Remote Management of a Large Set of Heterogeneous Devices using Existing IoT Interoperability Platforms

Heleen Vandaele, Jelle Nelis, Tim Verbelen, and Chris Develder

Ghent University – iMinds,
Department of Information Technology
Gaston Crommenlaan 8/201
9050 Gent, Belgium
firstname.name@ugent.be

Abstract. With the evolution of the Internet of Things, devices of many different technologies and manufacturers are being developed (e.g. for use cases ranging from home automation to smart cities). This creates challenges regarding interoperability between these heterogeneous devices, as well as integrating them to enable innovative applications. Currently, several integration platforms already exist to integrate technologies in a local gateway (e.g. OpenHAB, Zodianet, etc.). Yet, the local set-up and configuration still is overly complex, especially for non-technical users. In this paper, we present a remote management platform that focuses on ease of configuration and installation. It allows monitoring, configuration, diagnostics and service provisioning without manual intervention of a technical person. The platform reacts on local changes such as the installation of a new device or state changes of discovered services. This information can then be used to install required plugins, generate alarms or take problem-solving actions.

Key words: Internet of Things, Integration, Remote Management, DYAMAND, interoperability

1 Introduction

Today, many devices in our environment are connected, and communicate with other entities to deliver a service to the user. The amount of connected devices will continue to grow over the next years [17]. At the same time the amount of technologies and standards interconnecting these devices is growing. Due to this rapid growth, the landscape of technologies is very scattered, making it difficult for end users and application developers to decide which technology to use or support. Also for non-technical users it becomes more difficult to choose one or more products, without getting locked in by a vendor. If the market does not converge to one single standard, which is unlikely to happen in the near future [7], interoperability platforms are required to overcome this problem.

While some technologies, like Qeo [19], LooCI[6], AllJoyn [1] or CoAP [18] push new standards for constrained device communication, attempting to become the de facto standard in the future, other efforts are concentrating on integrating the huge amount of different technologies. These efforts result mostly in a gateway-centric approach, where an interoperability platform is located at the edge of the local network, providing the necessary abstraction for applications to communicate with devices, regardless of their technology.

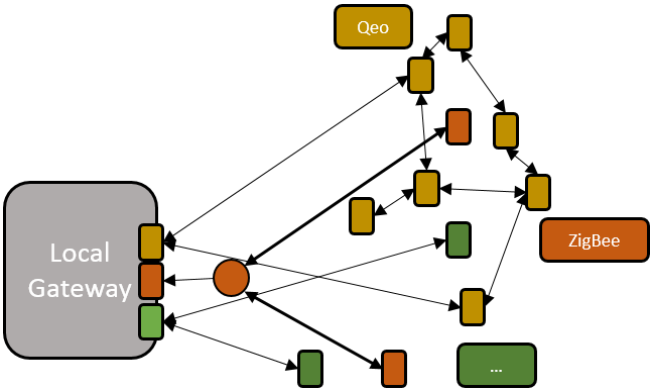

Fig. 1. Gateway Centric Integration

The most used technology for implementing these gateway integration platforms is the Open Service Gateway Initiative (OSGi) because of its platform and application independence, service collaboration, security, support for multiple network technologies and its simplicity [11]. OSGi is chosen by [4, 2, 20, 21, 8, 10] and many others to be the basis of their gateway implementation. There are alternative efforts as well, including HomeGate [20], Multimedia Home Platform (MHP) [4, 21, 10] or the Home Gateway Initiative [4]. However, many of these platforms are an installation and configuration nightmare. These platforms also tend to become very bloated in terms of storage and memory usage as they often try to support every technology out-of-the-box. Furthermore, a lot of manual configuration is required, and to manage all these platforms physical access to the device is needed. Especially in situations where multiple platforms are distributed across multiple locations, for example in home care or building management, the need for interventions needs to be kept to a minimum.

In this paper, we present a scalable architecture for a remote management platform that monitors and configures IoT gateways. This includes monitoring of information about the devices and services discovered by the gateway, data about their state and actuation of devices locally managed by the gateway as well as management of the local gateway itself. On the gateway, we also provide a technology discovery component, which enables to only install technology plugins that are present, resulting in a more lightweight gateway.

In the remainder of this paper, the existing solutions for local integration are discussed in Section 2. Based on the installation and configuration issues, the requirements for a remote management system are identified in Section 3. Based on these requirements, an architecture is proposed in Section 4. A proof-of-concept implementation is described in Section 5, and its scalability evaluated in Section 6. Section 7 concludes this paper with some pointers for future work.

2 Existing Integration Platforms

Today, quite a few integration platforms are available, both open source and commercial. To evaluate the state of the art, we have chosen five platforms for comparison, based on their availability to be used (free of charge), their supporting communication technologies and their underlying software platform. First, openHAB [15] is an OSGi based platform hosted by the Eclipse Foundation. Second we compared HomeOS [3], a Microsoft project to provide centralized control of devices in the home. A third platform is the ZiBase gateway for home control, manufactured by Zodianet [22]. Next we have OpenRemote [16], a software integration platform for residential and commercial building automation from OpenRemote Inc. Finally, DYAMAND [14] is a lightweight plugin based interoperability framework developed at Ghent University - iMinds.

Table 1. Existing Integration Platforms: Conclusion

Platform	**Platform installation**	**Add new technology**	**Detect new device**	**Install app**
openHAB	Command Line	Config Files + OSGi bundles	Config Files	Config Files
Open Remote	Command Line	Manual config in GUI	Manual in GUI	Manual config in GUI
Zodianet	App Store, Sync with web server	Automated	Config via GUI, Partially automated	App Store
HomeOS	Build from source	Manual config in GUI	Our device not detected	App Store
DYAMAND	Start Script	Automatic, At Runtime	Automatic Detection	Developer API

We evaluated the procedure to integrate a new technology they all claim to support (i.e. EnOcean), by checking the steps required to plug in an EnOcean USB receiver to the gateway and connect a new EnOcean device. Table 1 gives the summarized results of this platform comparison. The most important conclusion is that most platforms require manual configuration, either using configuration files, or using a graphical user interface. Only DYAMAND and Zodianet

automatically detect the EnOcean USB and DYAMAND is the only one to identify the EnOcean device automatically without any manual configuration. As DYAMAND is an academic project, it only has a developer API for supporting 3rd party applications, while the commercial products offer an App store to install new applications on the gateway.

3 Requirements

Based on the results of the comparison of different integration platforms, we identified the main functional requirements for usable and configuration friendly interoperability platforms. One important aspect is the ability to remotely manage the integration platform, automating the configuration for the end user as much as possible.

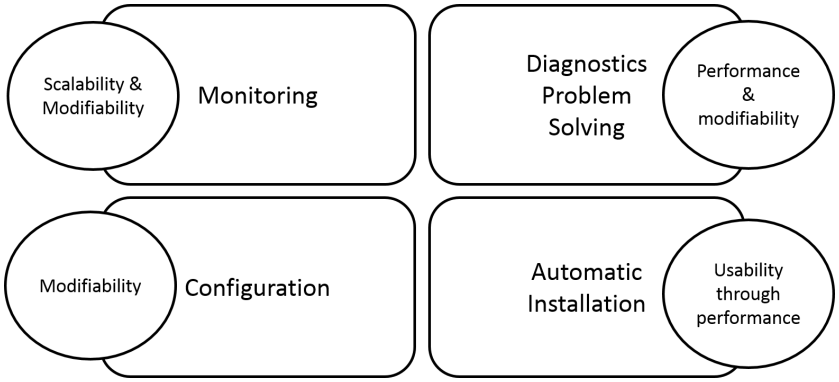

Fig. 2. Functional Requirements and Quality Attributes

This remote management entails that it should be possible to:

Monitor the interoperability platform and its environment: externalizing the local information so that it can be used for further analysis or improved diagnostics.

Configure the local interoperability platform.

Automatically install new software to support additional technologies (for example install a new driver on the platform) or additional applications without any physical interaction with the local gateway. This allows to ship a lightweight gateway platform and only install the required drivers.

Diagnose the system based on the externalized monitoring information: detect and solve problems without user intervention.

As interoperability platforms are meant to be used over a timespan of multiple years, and will change as the technologies evolve, modifiability is an important quality attribute for the remote management system. In addition to this, when

the remote management system is responsible for managing multiple local gateways, the information externalized will require the application to be scalable. Furthermore, since the goal of this remote management is to take away much of the installation and configuration burden, usability is of key importance.

4 Architecture

The general decomposition of the system is shown in Figure 3. We can make a distinction between the gateway plugin, located on the local interoperability gateway, and the remote management system. The gateway plugin is responsible for interacting with the interoperability platform installed on the gateway, for externalizing the information that is available on the gateway and executing commands that it receives from the remote management system. The remote management system (called Remote Manager in the figure), is responsible for analysing and acting upon the information it receives through both the plugin and the web client which provides a view on the monitored data for the end user, and can also be used by an administrator to manually trigger actions on the connected gateways.

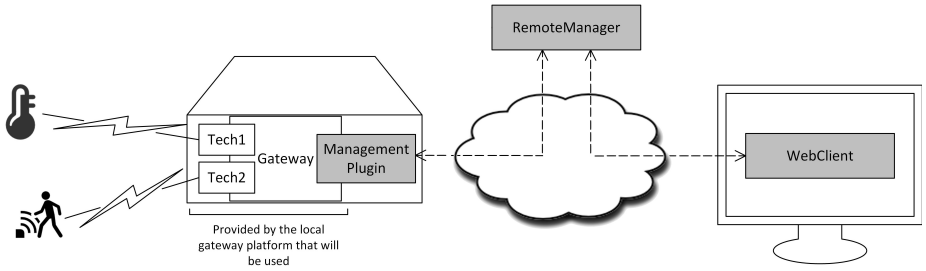

Fig. 3. High level deployment overview.

4.1 Monitoring information generated by the Gateway Plugin

The gateway plugin is tightly coupled with the integration platform running on the gateway. Choosing a different integration platform (e.g. one from Table 1), will require changing the gateway plugin implementation. The plugin collects information about the status of the platform (whether it is still online, or for example how much of the gateway system resources are used), the devices in the environment of the gateway, installed software, or errors that occur. This information is sent to the remote management system over the public Internet. Based on the commands that are included in the response received from the remote management server, the plugin can execute appropriate actions, including control of local devices. For example, when an EnOcean USB dongle is plugged

in at the local gateway, the gateway plugin will detect this device via the platform and notify the remote management system. In the response of the remote management system, an installation command for the EnOcean driver will be included. The plugin subsequently can execute the necessary steps to install that driver. The integration platform will be able to support EnOcean from then on.

4.2 Scalability through RESTful Design

The gateway plugin and the remote management system exchange REST messages. REST, which stands for Representational State Transfer [5], is a client-server model where the client and server are separated by a uniform interface and the messages are stateless. Each message should contain all the information that is needed by the remote management system to understand the request. To make this REST interface suitable for dynamic load balancing, when the amount of local gateways becomes larger than the amount of manageable gateways for one remote manager, the HATEOAS (hypermedia as the engine of application state [9]) principle is used. These principles state that the client must interact with the application entirely through the content provided by the server. This means that, for example when the local gateway is started and sends its first message to the remote management system, the response of the remote manager will contain the possible actions the client needs to proceed, as well as the web links to execute these actions. In further communication, the client will use the links that were provided in the previously received responses. This makes the client independent of where the server is located, and allows the server to dynamically redirect a client (based on e.g. the load of a particular instance of the remote management system, the profile assigned to a gateway, or even the type of request) or even migrate the remote management system to another location.

4.3 Decomposition of the Remote Management System

A more detailed decomposition of the Remote Management System is displayed in Figure 4. A REST request arrives from the gateway plugin at the Remote Management System, in the Communicator module. To analyse the different messages that are received, various Management components can subscribe to receive the information they are interested in. However, to ensure that the response time remains acceptable (within 2 seconds, according to the study performed in [12]), the Communicator starts a coordination session to verify that the Management Plugins provide a response well within time. All Management components process the information asynchronously, log everything into the knowledge repository, and provide a response in the form of Commands that have to be executed at the gateway. When all subscribed Management components have their response ready, or when the maximum time-out has elapsed, the coordination ends and the Communicator will send a response containing all available Commands. When a management component fails to provide a Command on time, this Command will be included in the next response.

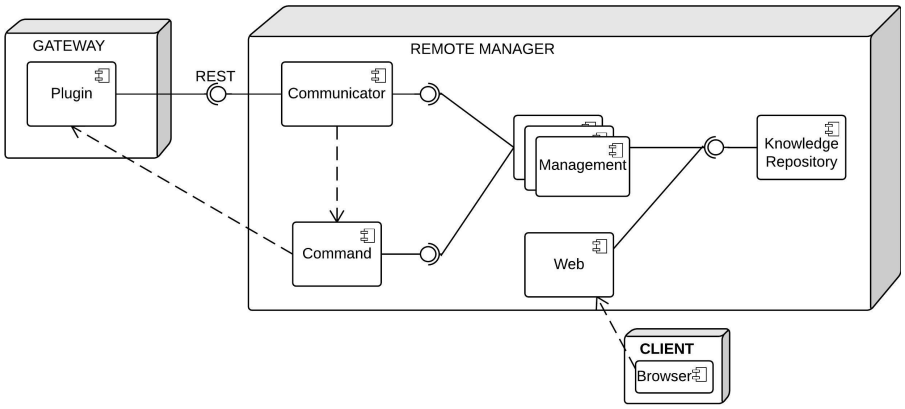

Fig. 4. Architectural Decomposition of the Remote Management System.

The Gateway Plugin is responsible to translate all information generated by the local gateway to the concepts understood by the Remote Management System. The Remote Management System can process information about the local gateway, e.g. resources used, plugins loaded and errors that occurred. Apart from that, information about devices and included services that are discovered by the local gateway can be sent. A technology-agnostic model is used to be able to represent all information about the discovered devices and services. This model is based on work done at Ghent University on the DYAMAND platform [14]. This model includes among others the concept of state variables that model the state of services in a generic way. Every time a state change is generated by a local device, the local gateway receives and translates it to the model used by the Remote Management System. Furthermore, services can contain Commands that can be executed locally. The Remote Management System will piggyback these commands using the next response sent to the gateway. This means that commands that are triggered by a local event can be executed immediately (given that the responsible Management component is loaded). In contrast, commands that are not the direct consequence of a local trigger must wait until the Gateway Plugin contacts the Remote Management System. Since the information mentioned before are sporadic events, a heartbeat is sent every 30 seconds. This enables monitoring of the local gateway in absence of other local events and ensures that commands will be executed within the heartbeat interval (as long as the gateway is online).

This architecture is extensible, as new Management components can be added to the system. Due to the asynchronous behaviour of these Management components, they can not only react quickly to incoming messages, but can also perform long-running analysis of data in the Knowledge Repository, which contains all information collected about the gateways that are managed by the remote management system. The Web component provides a view on the Knowledge

Repository for making monitoring information available to the end user. An administrative user can also issue new Commands that will be sent to the gateway once the next coordination session ends, using the Web component.

5 Proof of Concept

As a proof of concept, we have built a prototype implementation of our architecture using the DYAMAND interoperability framework as local integration platform. DYAMAND enables flexible protocol support with zero user interaction, as mentioned in a case study in a professional environment in [13] in which the requirements for a remote management system are expressed based on real-life experiences. All concepts understood by the Remote Management System are supported by DYAMAND. However, in this prototype only a limited subset is used:

- **Heartbeat request** - periodically sent to indicate that the installation is online
- **Plugin request** - sent when a new DYAMAND plugin is installed (e.g. a device driver for a certain technology)
- **Device request** - sent when the DYAMAND framework had detected a new device

Using a different local gateway involves translating the concepts used by the local gateway implementation to the concepts of the Remote Management System. Depending on the different concepts supported by the local gateway, some functionality may not be available.

The REST interface is implemented using Jersey[1] and runs on a Glassfish[2] server. The Knowledge Repository uses JPA with a Hibernate[3] backend to access the database.

At the remote manager side, several management components wait to react to REST requests arriving at the interface: for each type of request a specific analyser is implemented. The Heartbeat Analyser checks whether the heartbeat it received originated from a new or existing installation and updates the Knowledge Repository with the timestamp of the last received heartbeat. The path through the architecture is displayed by the yellow line in Figure 5. The Device Analyser handles information received with a device online request, adding new device information to the Knowledge Repository. The Plugin Analyser adds a new installed plugin to the list of plugins that are installed at the local gateway. This is the blue path displayed in Figure 5.

We also implemented two Management components that analyse the Knowledge repository. The New Technology Analyser is responsible for checking

[1] https://jersey.java.net/
[2] https://glassfish.java.net/
[3] http://hibernate.org/

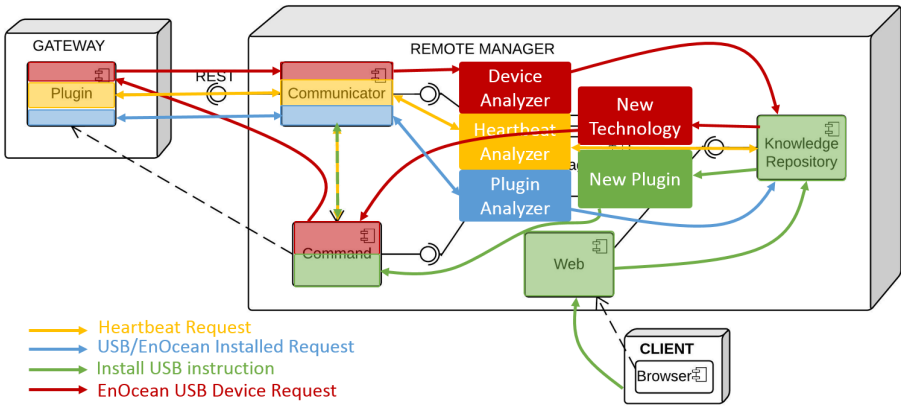

Fig. 5. Control flow path through the remote manager for the heartbeat, plugin and device requests. Each color represents a different scenario.

whether newly detected devices, added to the Knowledge Repository by the Device Analyser, are indicators for unsupported technologies in the local gateway environment. For example, when plugging in an EnOcean USB, this is detected by the gateway plugin, which sends a device online request, resulting in the device being added to the Knowledge Repository by the Device Analyser. This is picked up by the New Technology Analyser that decides the EnOcean support plugin must be installed. This adds an installation command to the response that is sent back to the gateway. The red path in Figure 5 displays this scenario. The New Plugin Analyser listens for new plugins that are added to the Knowledge Repository from the web user interface, and puts an installation command in the waiting line to be added to a response, as soon as the user adds a new (to be installed) plugin via the web interface. This is displayed in Figure 5 by the green path.

6 Evaluation

Revisiting the scenario to compare the interoperability platforms in Section 2, we now have a complete automatic system for detecting and installing a new EnOcean device. Initially the gateway will only have a basic USB device driver installed. Once the EnOcean USB stick is plugged in, the EnOcean driver is automatically downloaded and installed, and new EnOcean devices are automatically detected and monitored. Using the web interface, one can also install additional application plugins on the local gateway.

The implemented prototype focused on installing support for a new technology. Other use cases include, but are not limited to, generating alarms whenever always-on devices are no longer discovered or when the state of a particular de-

vice is erroneous, executing commands on local devices based on state of devices located in the same or other installation sites, etc.

To evaluate the scalability of the system, we measured the response times for requests coming from multiple gateways sending various requests. Each simulated gateway sends a heartbeat every 30 seconds, a device online request every minute and one plugin request in a 3 minute timespan as depicted on Figure 6. When a new plugin is installed, this is followed by 10 new devices that come online, simulating freshly installing support for a new technology. We use JMeter to send simultaneous requests for n gateways, n increasing from 1 to 1700.

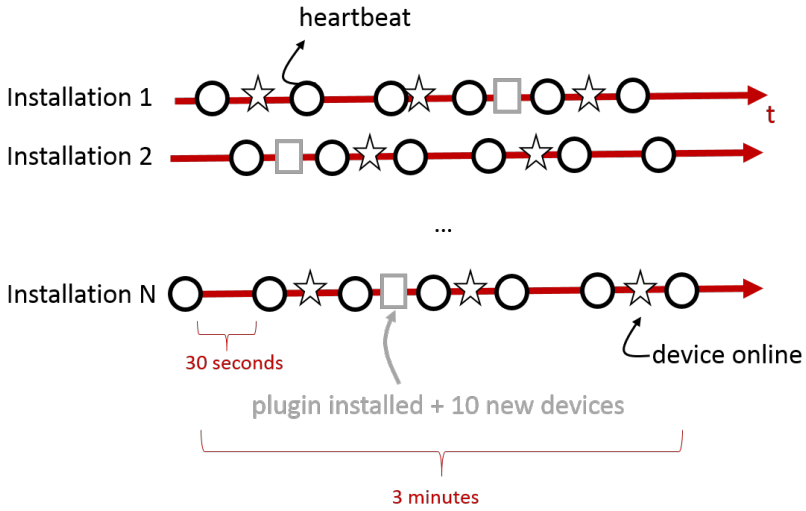

Fig. 6. Visualisation of the test scenario: each active gateway sends a heartbeat every 30s, a device online request every minute and one plugin installed request.

We repeated the experiment 10 times and visualized the results in Figure 7. The experiments were conducted on a server equipped with an Intel Xeon E5-2650 CPU clocked at 2.6GHz and 48GiB of RAM.

The left Y-axis shows the response time in milliseconds on a logarithmic scale for all graphs except *Error*, while the right axis' unit is the relative amount of lost requests in percentage for the *Error* graph. Up until 950 gateways using the same instance of the Remote Management System, the performance is up to par. When additional gateways are added, the maximum amount of parallel requests that can be handled within performance bounds by that instance is reached, which results in a rapid decrease of the performance of some requests. Although the median stays more or less the same, the average response time flirts with the 2-second threshold and the maximum response time skyrockets which implies that most requests still get processed fast enough, but that a percentage of requests have to wait for as long as 10 seconds. Although performance was not

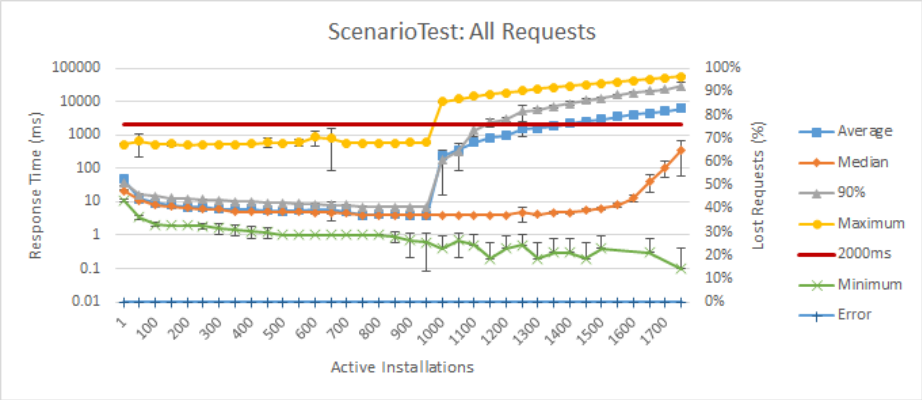

Fig. 7. Aggregated results for the test scenario, using JMeter to send data according to the scheme from Figure 6

the main focus of the prototype (technology choices like JPA and Hibernate were inherited from a predecessor project), this evaluation learns that it is possible to implement the presented functionality in a scalable way if you leverage the benefits of HATEOAS.

7 Conclusion

The interoperability platforms providing a solution for the interoperability issues between heterogeneous devices, often are not designed for ease of installation and configuration that is important for the usability of these platforms. Key requirements therefore include monitoring, diagnostics, automated configuration and automatic software installation. For such usability, it is critical to avoid manual intervention and allow technically trained supervision of geographically distributed integration gateways. In this paper we have presented a scalable architecture for a remote management platform, aiming to reduce the configuration burden for the end user. New technologies are automatically detected at the local gateway, and the required drivers are automatically fetched and installed, resulting in a lightweight gateway installation. We built a proof of concept of our system, showing it can easily manage hundreds of installations on a single server. As future work, we aim to look into more complex management components, i.e., for automatic fault detection or other data analysis. We will also investigate how to better handle technologies that require manual configuration, such as device pairing in ZigBee, or cross-technology discovery, e.g. the Philips Hue bridge that can be discovered locally using UPnP.

Acknowledgements

Part of the work was supported by the iMinds IoT research program.

References

1. AllSeen Alliance. A Common Language for the Internet of Everything, November 2014.
2. S. Arrizabalaga, P. Cabezas, J. Legarda, and A. Salterain. Multi-Residential Gateway: an Innovative Concept and a Practical Approach. *IEEE Transactions on Consumer Electronics*, 54(2):444–452, May 2008.
3. Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An Operating System for the Home. In *NSDI*. USENIX, April 2012.
4. J. C. Duenas, J. L. Ruiz, and M. Santillan. An End-to-End Service Provisioning Scenario for the Residential Environment. *IEEE Communications Magazine*, 43(9):94–100, 2005.
5. Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
6. D. Hughes, K. Thoelen, J. Maerien, N. Matthys, J. Del Cid, W. Horre, C. Huygens, S. Michiels, and W. Joosen. LooCI: The Loosely-coupled Component Infrastructure. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, pages 236–243. IEEE, August 2012.
7. AutomotiveIT International. Gartner Laments Absence of Internet-of-Things Standards, February 2015.
8. Choonhwa Lee, D. Nordstedt, and S. Helal. Enabling Smart Spaces with OSGi. *IEEE Pervasive Computing*, 2(3):89–94, July 2003.
9. Wayne Lee. Why HATEOAS a Simple Case Study on the often Ignored REST Constraint, June 2009.
10. Shou-Chih Lo, Ti-Hsin Yu, and Chih-Cheng Tseng. A Remote Control and Media-Sharing System using Smart Devices. *Journal of Systems Architecture*, 60(8):671–683, September 2014.
11. D. Marples and P. Kriens. The Open Services Gateway Initiative: an Introductory Overview. *IEEE Communications Magazine*, 39(12):110–114, December 2001.
12. Fiona F. Nah. A Study on Tolerable Waiting Time: How Long are Web Users Willing to Wait? *Behaviour & Information Technology*, 23(3):153–163, May 2004.
13. J. Nelis, H. Vandaele, M. Strobbe, A. Koning, F. De Turck, and C. Develder. Supporting Development and Management of Smart Office Applications: A DYAMAND Case Study. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1053–1058, May 2015.
14. 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*, pages 192–195. IEEE, October 2012.
15. openHAB UG. openHAB, Empowering the Smart Home, Sep 2014.
16. OpenRemote Inc. OpenRemote, Open Source Automation Platform, Sep 2014.
17. Janessa Rivera and Rob van der Meulen. Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020, December 2013.
18. Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.
19. Technicolor. Discover Qeo, November 2014.
20. Peter Utton and E. Scharf. A Fault Diagnosis System for the Connected Home. *IEEE Communications Magazine*, 42(11):128–134, November 2004.
21. D. Valtchev and I. Frankov. Service Gateway Architecture for a Smart Home. *Comm. Mag.*, 40(4):126–132, April 2002.
22. Zodianet. Zodianet, Home Robotics, Sep 2014.