Plug and Play Building Monitoring: The potential of low cost components

Reinhard Zach, Alexander Paul, Robert Zach, and Ardeshir Mahdavi
Department of Building Physics and Building Ecology, Vienna University of Technology, Austria

ABSTRACT: The integration of building monitoring into existing buildings can be an expensive process and may require inefficient installation efforts, since such existing buildings were not designed for this purpose. This paper presents a specific approach to address this challenge. Thereby, low cost hardware components are deployed to collect and process monitored data from multiple building zones. Moreover, energy-harvesting wireless sensor technology is applied.

1 INTRODUCTION

Given the continuous increase in energy costs, energy efficiency measures gain on importance. Specifically, building monitoring can be utilized to improve buildings' energy efficiency. However, most existing buildings are not equipped with the required technologies. Thus, the required infrastructure must be installed later, leading to high economical investments. This highlights the importance of a more efficient approach to tackle the problem.

1.1 *Approach*

The proposed gateway software runs on the Raspberry Pi 2014 and collects and processes measurements of all nearby EnOcean 2014 sensors. It is connected to a remote monitoring server through Internet based technologies (local Ethernet network, mobile telecommunication technologies – UMTS, etc.). The prototype setup uses the Monitoring System Toolkit on the server side (MOST 2014). However, any other BMS could be also used. The software is written in JAVA and is therefore executable on every device, which can run a Java Virtual Machine. The Linux based and credit-card-sized single-board computer Raspberry Pi is low-cost, has enough resources for our application, and is energy efficient (with a power of about 2 Watts). As a sensor and fieldbus technology, EnOcean is used because of the following reasons:

- it offers a wide range of products.
- is wireless, which makes it easy to install in existing environments.
- most of the sensors use energy harvesting mechanisms, and do not need an external power source, which again makes it easy to install them. This also makes their maintenance intervals longer.

These attributes suggest that EnOcean can be considered as one of the possible technologies for our application.

2 HARDWARE ARCHITECTURE

2.1 *Raspberry Pi*

The Raspberry Pi, as shown in Figure 1 (2), is a credit-card-sized ARM computer, which can run the Linux based OS "soft-float Debian wheezy".

In the proposed setup the Raspberry Pi runs the gateway software, which receives measurements

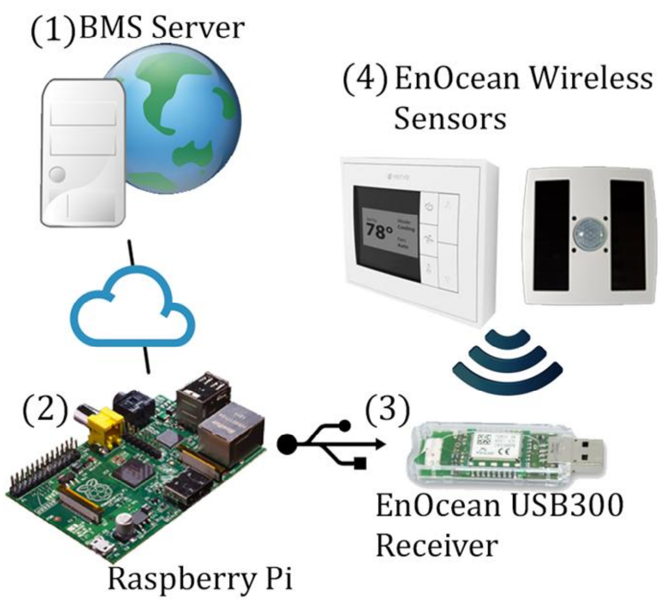

Figure 1: Hardware Architecture

from the sensors and sends them to the building monitoring software in the required format. The software caches measurements, in case the connection to the monitoring server is interrupted.

2.2 *EnOcean USB 300*

The EnOcean USB 300, as shown in Figure 1 (3), is a small USB stick that connects the Raspberry Pi to the EnOcean Sensors. It receives the sensors' radio signals and forwards the data via USB port to the gateway software.

2.3 *EnOcean Sensors*

The EnOcean Sensors, as shown in Figure 1 (4), use a wireless radio protocol called EnOcean Radio Protocol (ERP) for data transmission to the USB 300. They are characterized by the very low energy use. For example, a battery free switch uses not more than 50 µWs to send a message. This gives them the ability to run on energy harvesting mechanisms (e.g. solar panels) removing the need for a power cable. This makes them very easy to install.

The drawback of the focus on the low energy use is that the ERP is kept very minimalistic. Furthermore it varies between different sensor types and is thus difficult to process. Typically such a packet is only a few bytes in size (e.g. a typical telegram from a temperature sensor in hexadecimal format: "a5 ff 02 99 08 00 05 72 7a 00"). The sending process itself takes usually about 40ms. In this timespan the sender sends the packet three times with a random interval. A packet itself is sent in 1ms. This strategy is used by EnOcean to avoid collisions.

2.4 *BMS Server*

The BMS runs on an external server, as shown in Figure 1 (1). We use the aforementioned Monitoring System Toolkit (MOST).

3 SOFTWARE ARCHITECTURE

The software architecture is divided in two seperate layers; the connector layer, as shown in Figure 2 (3), and the gateway layer, as shown in Figure 2 (1).

The connector layer manages the connection to the building monitoring software MOST, whereas the gateway layer handles the processing of the received sensor data. They communicate through an interface, which is called Callback interface, as shown in Figure 2 (2). The gateway layer is not aware of the connector layer. Hence the connector could be easily replaced through another building monitoring solution, instead of MOST.

Furthermore the gateway layer constantly listens to the COM-Port where the ERP data of the EnOcean USB 300 is received.

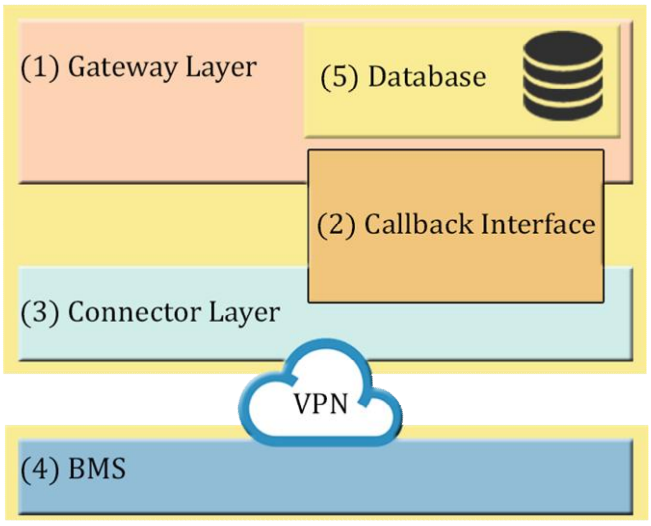

Figure 2: Software Architecture

3.1 *Gateway Layer*

This is the core application module of the gateway. It processes the received byte stream of the COM-Port, caches the measurement and forwards it through a callback interface to the MOST connector.

The COM-Port provides the packet based on the EnOcean Serial Protocol (ESP). The ERP with the sensor data is encapsulated within the ESP packet.

The ESP contains a checksum, which is used to verify if the data was correctly transmitted.

The EnOcean Equipment Profiles (EEP) are required for the correct processing of the ERP packets, as they give information about the structure of the received byte stream.

After a packet is processed, the data is stored in a lightweight Derby database. In this way, the measurements are cached in case of connection loss.

3.1.1 *COM-Port*

The EnOcean USB 300 is accessible via a virtual COM-Port. The RXTX Library is used, since it offers a vast collection of methods for accessing serial and parallel ports. It delivers a byte stream which contains the received data of the EnOcean USB 300, and hence the measurements of the sensors.

Since Linux offers by default a FTDI COM-Port driver, no further installations are required. Only the path to the USB device needs to be configured to /dev/ttyUSB0.

3.1.2 *Packet Receiving*

The byte stream from the virtual COM-Port is read by a listener thread waiting for new data to arrive. The EnOcean USB 300 encapsulates the ERP-packet into the ESP-packet in view of fault tolerance and assignability.

The ESP packet has a certain structure that has to be taken into account:

- The first byte called Sync Byte signals the beginning of a new ESP-packet with the value 0x55.
- It is followed by the header with a fixed length of four bytes. Whereas the first two bytes define the length of the data section, the third specifies the length of the optional data and the fourth the type of the packet. The gateway processes only type RADIO telegrams.
- The next byte includes a checksum of the header that is verified.
- With the seventh byte the data section begins with the length specified in the header. Here the ERP-Packet can be found, which includes the measurements.
- The optional data section starts after the data section. Its length is also defined in the header.
- Finally, a checksum byte for the two data fields begins, with is used to verify the packet.

The gateway receives the ESP-packet from the COM port, verifies it through the checksum, and then starts the actual processing of the data section and hence the ERP-packet.

3.1.3 *Packet Processing*

The processing of the ERP-packets turned out to be rather complicated due to the structure and its lack of uniformity. Each sensor type has its own packet format, which differs strongly in both length and interpretation of the data. This packet format is called EnOcean Equipment Profile (EEP). In the current Version 2.6 there are more than a hundred different EEPs defined, and the number is increasing, since new sensors, with new requirements, are constantly published.

Therefore this logic is transferred to the outside of the application source code, since every new EEP would require a modification of the source code, and hence a recompilation of the software.

An XML file is used to define a profile for each kind of EEP. The file only needs to be placed in the folder XML in the application directory and will get deserialized on startup of the gateway. For this purpose the SimpleXML framework is used. With this method flexibility is maximized while minimizing the efforts for supporting new sensor types (EEP types).

Since the ERP has no information at all which sensor type it is, it is necessary to link the Sender ID of the sensor with the EEP. This is done in the building monitoring software. The connector layer hands the mapping to the gateway layer through the Call Back Interface.

3.2 *Data Persistence*

To tackle the problem of potential connection loss to the server, be it a server update or an unplugged Ethernet cable, a local database is used to retain the measurements. This redundancy has a major advantage since the system runs completely stable and autonomous from the actual building monitoring server.

Due to performance aspects the lightweight database Apache Derby 2014 is used. It is written in JAVA, which enables direct integration into the gateway software. Comparable databases are written in C native code.

Furthermore, the Derby database is used in Embedded Mode, which means that it is executed within the gateways process, but in a separate thread, as shown in Figure 2 (5). This reduces the overhead that would elsewise occur, since the process scheduler gets relieved.

The database itself creates its DBMS and storage within the application folder during the first execution. Every received packet is stored in the database directly after successful processing. The data is then ready to be sent to the building monitoring system.

The database is designed with a fixed table row count. This means that when the predefined maximum measurements are reached, the oldest ones are removed. The removal of the old data is handled by a separate cleaner thread, which checks periodically whether the cache is full. Both, the maximum row count and the interval of the cleaner check can be configured in the gateways configuration file.

According to tests, 10.000.000 measurements have a size of about 10MB in the database. Still no performance issues were noticeable, except for the cleaner job that takes a few seconds more to finish, compared to 10.000 values.

The entire ERP-packet is stored with a reception timestamp. When the data is requested by the BMS we process it again to generate a new interpretation of the previous received bytes. This has the advantage that the data uses always the newest settings for the interpretation of the bytes. This allows latter adaptions of the XML files with the definition of the EEP, or the change of the configured EEP for the sensor on the server.

According to our tests, the reprocessing of 10.000 packets takes about 1.5s. A usual request is normally for about 10-50 packets, and since real time requirements are satisfied, the advantages outweigh the disadvantages. Nevertheless during the tests we noticed that the OpenJDK-JVM was processing significantly slower, while the Oracle-JVM was around seven times faster. To access the stored data from the connector, the Callback Interface can be used.

3.3 *Callback Interface*

To detach from the BMS specific implementation, the gateway and the actual connection to the BMS is separated. Since the connector layer and the gateway layer need to communicate, the Callback Interface is designed, as shown in Figure 2 (2). The interface provides different methods for the interaction with the gateway layer. This design decision opens up the opportunity to use the gateway layer with other BMS than MOST. The interface only needs to be implemented in a connector to the server, and it can access the methods, and hence the gateway.

The Callback Interface defines the reading of the measurements, and the mapping of the sender id to the required EEP, which has to be implemented in the connector layer, since it is BMS specific.

3.4 *Connector Layer*

The connector layer connects the gateway with the BMS, as shown in Figure 2 (3) and (4), via VPN. It has to implement the business logic required for forwarding the interpreted data of the sensors to the BMS server. Furthermore, it also needs to set the sender mapping for the gateway layer. Each data point in the BMS gets an assignment of a specific EEP, and a specific part of it (e.g. occupancy-brightness sensor: get brightness). Those values need to be handed over from the connector to the gateway layer. Sensors whose data is not handed over and have thus no sender ID to EEP mapping, cannot be processed.

3.5 *Building Monitoring Software*

This approach uses use the BMS MOST because of the following reasons:

- All components are open-source, and coded in JAVA, so we can make adaptions if necessary.
- It offers a wide range of different connectors for interacting with the BMS, starting with classic JDBC connectors to RMI connectors.
- It has a descriptive and vivid web view of the measurements. There is also the possibility to display the data within a 3D building model.

4 TEST SETUP

To verify the implementation two independent test series are used.

4.1 *Test Bed*

This test takes place in the test bed of the Department of Building Physics and Building Ecology, Vienna University of Technology. Several different sensor types are used, including window and door contacts, temperature, CO2, illumination, and humidity sensors.

During the test period of four and a half months there was only one system failure that caused the gateway to stop processing data. This was traced back to an EnOcean USB300 error caused, perhaps, by power fluctuations. After a reboot of the system, including the removal of the power supply, the EnOcean USB300 started working again. In total the gateway transmitted over 25.000 measurements from twenty different data points.

Furthermore, we noticed that dark rooms are not optimal for the mostly solar energy powered sensors. We measured that at least 14-35 lx on a regular basis are necessary for most solar powered sensors to work properly. This is not a real problem for most typical architectural spaces. It could be a problem, however, in basements or rooms with very little light. In such cases, batteries or a power cables would be required.

4.2 *BPI Office*

Unlike our test from the test bed, this test was executed in a real world office, which was already equipped with numerous EnOcean sensors, and runs proprietary OPC Bus software for forwarding data to the BMS. This has the advantage that we have a reference system we can compare the measurements with.

Within one month the gateway forwarded over 120.000 measurements from about 250 data points to the BMS, without any incident so far. We noticed though that the received packets occasionally differ from each other, meaning that one system receives a packet that the other does not: Since radio transmission is used, packets can get lost or collide with each other. EnOcean itself gives a transmission reliability of 99.99% with hundred senders sending once per minute. This problem though is negligible since we are dealing with periodic measurements, and the packet loss happens infrequently.

5 CONCLUSION AND FUTURE OUTLOOK

Notwithstanding that the current implementation is optimized for the EnOcean sensor technology, it is proposed to add multiple other solutions as well. Furthermore, it is planned to add bidirectional communication and thereby enable the BMS to control device actuators.

The combination of MOST, the gateway on the Raspberry Pi, and the EnOcean sensor technology is working fine for reading purposes. At this point of time, the implementation of the Callback Interface allows only reading access of the sensors, although the gateway software could also send data to devices.

Currently, the gateway implementation does not support the EnOcean Telegram Type VLD, and hence SmartAcks. Nevertheless this only concerns very few sensor types.
Another drawback is that the sensors have to be registered beforehand in the BMS to be interpreted, because the received packet data cannot be interpreted without the knowledge of the type.

We can conclude that the proposed solution is economical and easy-to-use, even though there are some initial problems. By enabling a "plug-and-play" setup and communication to a BMS via Internet based technologies (e.g. local Ethernet network, mobile telecommunication technologies – UMTS, etc.), a scalable network of gateways could be set up, thus paving the way for future large-scale (e.g. urban monitoring) applications. The developed source code is integrated in the MOST module *most-connector* and available at MOST 2014.

6 ACKNOWLEDGEMENT

The research presented in this paper is supported by funds from the "Klima- und Energiefonds" within the program "Neue Energien 2020".

7 REFERENCES

Alexander P., Zach R. FH Technikum Wien 2013. *Raspberry PI als Universal Gateway* Vienna, Austria

Anders A., EnOcean GmbH 2011 *EnOcean Technology – Energy Harvesting Wireless (PDF)* July 2011

Apache Derby 2014. Relational database implemented entirely in Java. *http://db.apache.org/derby/* December 2013

EnOcean 2014. Fieldbus network optimized for low power use. *http://www.enocean.com*

EnOcean Alliance 2013. EnOcean Equipment Profiles V2.6 December 2013

MOST 2014. Open-source Build Monitoring Toolkit. *http://most.bpi.tuwien.ac.at*

Raspberry Pi 2014. Low Cost Embedded System running Linux OS. *http://www.raspberrypi.org*

SimpleXML 2014. Java based XML serializer, March 2014, *http://simple.sourceforge.net/*

Zach R., Glawischnig S., Appel R., Weber J., Mahdavi A. 2012a. *Building data visualization using the open-source MOST framework and the Google Web Toolkit.* 25 – 27 July, Reykjavik, Island

Zach R., Schuss M., Bräuer R. and Mahdavi A. 2012b. *Improving building monitoring using a data preprocessing storage engine based on MySQL.* 25 – 27 July, Island

Zach R. 2012. *An open-source, vendor and technology independent toolkit for building monitoring, data preprocessing, and visualization.* Dissertation, Department of Building Physics and Building Ecology, 11.09.2012.