Supporting development and management of smart office applications: a DYAMAND case study

Jelle Nelis[1], Heleen Vandaele[1], Matthias Strobbe[1], Arnoud Koning[2], Filip De Turck[1] and Chris Develder[1]

[1]Ghent University - iMinds
Dept. of Information Technology - IBCN, Ghent, Belgium
Email: jelle.nelis@intec.ugent.be

[2]P&G
Brussels, Belgium
Email: koning.a@pg.com

***Abstract*—To realize the Internet of Things (IoT) vision, tools are needed to ease the development and deployment of practical applications. Several standard bodies, companies, and ad-hoc consortia are proposing their own solution for inter-device communication. In this context, DYnamic, Adaptive MAnagement of Networks and Devices (DYAMAND) was presented in a previous publication to solve the interoperability issues introduced by the multitude of available technologies.**

In this paper a DYAMAND case study is presented: in cooperation with a large company, a monitoring application was developed for flexible office spaces in order to reliably reorganize an office environment and give real-time feedback on the usage of meeting rooms. Three wireless sensor technologies were investigated to be used in the pilot. The solution was deployed in a "friendly user" setting at a research institute (iMinds) prior to deployment at the large company's premises. Based on the findings of both installations, requirements for an application platform supporting development and management of smart (office) applications were listed. DYAMAND was used as the basis of the implementation. Although the local management of networked devices as provided by DYAMAND enables easier development of intelligent applications, a number of remote services discussed in this paper are needed to enable reliable and up-to-date support (of new technologies).

***Index Terms*—Device Management, Device Discovery, Middleware, Office Management, Deployment, Sensor technologies**

I. Introduction

Our homes, offices, cars, and cities are increasingly becoming connected. Networked devices offer services (such as printing, environmental sensing, location tracking, etc.) that can interwork with and/or be controlled by others. Sensors are installed everywhere to measure environmental conditions as temperature, light intensity, user presence, location, and air quality [1].

This trend has triggered new ways of working as employees have become more and more time and place independent. They can work at home, in satellite offices, even in their cars in the future, and commute between different sites of the same company or clients at external locations. For example, at the headquarters of Procter & Gamble (P&G) in Belgium, around 30% of the +/- 1600 employees work at home at least once a week, 5 to 10% is travelling or in external meetings. This results in an actual average occupancy rate of the offices in Brussels of 60 to 70%. With an average facility cost of € 10,000 per employee per year, the potential cost savings by space reduction is significant. Furthermore, there is a positive impact on the company's carbon footprint by cutting down on office space and trips to and from the office.

At the same time, employees often struggle to find a working space adapted to their current needs: a place to make phone calls without interrupting colleagues, quiet places to focus on an individual task, meeting rooms, etc. In order to reduce facility costs, improving employee satisfaction and productivity, and increasing collaboration between employees, offices should be transformed into flexible spaces. In such an environment employees don't have an own desk anymore, but can choose a workplace according to their current task, such as desks, phone booths, quiet zones, huddle rooms, meeting rooms, etc. for different forms of teamwork, meetings, calls, and trainings.

End users can take advantage of real-time information about their workplace while facility managers need an aggregated view on (the history of) the actual occupation of the different spaces to make better and faster decisions with respect to the office layout. Such applications should be flexible and open to interface with different kinds of existing and future sensors, devices and systems, it should be easily customizable to the specific needs of a group of employees, and be cost and energy efficient. Currently available solutions however are typically closed and very expensive (the standard P&G room finder solution has an associated equipment cost of $1200 to $2000 per huddle room) and lack integration with existing systems.

The development of such applications that combine information from a broad range of devices and sensors in an intelligent way, is hindered by the current technology landscape which is extremely scattered with different technologies or standards prevailing in different application domains. Even in the same ecosystem, interoperability is not straightforward as the same standard can often be interpreted in a number of

different ways, hindering their adoption [2].

In order to cope with different types of sensors and communication protocols, DYnamic, Adaptive MAnagement of Networks and Devices (DYAMAND) [3] was developed. Originally, DYAMAND was designed to be deployed on a single home gateway that could communicate with all devices nearby, which is the case in a typical residential environment. However, in large-scale office environments, this assumption does not hold, and we extended DYAMAND given this newfound knowledge.

In this paper, requirements for a general application platform for supporting inter-domain and inter-technology applications are listed based on the experience gained deploying a real-life office management application within P&G. Our findings during deployment are documented in Section II and lead to a number of requirements listed in Section III. These requirements are used in Section IV to test a number of state-of-the-art platforms that promise a (partial) solution to the proposed problem.

Section V presents our solution based on DYAMAND, and discusses its large-scale deployment. Section VI focuses on the lessons learnt and presents the results of the pilot. Section VII focuses on the current status and Section VIII discusses future extensions of the solution.

II. Flexible Offices Pilot

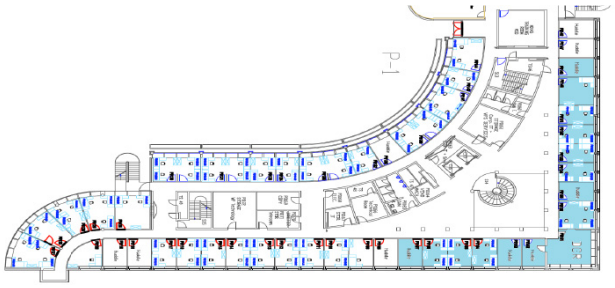

Fig. 1. Overview of the Flexible Offices pilot space at P&G in Brussels, Belgium.

In the spring of 2014 P&G in Belgium started a pilot where an area of about 1,350 square meters of office space (100 places) was redesigned (Figure 1).

To allow employees to easily find a free desk or meeting room, and to monitor the usage of the flexible offices and huddle rooms (non-reservable meeting rooms), a system was needed that automatically and in real-time monitors and visualises the occupancy of the different spaces.

In a first step, three different Commercial Off The Shelf (COTS) sensor technologies were tested in terms of accuracy, communication range, battery life, and Total Cost of Ownership (TCO):

- **EnOcean [4]**: 868.3 MHz in Europe and 315 MHz outside of Europe, typically battery-less devices that harvest energy using the environment (using light, temperature, or pressure).
- **Z-Wave [5]**: sub-GHz range using a mesh network architecture.
- **INSTEON [6]**: dual-mesh network combining a sub-GHz RF network with power line communication.

The tests were done in a number of different environments, ranging from line-of-sight connection between sensor and receiver to obstacle dense conditions.

The different technologies behaved similarly in terms of communication range and accuracy of the sensor. However, with respect to ease of use and ease of integration Z-Wave proved to have some drawbacks: all devices must be physically paired with the controller, batteries needed to be changed during the two-week test, the protocol is proprietary, and a license fee of $3,000 needs to be paid to get the appropriate documentation (and possible additional costs for product certification) [7]. An open-source library is available, based on reverse engineering of the protocol [8], but this of course imposes additional risks, e.g., with respect to the support of future versions of the protocol. INSTEON proved to be a promising technology, but was not ready for integration in a complete system, as an INSTEON gateway (Hub) was needed for which no documented interface was available. The INSTEON Hub is mainly targeted at consumers that wish to configure their home automation system rather than being integrated in a bigger system. In the end, EnOcean was chosen as the supporting technology. The purchase cost is a little higher than for the others, but maintenance costs are much lower as no batteries need to be replaced over time.

Room occupancy was detected by using two types of sensors per room (a passive infrared (PIR) and a door contact). The algorithm used is shown in Figure 2. A room always starts in the NOT_OCCUPIED state, if motion is detected while the door is closed, the room is automatically considered as occupied. Between the OCCUPIED and NOT_OCCUPIED states, there are buffer states that solve false positives like cleaning staff entering the room (in case of MAYBE_OCCUPIED, if no events arrive within t_{start}, go to OCCUPIED state) and employees not moving for a certain amount of time (in case of MAYBE_NOT_OCCUPIED, if no events arrive within t_{stop}, go to NOT_OCCUPIED state).

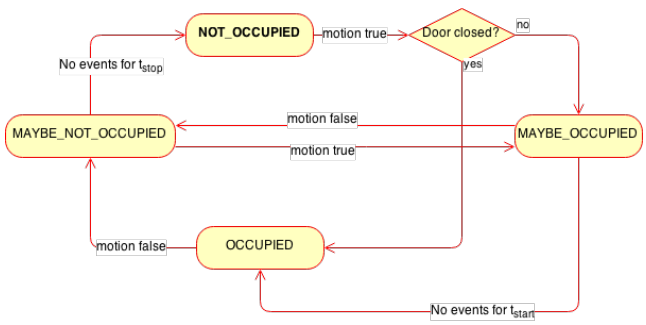

Fig. 2. Algorithm used to detect room occupancy

Next, a web application for visualising occupancy of meeting rooms was developed. This application, shown in Figure 3,

allows employees to find free meeting space without having to visit them and also provides real-time and continuous occupancy information to facility managers (in contrast with traditional methods where somebody counts occupancies manually during a limited period). So it allows organizations to understand the actual usage of their space and to make better informed decisions about the required office space.

Fig. 3. Screenshot of the Flexible Offices web application.

III. Requirements for supporting application platform

Based on the use case requirements and the initial deployment, a list of requirements for a general application platform was assembled:

- *Interoperability:* Support for future technologies and device types without having to restart the complete system.
- *Device abstraction:* Abstraction of technology-specific details of the used devices for the application (and thus developer).
- *Configuration:* Configuration of an application should be as simple and minimal as possible. Ideally, everything should be done automatically once the devices are physically installed. Remote configuration should be favored over local configuration since visiting the installation site might be expensive.
- *Remote monitoring & management:* For a cost effective follow up of deployed installations, real-time monitoring and remote mangement are key. The status of all locally installed devices should be available for inspection to solve problems in a timely fashion without on-site interventions.

IV. Evaluation of related Work

A technology scan of current state-of-the-art revealed different categories of solutions to the interoperability problem: a first category involves platforms that introduce their very own technology to solve the problem. Every single one of these platforms claims that their technology will be the next big thing. In this case, limiting ourselves to a single technology hinders the adoption of new solutions in the long run.

A second category consists of platforms that focus on a particular application domain and take the pragmatic approach of supporting the dominating technologies in that application domain. This limits developments costs and time to market for their particular case.

The third category, however, is the most relevant for this case study. There are a few platforms that claim to provide an interoperability layer between applications and networked devices, making the device technology transparent for the applications. The three most promising platforms in this category are: openHAB [9], OpenRemote [10] and Microsoft's HomeOS [11]. These platforms were subjected to a qualitative analysis based on the requirements stated in Section III. The test constituted the installation of the platform on a supported host and enabling support for a supported technology that was readily available at our lab (a door/window contact was used as a common test case in all cases) all the while taking into account the amount of configuration needed.

A. openHAB

Using openHAB [9], users are able to integrate different home automation systems in one, single application. The user is not limited by the provided use cases of manufacturers, but can create his own use cases for his home automation devices. OpenHAB is an open source solution, that is claimed to be maintained by a big community. Users are not obliged to choose one technology and stick with it, vendor lock-in is avoided. A broad range of technologies and applications can be integrated with openHAB.

The Getting Started guide [12] was sufficient to get started. However, it took a significant amount of configuration to get the platform in a working state.

The openHAB runtime is a set of OSGi bundles deployed on an OSGi framework. To enable support for the EnOcean technology, the bundle for EnOcean was added to the project. In the openHAB configuration file, the EnOcean bundle needed to be configured, specifying the serial port that corresponds to the EnOcean USB stick. This is not very user-friendly nor robust as without additional OS configuration, the serial port assigned to a particular USB device can change over time (e.g., due to power outages). This leads to either extra OS configuration or on-site visits to do platform configuration.

After having configured EnOcean support in openHAB, it became apparent that although EnOcean device events are detected (printed in the openHAB log file), the detected devices still need to be configured manually. The user must configure the *Items* configuration file, defining all devices that the system should listen to, specifying technology-specific details such as the ID and type of the sensors together with the data they can send.

After elaborate manual configuration, everything worked as promised (i.e., the sensor information could be visualized in a user interface[1]).

OpenHAB does not provide any type of automatic device discovery. It requires the end user to manually configure everything from technology and device configuration (by either entering static IP addresses on which devices are reachable, USB ports, or other location information) to the actual user interface (and only provides a text editor tool to check syntax).

[1]The user interface requires another configuration file – the site map – to be filled out.

B. OpenRemote

OpenRemote consists of multiple components. The OpenRemote Controller manages runtime integration between devices and is deployed locally on one of the supported platforms (in this case a Ubuntu laptop). The OpenRemote Designer is a cloud-based tool to create the necessary configuration and design the user interfaces used by the web console or smartphone application.

OpenRemote also supports EnOcean. The same scenario as with openHAB was repeated, trying to integrate an EnOcean door/window contact with the OpenRemote platform.

To install the Controller, the Getting Started guide [13] was used. Using the Designer, the manual configuration is visualized in a user interface. However, the ID, type and other information to define a new device or command are still required. Not only the device details, some of which needed to be looked up in technical specifications of EnOcean, but also the serial port for the EnOcean USB stick and the serial protocol and library that OpenRemote needs to use to communicate with the serial device needed to be inserted via the user interface. These details should not be specified by a non-technical end user.

C. HomeOS

HomeOS does not claim EnOcean support, so Z-Wave was chosen as the technology used in the test. Installation was hindered by the fact that a number of different Getting Started guides were available, the most recent of which [15] used different names for HomeOS[2] components. Installation of HomeOS on a Windows 8.1 PC can be done in 2 ways, either by downloading the release binaries or by getting the source code and building the project. We chose the binaries. Next, the Z-Wave USB stick was inserted into the Windows 8.1 PC and the Windows driver for the stick was automatically installed. However, apparantly, an additional HomeOS driver was needed. After some research, it became clear that one had to send an e-mail to the Lab-of-Things team to get the source code for the Z-Wave module.

Even though [11], dating 2012, already claims support for different types of devices and technologies (Z-wave and DLNA), it seems to be still in its infancy. We were not able to integrate any device with the released HomeOS binaries.

V. Real-life deployment

Before installing the occupancy system, an assessment was made on which platform to use for the real-life deployment. Given the fact that configuration was the bulk of the overhead in the iMinds deployment and the fact that the investigated frameworks failed that initial test, DYAMAND was chosen based on a number of advantages it offers.

The device model used by DYAMAND is flexible enough to support every known & foreseeable technology. Physical devices are modelled by a root device that can contain embedded devices corresponding to technology-specific functionality offered by that device. Every device can contain services with state variables modelling the state of that particular service and commands that can be executed to change the associated state.

The use of this device model enables applications to dynamically react on changes in the physical environment. In DYAMAND's eyes, USB is just another service discovery protocol and as such, any plugin - in this case the EnOcean plugin - can react on the fact that a particular USB device comes online, removing the need to configure the USB serial device as seen in the platforms discussed in Section IV.

Furthermore, the configuration needed for EnOcean sensors (ID, type, supported functionality) also gets discovered automatically once the DYAMAND EnOcean plugin receives the first message of that particular sensor. This removes the need for the EnOcean sensor configuration discussed in Section IV.

Additionally, generic service types, a motion sensor, a door/window contact, etc., are provided by DYAMAND to be used by applications. Technology-specific services can be translated to these generic service types to enable usage by applications, so applications are not forced to be aware of low-level information, and can focus on their application logic. It is, however, still possible to retrieve technology-specific information about discovered devices if required by the application. This provides the additional advantage that once applications are installed in the field, new devices of the same device type, but a new technology – that the application was not aware of at the time of development – can be added without any problems.

Before deploying the solution at P&G, it was first deployed at the research partner's (iMinds) premises for initial testing and debugging. This enabled early discovery of potential installation and configuration problems.

Figure 4 gives a schematic overview of the setup at iMinds. The sensors communicate wirelessly with a DYAMAND instance installed on a nearby BeagleBone Black (BBB) device (a low cost Linux based development platform). The BBB devices send all received data to the backend, which also hosts the web application.

Since the office space is too large to cover with one single gateway, a centralised backend is introduced to solve the problem of not being able to communicate with every single device in a certain building. All information gathered from the sensors is sent by the local DYAMAND instances to a dedicated backend that processes the sensor data, gathers information from the meeting room reservation system and combines both streams of data as input to the interface as shown in Figure 3. Additionally, configuration information about the location of sensors is transferred from the local instance to the remote backend which means that logically assigning a sensor to a meeting room does not require you to be on-site.

At the end of the summer of 2014 the deployment of EnOcean sensors at the pilot space at P&G started. A 4G router

[2]HomeOS activity seemed to have slowed down since 2012, but the Lab of Things project was started as a follow-up project to manage HomeOS installations remotely.

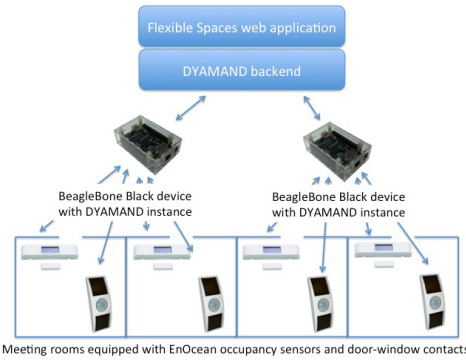

Fig. 4. Overview of the deployment of sensors and DYAMAND for the Flexible Offices pilot.

was used to communicate with the backend as the internal P&G office network could not be used for the pilot.

As start of a remote configuration tool, a simple monitoring application was deployed to monitor if the DYAMAND instances are still alive. This tool will be expanded to allow real remote configuration. DYAMAND is responsible for the local management of all discovered devices, and will report the status of these devices to the backend to enable more targeted support and simplify the development of applications such as the meeting room occupation visualizer as shown in 3.

VI. Experiences and Results

The system was installed in three phases. In a first phase, INSTEON, Z-Wave and EnOcean sensors were installed at P&G. As the system was not developed fully, it only saved the collected sensor data locally. Sensor data was analysed offline, which resulted in a reduction of 17% of the total available space; additionally, a number of offices were transformed into flexible offices where employees can use a desk for the day if space is available.

When the system was fully developed, it was first installed at iMinds, where all six meeting rooms were equipped with both a motion sensor and a door contact. The meeting rooms are scattered throughout the building, so due to the limited range of the sensors, three gateways needed to be used to cover all areas. Once the installation was finished, the web application was tested by iMinds's administrative staff during a month. After interviewing the staff, it became apparent that they mainly used it whenever someone requested a meeting room to be reserved. If they saw all meeting rooms were reserved (which is fairly common), they checked the web application to know whether or not the meeting room was actually being used. It became clear that employees either did not cancel their reservations if meetings got cancelled or reserved a meeting room longer than necessary. Using the web application, the administrative staff reminded employees to either start their meeting or cancel their reservation resulting in a more efficiently used building. During the month-long test, administrative staff reported downtime of the system twice, which was solve both times within half an hour. The iMinds installation only used the collected data to present the real-time view as shown in Figure 3, no historical data was analysed.

After the iMinds installation, the system was installed in P&G's offices in Strombeek-Bever. This installation encompassed two offices, a flexible office and a traditional office.

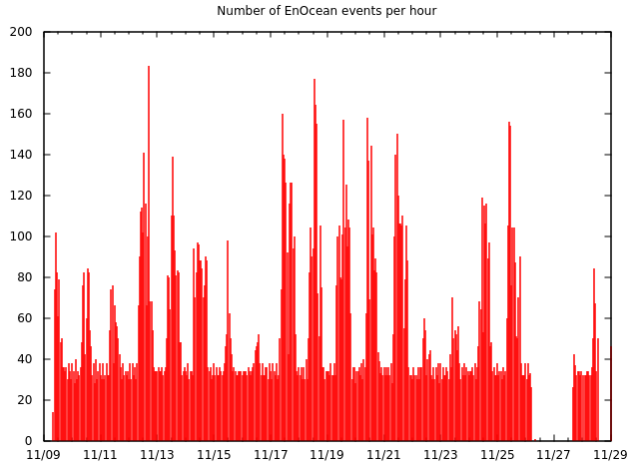

Fig. 5. Total sensor events per hour.

Figure 5 shows the amount of events DYAMAND sent to the backend per hour. There is a clear daily pattern that can be seen. This is to be expected since the sensors will detect more movement during work hours than at night. It is also obvious that there is a minimal number of events that are sent. This can be explained by the fact that the sensors send their data periodically even when it has not changed. Obviously, these events should not propagate beyond the local network in production, however, in this pilot we wanted to capture all possible events for future development efforts. There is a gap where no events where received (November 26–28), after some analysis, 4G connectivity appeared to be the culprit. Re-establishing the 4G connection solved the problem. This shows that the system as such (sensors plus gateway running DYAMAND) run stable during the complete pilot.

When analysing the sensor data using the algorithm specified in Figure 2, a number of interesting observations can be made. First of all, weekends and nights correctly get classified as NOT_OCCUPIED, which acts as a simple sanity check for the algorithm. The flexible office got used starting from 7:30 AM to 7:00 PM consistently throughout the pilot, while the traditional office was only used from 9:00 AM until 18:00 PM with a great variety between days, i.e. a number of days, the office was only used for an odd hour or less that day. If we look at the total occupation for both offices, it is abundantly clear that the concept of a flexible office is beneficial for the efficiency of use of office space. When we average out the total occupation rate of both offices, the flexible office is used for approximately 5 hours a day while the traditional office is only used for approximately 1 hour a day. This is largely due to the fact that the traditional office was not used at all a number of days during the pilot. If we discard the days the

traditional office was used less than 1 hour, the average usage rises to approximately 4 hours of usage which still is lower than the flexible office.

These findings can be valuable input for facility managers to help them efficiently use the scarce resources they have available.

VII. Conclusion

In this paper the deployment of an interoperability platform (DYAMAND) to support a meeting room occupation management system was discussed. The hands-on evaluation of similar platforms was performed by installing and enabling support for a single wireless sensor technology supported by both DYAMAND and the platform under test. In general, the required end user configuration for the tested platforms was daunting as technology-specific information was needed. This way configuration and managment of sensors and device of a variety of technologies becomes overly complex. DYAMAND, however, is completely dynamic and discovers the capabilities of each device at runtime, enabling simple and automated configuration of sensors and appliances. Using DYAMAND, application developers can take advantage of a number of features:

- *Device abstraction:* The device model enables application developers to focus on their application logic without having to be aware of low-level technology-specific details to be able to communicate with networked devices. This does, however, not mean that these details are completely hidden, applications that are interested in using technology-specific information are still able to do so.
- *Combination of different technologies in the same application:* Given the results shown in [3] (interoperability) and here (deployability), application developers can combine different technologies to better support their application without having to duplicate development and configuration effort for every supported technology. E.g., for the presented use case EnOcean could be used wherever enough environmental light is available to power the battery-less sensors in combination with any other battery-powered sensor technology to cover the blind spots.
- *Support to update technology at runtime:* Once an application has been developed and deployed, a new – better – technology might arise that provides functionality useful for the application. DYAMAND allows that sensors and devices can easily be replaced allowing already deployed applications to benefit from new and better functionality without extra development or deployment efforts for the application developer. A DYAMAND component to detect the new device has to be developed for its discovery in the framework. From this point on, applications can take advantage of all technology-specific functionality. If the device supports functionality already defined in the framework (such as motion sensing), a component translating the technology-specific device into generic services is needed for full support. If, on the other hand, the device is a completely new device, additionally, a new service type should be defined for the framework to be able to handle the new functionality. Once these components are developed, all current and future installations can benefit from this newly created support.

VIII. Future work

During deployment it became clear that simply solving interoperability issues on a local level (per installation) is not sufficient. A number of remote services are needed to fully live up to the expectations discussed in this paper. Examples are automatic addition of technology support based on local triggers (e.g., USB stick inserted, new device type discovered, etc.) and error statistics (enabling targeted development effort). Our current research focuses on listing the remote services needed to be able to remotely monitor and manage a heterogeneous installed base, both in terms of gateway hardware and in terms of heterogeneity in supported technologies across installations. These services will be tested using a wide variety (residential versus office, small-scale versus large-scale) of installations.

Acknowledgement

This work is supported by EIT ICT Labs via the project Flexible Spaces Service.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[3] J. Nelis, T. Verschueren, D. Verslype, and C. Develder, "DYAMAND: DYnamic, Adaptive MAnagement of Networks and Devices," in *Local Computer Networks (LCN), 2012 IEEE 37th Conference on*, Oct 2012, pp. 192–195.

[4] *EnOcean Technology – Energy Harvesting Wireless*, EnOcean GmbH, 7 2011.

[5] Z-Wave Alliance. (2014, Sep) About Z-Wave technology. [Online]. Available: http://www.z-wavealliance.org/technology

[6] *WHITEPAPER: The Details*, INSTEON, 2013.

[7] Z-Wave Alliance. (2014, Sep) Z-Wave for developers and OEMs: How to get started. [Online]. Available: http://www.z-wavealliance.org/z-wave-for-developers-oems

[8] open-zwave. (2014, Sep) An open-source interface to Z-Wave networks. [Online]. Available: https://code.google.com/p/open-zwave/

[9] openHAB UG. (2014, Sep) openHAB, empowering the smart home. [Online]. Available: http://www.openhab.org/features.html

[10] OpenRemote Inc. (2014, Sep) OpenRemote, open source automation platform. [Online]. Available: http://http://www.openremote.org/display/HOME/OpenRemote

[11] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *NSDI*. USENIX, April 2012.

[12] openHAB UG. (2014, Sep) openHAB, getting started. [Online]. Available: http://www.openhab.org/gettingstarted.html

[13] OpenRemote Inc. (2014, Sep) OpenRemote documentation. [Online]. Available: http://www.openremote.org/display/docs/OpenRemote+Documentation

[14] Microsoft. (2014, Sep) HomeOS project description. [Online]. Available: http://homeos.codeplex.com/

[15] ——. (2014, Sep) Lab of Things, getting started guide. [Online]. Available: https://labofthings.codeplex.com/wikipage?title=Getting%20Started%20with%20Lab%20of%20Things