Evaluation of OSGi-based Architectures for Customer Energy Management Systems

Mike Pichler
Corporate Technology
Siemens AG
Vienna, Austria
Email: mike.pichler@siemens.com

Armin Veichtlbauer and Dominik Engel
Josef Ressel Center for
User-Centric Smart Grid Privacy, Security and Control
Salzburg University of Applied Sciences
Email: {firstname.lastname}@en-trust.at

***Abstract*—The Smart Grids requires distributed components to control the demand side. These components have to fulfill various requirements from communication issues to graphical user interfaces for monitoring and controlling. In this paper, we define high level requirements to software architectures for Customer Energy Management Systems and evaluate some existing solutions based on Java and OSGi. The conclusion shows, that there are some promising solutions available, although none of the evaluated frameworks fulfills all requirements.**

***Keywords*—Smart Grid, Customer Premise, Smart Home, Energy Management, Architecture, CEMS**

I. Introduction

Applications like demand side management and demand response allow to integrate the customer premise into the energy system. While there is plenty of experience in optimizing energy generation and distribution, the customer premise is a relatively new topic for research and industry [1]. The Smart Grid Coordination Group introduces the *Customer Energy Management System* (CEMS) to connect appliances that consume, generate or store energy to the Smart Grid [2]. These appliances are located in residential or commercial buildings.

While commercial facilities usually are equipped with *Building Automation Systems* (BAS), today's residential buildings mostly don't have any automation. Even if there is a *Home Automation Systems* (HAS) installed (making the customer premise a "Smart Home"), it is completely different to a BAS, as it usually lacks higher order business logics such as energy optimization, etc. Yet it is expected that major consumers in households like heat pumps will become "smarter", i.e., they will be able to communicate with a CEMS.

This paper evaluates different open source software architectures and frameworks, which can be used for a CEMS. It is structured as follows: After a short overview of related work, the software requirements on CEMSs will be defined based on use cases from different sources. In the evaluation section, the candidate solutions will be evaluated based on these requirements. Finally, a conclusion of the work is given, as well as an outlook to further steps.

II. Related Work

In [3], the authors introduce Information and Communication Technologies (ICT) architecture functionalities for Smart Homes. They mention three technical "measures" on which the functionalities of the ICT architecture are based: End user feedback on his/her energy behavior, automated decentralized control of distributed generation and demand response for a better local match between demand and supply, and finally control for grid stability and islanding operation to maintain or restore stability in (distribution) grids. These measures describe the basic functionality of CEMSs. More detailed use cases are formulated by the *Home Gateway Initiative* (HGI) in [4]. Nevertheless, all HGI use cases can be allocated to the measures listed above. The use cases and measures are used to extract some basic requirements of CEMSs.

Smart Homes can be divided into several domains like safety, comfort, entertainment, etc. [5]. An increasingly important role over the last years play electro mobility and Ambient Assisted Living. These domains have an enthusiastic development and user community. Thus, several software architectures for Home Automation are available, most of them under open source licenses. Many architectures are implemented in Java and use the OSGi framework [6] to ensure a modular and adaptable design. The benefits of using OSGi for CEMS have been described in [7] and [8].

Additional inputs come from different research and development projects within the *Smart Grid Model-Region Salzburg*. In the Building2Grid project, so-called Building Agents were used to enable physical and marked-oriented demand response [1]. This agent got enhanced to the *Building Energy Agent* (BEA), which has been installed in 40 buildings in the township Köstendorf near Salzburg. The third relevant project is the Smart Grids-friendly housing area HiT, which is also equipped with a BEA enabling demand response with day-ahead pricing.

III. Requirements and Evaluation Criteria

This section describes the functional requirements of software architectures for CEMS. In principle, the evaluation of the considered architectures and frameworks can be done in three ways: Evaluation of non-functional requirements, source-code evaluation based on static and dynamic software metrics, and evaluation of functional requirements. Whereas non-functional requirements can be evaluated using methods like the Architecture Tradeoff Analysis Method (ATAM) [9], source-code evaluation is usually done with tools like Sonatype Qube. This paper focuses on the evaluation of functional requirements only, based on the topics named by [10].

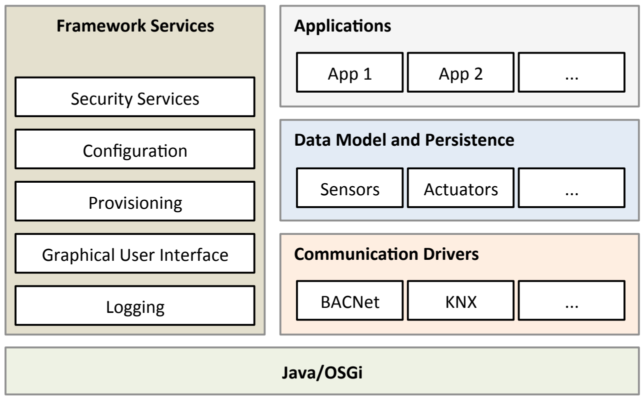

Fig. 1. Application Framework on the basis of [12]

The basic components of an application framework for a CEMS are shown in Figure 1. On top of Java and OSGi, communication drivers enable connecting different devices and appliances to the CEMS. These components can be accessed from applications using a standardized data model. In addition, common services like logging, configuration and a graphical user interface have to be provided. In this section, we define the functional requirements for the later evaluation. All requirements listed in the following sections have been derived from the use cases described in [4], [11] and [2].

A. Applications

Applications represent the business logic of the CEMS. They can use different services provided by the framework and provide data to or consume them from other applications (see Figure 1). A CEMS can be a multi-vendor system, running applications from different manufactures. This leads to the problem, that a single application can cause a system crash, for instance when allocating too much memory. A way to isolate applications in OSGi platforms has been presented in [13].

The architecture should enable application development in an effective and unified way. Developers shall be able to access common functionality provided by the framework using standardized methods and services. A common way to provide this access is to pass a context object to the application during initialization. This method is also used by OSGi, which passes a context instance to bundles and components [6]. An architecture for CEMS can use the mechanisms provided by the OSGi framework or can define own classes or interfaces for application development.

R1.1: The CEMS shall provide a standardized way to access framework services.
R1.2: The CEMS shall separate the runtime environment of different applications (sandboxes).
R1.3: The CEMS shall provide a common (interface) class for applications.

B. Communication

As described in [2], one of the main purposes of a CEMS is to connect the customer premise to higher-level energy and/or power management systems. Thus, a CEMS in its most simple form is a communication gateway between the BAS/HAS and the Smart Grid. Thus, communication protocols and data models are the most important aspects of a CEMS architecture or framework. Figure 2 shows the context of a CEMS. The arrows indicate communication channels with the surrounding world. The communication with external entities will be standardized by IEC TC57 WG21 and is still work in progress. As transportation protocol, the extensible messaging and presence protocol (XMPP) is considered.

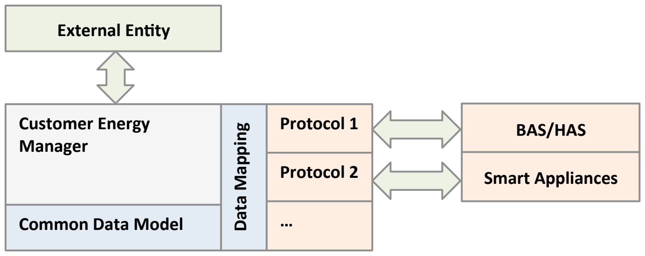

Fig. 2. CEMS context as defined in [11]

Inside the building, the CEMS mainly communicates with energy-related devices and appliances. As this domain is very heterogeneous and uses a lot of communication protocols like BACNet, KNX, or Zigbee, the CEMS must be able to integrate drivers for these protocols very easily and in a standardized way. Requirements may differ depending on the type of the building. For commercial buildings, equipped with a BAS, the CEMS can pass schedules and other complex data structures to the BAS using management level protocols like BACNet or OPC-UA. For residential buildings, equipped with a HAS, simple data structures like on/off commands must be passed to devices like heat pumps or electrical heaters using field level protocols like KNX. Inside the CEMS, this makes a great difference. While the control algorithms in larger buildings are located in the BAS, residential buildings need to run these algorithms inside the CEMS, depending on the capabilities of the connected devices and appliances. This leads to the following communication-related requirements:

R2.1: The CEMS shall provide drivers for the communication with external entities for protocols like XMPP, REST or SOAP web services.
R2.2: The CEMS shall provide management level communication drivers for protocols of the building automation domain like BACNet or OPC-UA.
R2.3: When the CEMS will be used in residential buildings, it shall provide field level communication drivers for protocols of the home automation domain like KNX.
R2.4: The CEMS shall provide developers with the ability to implement and add new communication drivers easily using a standardized internal interface.

C. Persistence and Data Models

Persistent data have to be stored in a way, that they are still available after a system shutdown or restart. One of the most common ways doing this is to use (relational) databases. These databases work very well, if the structure of the data is known and does not change over time. Yet developers have to be able to define and add new complex data structures to the framework. Thus it must be possible to store new – and at the moment of framework creation unknown – data structures into the system's local database. For that purpose document-oriented database systems are more appropriate [14].

On the other hand, a common task for CEMS is to store time series of primitive data values, e.g., storing historical energy consumption data. Values are sampled in a predefined interval and are stored together with the current date and time. Since the structure of this data doesn't change, relational databases can be used.

R3.1: The CEMS shall be able to store complex data structures in a generic way, independent of the data model.
R3.2: The CEMS shall be able to store historical data like time series and query them effectively.
R3.3: The CEMS shall provide developers with the ability to define and add complex data structures, which can be used by applications running within the framework.
R3.4: The CEMS shall provide developers with the ability to map complex data structure to primitive types like Boolean or float values to enable the usage of field level protocols.

D. Provisioning and Configuration

When a CEMS has been installed inside a building, a lot of configuration work has to be done. The initial configuration is called *provisioning* and may also include software downloads and updates. The provisioning process can be done manually (by an engineer or the customer) or remotely by an external service provider.

Since configuration can be a very cumbersome process for customers, it should be made as easy as possible. The CEMS must thus provide capabilities to handle configuration for all applications in a standardized way. A single instance must administrate all configuration artifacts in order to enable efficient configuration.

As mentioned before, another aspect of provisioning is the download and installation of software from a central repository or market place. Beyond that, the framework shall provide a standardized way for software updates, which is independent of the provider of a single application running on the system. Apple's ecosystem for the mobile operating system iOS is an example for that: the App Store enables to install new applications and also handles software updates centrally. Additionally to the possibility to update single applications, an update process for the framework itself should exist.

R4.1: The CEMS shall administrate all configuration artifacts centrally and in a standardized way.
R4.2: The CEMS shall be able to handle software update for the installed applications centrally.
R4.3: The CEMS shall be able to download and install software updates for the framework itself from a central server ecosystem.
R4.4: The CEMS should provide the customer with the ability to search, download, install and uninstall software applications.

E. Privacy and Security

Both, privacy and security are important aspects for CEMS. This applies due to three reasons:

(1) CEMS are connected to many real-world devices like heating systems or smart appliances. Offenders can cause serious damage yielding high costs.
(2) CEMS are often designed as multi-vendor devices, running applications from a lot of (more or less trustable) sources.
(3) CEMS collect confidential data from its users like energy consumption.

Security issues can be split up into several layers. In [15], the author introduces the network security layer, the operating system layer and the application layer. While the network security layer and operating system layer are out of scope of this paper, the application layer should be covered by CEMS frameworks and architectures. Java brings along a very comprehensive security concept for application security with domain specific policies, e.g., for accessing files, establishing network connections and so on. By default, a Java application has unrestricted permissions, yet by instancing a so-called Security Manager, policies must be provided to control the application's permissions.

Another aspect in this context is code security. As malicious code can cause serious damages inside the system, only trusted code should be executed. This can be realized by signing code with a certificate. The CEMS should only execute code signed by a trusted instance. This is especially important for multi-vendor systems, which execute several applications from different vendors in a single system.

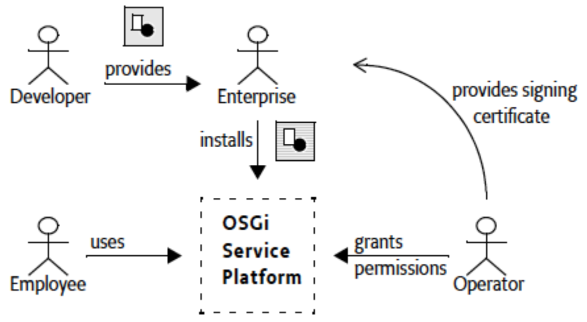

Fig. 3. OSGi Delegation Model as defined in [6]

The OSGi specification adds several new features to the Java 2 security architecture to adapt it to the typical use cases of OSGi deployments [6]. In the current version 5 of the OSGi specification, this extension is called *Conditional Permission Admin Service Specification*. The service maintains a system-wide table of policies. The framework operator can provide certificates for signing bundles and grants permissions to the installed bundle in the OSGi framework. Figure 3 shows this process, which is called the *Delegation Model*.

R5.1: The CEMS shall only execute code which is signed by the operator or a delegated actor.
R5.2: The CEMS shall restrict accessing functions using the Java and OSGi security mechanisms.
R5.3: The CEMS shall encrypt confidential customer data like measurement time series.

F. Graphical User Interface (GUI)

GUIs are important parts of CEMSs in order to visualize energy-related data like current consumption. Common ways for providing such interfaces in the domain of home and building automation systems are web-based interfaces running locally or on a central server and applications for mobile phones or tablet PCs. Independent of the GUI type, data from the CEMS must be provided. Because this can be a security and privacy issue, all data should be accessed in a standardized way using a single point of entry to the system. This enables to install a right management system and to apply policies like described in the previous subsection.

R6.1: The CEMS shall provide a single point of entry for accessing data related to graphical user interfaces (e.g. a webservice).
R6.2: The CEMS shall provide a web-based graphical user interface (GUI) for operation and configuration.
R6.3: The CEMS shall provide developers with the ability to extend the GUI with a standardized look and feel.
R6.4: The CEMS should provide applications for common mobile device operating systems like Apple's iOS or Google's Android.

IV. Evaluation Results

Based on the requirements described in the last section, we have evaluated four existing candidate solutions. Table I shows these frameworks and the evaluated versions. The listed frameworks have been chosen according to following reasons: Distribution in the Smart Home domain and availability with open source licenses (economic reasons), as well as implementation in Java and utilization of the OSGi framework [6] (technical reasons).

TABLE I. Evaluation Candidate Solutions

Framework	Manufacturer	Version	Link
OpenMUC	Fraunhofer ISE	0.12.3	http://www.openmuc.org/
IoTSyS	TU Vienna	0.1	https://code.google.com/p/iotsys/
OpenHAB	openHAB UG	1.4.0	http://www.openhab.org/
OGEMA	Fraunhofer IWES	2.0	http://www.ogema.org/

A. Applications

OpenMUC as well as IoTSyS provide no special classes or interfaces for building an application. Services like the data access service can by used directly from the OSGi service registry. OGEMA follows a different approach for applications: the framework provides an interface class for application development. This interface allows to access framework services. For security reasons, the direct use of OSGi services is not allowed.

While building applications with the mentioned frameworks requires Java programming knowledge, OpenHAB uses scripts based on Eclipse's Xtend language. Base functionalities can still be implemented in Java using so-called actions. These actions can be called from scripts afterwards. The third concept of OpenHAB are rules, which are also Xtend scripts. Rules can be compared with if-statements and allow to execute scripts based on certain conditions (e.g., the room temperature drops below a predefined threshold). Another interesting feature is the possibility to run scripts via Google calendar entries or manually from an XMPP-based chat console.

Table II summarizes the fulfilled requirements of each framework which has been evaluated. While all frameworks provide a standardized way to access the framework services (R1.1), none of them uses sandboxes to separate the different applications (R1.2). OGEMA is the only framework that provides an application interface (R1.3).

TABLE II. Fulfilled Application Requirements

Requirement	OpenMUC	IoTSyS	OpenHAB	OGEMA
R1.1	✓	✓	✓	✓
R1.2				
R1.3				✓

B. Communication

All evaluated frameworks provide the concept of protocol drivers for communication. The number of supported protocols diverge widely between the frameworks. While OGEMA only supports four protocols, OpenHAB brings along over 70 so-called bindings. These range from building automation protocols like KNX to appliances like Samsung Smart TVs. IoTSys comes with about 10 drivers for the most common protocols from the building automation domain. Furthermore, it is the only framework that supports the BACNet protocol, which is very popular for commercial and industrial sites.

On the other side, OpenMUC has a strong support of energy-related protocols. In addition to KNX and Modbus, OpenMUC also supports IEC 61850, DLMS/COSEM and SML. These protocols can be very useful to connect the energy domain with the building domain. Table III shows a selection of the supported protocols from these two domains.

TABLE III. Supported Protocols (Selection)

Protocol	OpenMUC	IoTSys	OpenHAB	OGEMA
KNX	✓	✓	✓	✓
Modbus TCP	✓	✓	✓	✓
wM-Bus	✓	✓		
BACNet		✓		
EnOcean		✓	✓	
IEC 61850	✓			
DLMS/COSEM	✓			

Table IV shows the fulfilled requirements related to communication. All frameworks provide field level drivers, but only IoTSyS also provides management level drivers like BACNet.

TABLE IV. Fulfilled Communication Requirements

Requirement	OpenMUC	IoTSyS	OpenHAB	OGEMA
R2.1			✓	
R2.2		✓		
R2.3	✓	✓	✓	✓
R2.4	✓	✓	✓	✓

C. Data Model and Persistence

Persistence is strongly related to the internal representation of the data, which is also called the internal data model. All of the evaluated frameworks introduce such a data model, independent of the communication protocols. As described in the previous chapter, this works well for primitive data types like numbers or boolean values. OpenMUC defines a set of primitive types like boolean, integer or double values. Complex types like data structures are not possible, but all data types can be persisted in a unified way into a time series database. IoTSyS uses the Open Building Information Exchange (oBIX) format, as defined in [16]. Although only primitive data types are implemented at the moment, complex data structure are possible with oBIX. In the current version, the IoTSyS persistence layer doesn't support complex data types.

OGEMA brings along its own data model, which also includes complex data types. It defines so-called standard resource types like temperature sensors and switching actuators [17], but also models for more complex devices like heat pumps or electrical meters. The data model can be extended by writing Java interface classes that derive from a resource interface. OpenHAB defines singlevalue primitive types as well as complex types, which are sorted maps of primitive values with a string-key. All types are called items. Different persistence layer implementations allow to store these items into databases like MySQL, but also to persist item states, e.g., through MQTT messages (see [18] for details).

TABLE V. FULFILLED PERSISTENCE REQUIREMENTS

Requirement	OpenMUC	IoTSyS	OpenHAB	OGEMA
R3.1				✓
R3.2	✓		✓	✓
R3.3		✓		✓
R3.4		✓		✓

D. Provisioning and Configuration

The configuration of all evaluated frameworks is done using configuration files. Although these files could also be located on or updated from centralized servers, remote configuration and provisioning is not implemented in any of the frameworks.

The OpenMUC configuration can by accessed by a ConfigService implementation, which handles file access and listens for configuration changes. The config file contains all channel information, but is not intended for storing application configurations, too. This must be implemented by developers for each application separately, a unified mechanism is not provided. A similar approach is used in IoTSys. The configuration is read from a single Java properties file.

With the *ConfigDispatcher* bundle, OpenHAB is the only framework that provides a unified way to handle configuration for applications in the framework. OpenHAB reads all configuration entries from a single text file and distributes the entries using OSGi's ConfigAdmin service. The format of the configuration file is similar to a standard Java properties file, with the exception that the property name must be prefixed by the name of the application.

The current release of OGEMA doesn't provide configuration handling. As the direct access to OSGi services is prohibited, applications must handle loading and persisting configuration by their own.

None of the evaluated frameworks provides support for handling provisioning or configuration issues from a remote server. Because of the central administration of all configurations, the implementation of remote configuration can be done easily in OGEMA and OpenHAB. Provisioning (e.g., remotely install software bundles) and handling applications compared to an App Store is a new concept for all evaluated frameworks and has to be implemented from scratch. Table VI shows, that most of the requirements related to configuration and provisioning are not fulfilled by the evaluated frameworks. Approaches for provisioning residential gateways can be found in [19] and [20].

TABLE VI. FULFILLED CONFIGURATION REQUIREMENTS

Requirement	OpenMUC	IoTSyS	OpenHAB	OGEMA
R4.1			✓	
R4.2				
R4.3				
R4.4				

E. Privacy and Security

None of the evaluated frameworks provides encryption for the measurement and configuration data. This leads to the problem, that an offender can access all private data of a compromised system. In general, OpenMUC provides no security mechanisms except an authorization when trying to access the graphical user interface. IoTSyS brings an implementation of the *Extensible Access Control Markup Language* (XACML) standard, which is described in [21].

The OGEMA framework uses the Java security architecture. It defines permissions for common tasks like accessing online values or historical data and uses the security manager to check these permissions when executing the related commands. OpenHAB doesn't provide any security mechanisms. Table VII shows, that security and privacy issues, as introduced in [22] and [23], are not implemented in most cases.

TABLE VII. FULFILLED SECURITY REQUIREMENTS

Requirement	OpenMUC	IoTSyS	OpenHAB	OGEMA
R5.1				
R5.2		✓		✓
R5.3				

F. Graphical User Interface

OpenMUC provides a web-based user interface with some functionalities related to recording and viewing time series of measurement data. The user interface has been implemented in a modular way, so extensions with a consistent look and feel are possible. All data necessary for the user interface can be retrieved from a REST webservice. IoTSyS doesn't have a user interface for operation or configuration. On the other hand, an engineering tool called Obelix is part of the project, which allows to define wirings between single data points. Together with the data abstraction layer, data point values can be exchanged over different communication protocols using this wiring. As IoTSyS is designed to use REST webservices and the oBIX data model [16], user interfaces can easily access all data points.

OGEMA uses the Apache Wicket framework for its user interface. The evaluated alpha-version of OGEMA 2.0 provides an explorer for the data repository, which enables supervision of all data structures in a generic way. Application programmers are able to extend the user interface with custom pages, very similar to the possibilities provided by OpenMUC.

The most comprehensive user interface support comes with OpenHAB. It ships with a web-based user interface, as well as native applications for iOS and Android (which can be downloaded in the particular app stores). The apps allow to monitor and control home automation components in a generic way, so additional devices can be added easily and without programming knowledge. Additionally, OpenHAB includes an Eclipse-based rich-client application which is called "Designer". This application allows to implement scripts and define rules using the Xtend scripting language (see Section IV-A).

TABLE VIII. FULFILLED GUI REQUIREMENTS

Requirement	OpenMUC	IoTSyS	OpenHAB	OGEMA
R6.1	✓	✓	✓	✓
R6.2	✓		✓	✓
R6.3	✓			✓
R6.4			✓	

V. CONCLUSION AND OUTLOOK

Most of the requirements have been fulfilled by the OGEMA 2.0 framework, closely followed by OpenHAB. As the analyzed OGEMA framework is alpha-release, it seems to be a promising solution for the future. OpenHAB has a comprehensive support of communication drivers and offers various graphical user interfaces. On the other hand, security concepts are completely missing in OpenHAB. IoTSyS as well as OpenMUC strongly focus on communication issues, while disregarding complex data structures and centralized configuration. In particular the combination of using IPv6 to address data points, data abstraction with the oBIX standard as well as XACML for policy-based access control in IoTSyS is an interesting approach for web-based "Internet of Things" (IoT) systems.

The result also shows, that none of the evaluated frameworks fits 100% to the defined requirements. Although solutions and implementations for all issues are available (some references can be found in Section IV), they still have to be combined in comprehensive framework solution. Research topics are unified data models as well as communication standards for connecting the CEMS with the outside world. Researchers and standardization groups are already working on these issues, considering aspects like security, privacy and scalability.

ACKNOWLEDGMENTS

The work described in this paper has been performed within the research project "Josef Ressel Center for User-Centric Smart Grid Privacy, Security and Control". The financial support by the Austrian Federal Ministry of Science, Research and Economy (BMWFW) and the Austrian National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES

[1] P. Palensky and D. Dietrich, "Demand side management: Demand response, intelligent energy systems, and smart loads," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 3, pp. 381–388, 2011.

[2] *Smart Grid Reference Architecture*, CEN/Cenelec/ETSI Smart Grid Coordination Group Std., Nov. 2012.

[3] K. Kok, S. Karnouskos, D. Nestle, A. Dimeas, A. Weidlich, C. Warmer *et al.*, "Smart houses for a smart grid," in *Proc. 20th International Conference and Exhibition on Electricity Distribution (CIRED 2009)*. IET, Jun. 2009.

[4] Home Gateway Initiative (HGI), "Use Cases and Architecture for a Home Energy Management Service," HGI, Tech. Rep., Aug. 2011.

[5] *Die Deutsche Normungs-Roadmap Smart Home + Building*, Verband der Elektrotechnik, Elektronik und Informationstechnik e.V. (VDE) Std., 2013, (in German).

[6] The OSGi Alliance, "OSGi core release 5," 2012. [Online]. Available: http://www.osgi.org/download/r5/osgi.core-5.0.0.pdf

[7] R. P. Diaz, A. Fernández, M. Ramos, J. J. Pazos, J. García, and A. Gil, "Enhancing residential gateways: a semantic OSGi platform," *Intelligent Systems, IEEE*, vol. 23, no. 1, pp. 32–40, 2008.

[8] S. Zeadally and P. Kubher, "Internet access to heterogeneous home area network devices with an OSGi-based residential gateway," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 3, no. 1, pp. 48–56, 2008.

[9] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*. IEEE, 1998, pp. 68–78.

[10] G. Starke, *Effektive Softwarearchitekturen*. Carl Hanser Verlag, 2014.

[11] *IEC62746 - System interface between customer energy management system and the power management system*, International Organization for Standardization (ISO) Std., 2014, (Draft).

[12] Fraunhofer IWES, "OGEMA: Open Gateway Energy Management," Fraunhofer IWES, Tech. Rep., 2012. [Online]. Available: http://www.ogema.org/downloads/fraunhofer-iwes_ogema_fs_1.pdf

[13] N. Geoffray, G. Thomas, G. Muller, P. Parrend, S. Frénot, and B. Folliot, "I-JVM: a Java virtual machine for component isolation in OSGi," in *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*. IEEE, 2009, pp. 544–553.

[14] N. Leavitt, "Will NoSQL databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[15] C. Scarioni, *Pro Spring Security*. Apress, 2013.

[16] OASIS Open Building Information Exchange (oBIX) TC, "oBIX 1.0. OASIS Committee Specification," Dec. 2006. [Online]. Available: http://www.oasis-open.org/committees/download.php/21462/obix-1.0-cs-01.zip

[17] D. Nestle, J. Ringelstein, H. Waldschmidt, and I. Fraunhofer, "Open energy gateway architecture for customers in the distribution grid," *Information Technology, Oldenbourg Verlag, Munich*, pp. 83–88, 2010.

[18] D. Locke, "MQ Telemetry Transport (MQTT) V3. 1 Protocol Specification," IBM, Tech. Rep., 2010. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html

[19] J. C. Duenas, J. L. Ruiz, and M. Santillan, "An end-to-end service provisioning scenario for the residential environment," *Communications Magazine, IEEE*, vol. 43, no. 9, pp. 94–100, 2005.

[20] D. Zhang, "A new service delivery and provisioning architecture for home appliances," in *Consumer Electronics, 2003. ICCE. 2003 IEEE International Conference on*. IEEE, 2003, pp. 378–379.

[21] T. Moses, *Extensible access control markup language (XACML) version 2.0*, OASIS Open Std., Feb. 2005.

[22] C.-C. Huang, P.-C. Wang, and T.-W. Hou, "Advanced OSGi security layer," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 2. IEEE, 2007, pp. 518–523.

[23] P. H. Phung and D. Sands, "Security policy enforcement in the OSGi framework using aspect-oriented programming," in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*. IEEE, 2008, pp. 1076–1082.