stores configuration data provided through the Web configuration page.

The **Web server** stands as entry point of the application. It implements the *doGet()* and *doPost()* methods for handling the HTTP requests. The first step is to decode the URL in order to identify which action is requested on which group object, as it could be a direct interaction or a notification registration message. Once the operation identified, it acts as controller and dispatches the request to the right modules. At the end of the processing, it will respond either with a value corresponding to the GET read request, or only with the HTTP response code in the case of a POST.

The **XML generator** processes the ETS project archive for generating the XML datapoints file. It first decompresses the archive and then applies the XSL stylesheet to the project. All special characters are removed during this processing as they can not be present in URLs. The XML file is saved inside the resource directory of the Web server.

The **DNS manager** is responsible of adding DNS record entries representing groups. This is performed with the DNSJava library offering methods for managing zones and records of a DNS server [22].

The **Datapoint locator** acts as query engine for the datapoints file. It can look up specific group objects according to the datapoint type, group name and location, and then returns them in the Calimero datapoint object representation. It is also used for retrieving all the possible domain names for accessing the group objects, used by the DNS manager for adding entries on the DNS server. In the case of a client willing to discover the available datapoints, the Datapoint locator can answer with all sub-domains or with all datapoints descriptions of a group.

The **KNX comm** represents an interface to the KNXnet/IP network. This module can discover KNXnet/IP gateways by sending multicast messages, thus avoiding administrators to look after the IP address of the gateway. This feature is useful

when DHCP is used to configure the IP addresses. The KNX comm module handles the tunnel connection, allowing to talk with the KNX network. By listening to the network, it will notify the Events manager of incoming telegrams that might concern some consumers. Activating a cache feature can avoid to overload the KNX network due to many clients reading the same group object.

The **Storage manager** is the interface to the database that offers methods for handling clients (un)registrations and for retrieving group object data. Another role of this module is to implement the storage rules as explained above, for example shrinking the database when some data are not anymore used. This allows to store data for many group objects even having only a few GB of space.

For managing the notification paradigm, we introduce an **Events manager**. This module stores in an associative table the consumers registered on group objects for notifications. Triggered by the KNX comm module, it will lookup if consumers are registered for the group object having undergo a change of state, and then launches the notification by calling all related callbacks.

VI. Evaluation

We evaluated the capabilities and limitations of the implemented gateway through several tests. In order to have realistic feedbacks, we performed our tests on a building already equipped with a KNX installation. From the evaluation results, we also discuss some improvements of the gateway that would be beneficial for BMS.

A. Performance

To establish the performances of our gateway, we decided to measure various key-values such as: maximum number of requests per second, maximum simultaneous requests, notification reaction time (from the action on the KNX device until producer notification) and processing time of the ETS project archive (during configuration). All our measurements are done with an existing KNX installation of the 4 floors office LESO building located on the EPFL campus in Lausanne, Switzerland. The installation features 265 devices, distributed in 765 groups, with a total of 795 group objects and represents an average installation that can be found in many 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 I. Gateway performance measured on a real-life KNX installation running 265 devices

B. Discussion

Table I summarises the performance of the gateway implemented on a Raspberry Pi as described in Section V. The ETS archive processing time is quite long, mainly due to the XSLT that is extremely resource consuming. 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. We can see that all the results are suitable for such an installation and that the Raspberry Pi has to be considered as an alternative to classical PCs running gateways or serving as middleware.

A potential limitation of our proposition lies in the DNS approach which implies access to the DNS server of the host IP network. Such access may be restricted by security policies in which case a dedicated DNS server has to be made available for the gateway. A second issue is related to the security of our gateway where currently no authentication is implemented. An authentication layer based on access lists could be a solution to this.

Some developers have actually built small applications interacting with the KNX devices through our gateway, this in various languages. Their feedback were positive, showing the benefits of leveraging on standardized and well-accepted protocols to reduce the integration time of KNX devices on a BMS.

Some developers have also asked us to provide a mechanism of actions scheduling in order to implement some behaviours of proactive BMS and to deport the logic on the gateway. This idiom would lead to distributed BMS, where the heavy modelling tasks would run somewhere in the cloud and bring back the logical control on the gateways.

VII. Conclusions

Inspired by Web-of-Things paradigms, we explored the feasibility and benefits of using well-known web standards like HTTP and RESTful APIs to interface KNX networks and building management systems. We proposed an architecture for a KNX-WoT gateway that has been validated through an implementation on a low-cost Raspberry Pi and validated on a realistic KNX configuration. Positive feedbacks were also returned by developers of building management systems thanks to the simplicity of use of WoT APIs.

Generally speaking, we believe that WoT approaches are good candidates to facilitate the integration of heterogeneous networks. We also believe that building management systems will have to dialog with various networks in a near future as new technologies are emerging, such as for example Enocean.

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. Finally, we will investigate the feasibility of building a direct connection to the KNX twisted pair in the gateway to eliminate the need of the KNXnet/IP module.

The software running on the gateway can be made available for any scientific research project upon request to the authors.

Acknowledgment

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. The authors would also like to thank Sébastien Baudin, scientific collaborator at the University of Applied Sciences of Western Switzerland for its contribution to the development of the gateway.

REFERENCES

[1] KonnexAssociation, "Knx specification," 2004.

[2] E. Alliance, "Enocean technology – energy harvesting wireless," http://www.enocean.com/fileadmin/redaktion/pdf/white_paper/WP_EnOcean_Technology_en_Jul11.pdf, July 2011.

[3] G. Bovet and J. Hennebert, "The web-of-things conquering smart buildings," vol. 10s/2012. ElectroSuisse, 2012, pp. 15–19.

[4] W3C, "Soap version 1.2," http://www.w3.org/TR/soap12-part1/, April 2007.

[5] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the internet of things to the web of things : Resource oriented architecture and best practices," in *Architecting the Internet of Things*, D. Uckelmann, M. Harrison, and F. Michahelles, Eds. Springer Berlin Heidelberg, 2011, pp. 97–129.

[6] T. Kindberg and al., "People, places, things: Web presence for the real world," in *Mobile Networks and Applications*, vol. 7. Building, 2002, pp. 365–376.

[7] B. Ostermaier, F. Schlup, and K. Römer, "Webplug: A framework for the web of things," in *Proc. of the First IEEE International Workshop on the Web of Things (WOT2010)*, Mannheim, Germany, 2010.

[8] G. Bovet and J. Hennebert, "Communicating With Things - An Energy Consumption Analysis," in *Pervasive*, Newcastle, UK, June 2012, pp. 1–4.

[9] F. Aijaz, M. Chaudhary, and B. Walke, "Performance comparison of a soap and rest mobile web server," in *Proc. of the Third International Conference on Open-Source Systems and Technologies (ICOSST 2009)*, 2009.

[10] H. Hamad, M. Saad, and R. Abed, "Performance evaluation of restful web services," in *Computer Engineering*, vol. 1. Computer Engineering Department, 2010, pp. 72–78.

[11] C. Groba and S. Clarke, "Web services on embedded systems – a performance study," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops PERCOM Workshops*, vol. 3. IEEE, 2011, pp. 726–731.

[12] W. Kastner, G. Neugschwandtner, and M. Kögler, "An open approach to eib/knx software development," in *Fieldbus Systems and their Applications*, 2005, pp. 255–262.

[13] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner, "Web services in building automation: Mapping knx to obix," in *Proc. of the 5th IEEE International Conference on Industrial Informatics*, vol. 1, 2007, pp. 87–92.

[14] "openhab," http://code.google.com/p/openhab/, April 2013.

[15] D. Guinard, "A web of things application architecture – integrating the real-world into the web," Ph.D. dissertation, ETHZ, 2011.

[16] *KNX Advanced Course Specification*, February 2012 ed. KNX Association, 2012.

[17] N. Morel, M. Bauer, M. El-Khoury, and J. Krauss, "Neurobat, a predictive and adaptive heating control system using artificial neural networks," *International Journal of Solar Energy*, vol. 21, no. 2-3, pp. 161–201, 2001.

[18] C. Gisler, G. Barchi, G. Bovet, E. Mugellini, and J. Hennebert, "Demonstration Of A Monitoring Lamp To Visualize The Energy Consumption In Houses," in *The 10th International Conference on Pervasive Computing (Pervasive2012)*, Newcastle, jun 2012.

[19] "Rapberry pi," http://www.raspberrypi.org/, January 2013.

[20] "Calimero 2.0," http://calimero.sourceforge.net/, January 2013.

[21] "Jetty server," http://jetty.codehaus.org/jetty/, January 2013.

[22] "Dnsjava library," http://www.xbill.org/dnsjava/, January 2013.

Gérôme Bovet (M'12) was born in Fribourg, Switzerland, in 1985. He received the engineer degree in computer science from the College of engineering in Fribourg Switzerland, in 2008 and a M.S degree in engineering of information systems from the University of Applied Sciences of Western Switzerland in 2012. He is currently pursuing the Ph.D. degree in computer science at Telecom ParisTech, Paris, France.

From 2008 to 2012, he was working in the field of LBS, geolocation and mobile computing for two Swiss companies. Since 2011, he is also lecturer at the College of engineering in Fribourg, Switzerland. His research interest includes architecting the Web-of-Things for smart buildings, the development of smart gateways, and low power sensor networks.

Mr. Bovet's awards include the Swiss Engineering prize 2008 for his engineer degree final work, and the third prize of the European Satellite Navigation contest 2009.

Jean Hennebert (M'06) Jean Hennebert received the Electrical Engineering degree from the Faculté Polytechnique de Mons, Hainaut, Belgium, in 1993 and the Ph.D. degree in computer science from the Swiss Federal Institute of Technology, Lausanne, Switzerland, in 1998.

He then worked for six years as IT System Architect and Independent Consultant for different private companies. In 2004, he was with the multimedia engineering DIVA group of the Computer Science Department, University of Fribourg, Fribourg, Switzerland, where he is in charge of teaching and research activities. Since 2007, he is Professor with the University of Applied Sciences Western Switzerland – HES-SO and is affiliated as Lecturer with the University of Fribourg. His research interests are in the areas of telecommunication, signal processing and machine learning where IoT and WoT technologies are enablers.