Development of A General Virtualization Approach for Energy Efficient Building Automation

Daniel Kretz[*1], Tobias Teich[2], Marek Kretzschmar[3], Tim Neumann[4]

Academic Department of Economics, University of Applied Sciences Zwickau

Dr.-Friedrichs-Ring 2a, 08056 Zwickau, Germany

[*1]daniel.kretz@fh-zwickau.de; [2]tobias.teich@fh-zwickau.de; [3]marek.kretzschmar@fh-zwickau.de; [4]tim.neumann@fh-zwickau.de

***Abstract*- Networking and automation are fundamental for the implementation of intelligent and energy saving systems in home environments as well as comprehensive smart grids. An enormous variety of available technologies, devices and bus systems create a heterogeneous landscape that is causing a great challenge for the development of versatile and secure software solutions. This article presents our general virtualization approach as a basis for the development of reusable and flexible software components.**

Keywords- Building Automation; Virtualization Architecture; Virtual Devices; Energy Efficiency

I. INTRODUCTION

Progressing climate change and the limited availability of fossil energy and natural resources are global challenges that require new technologies and innovative ideas with sustainability. Hence in 2006, the German federal government has announced the "High-Tech Strategy 2020" as a national master plan defining research demands in different application areas. There are several research issues like the realization of a carbon dioxide neutral, energy efficient city adjusted to climatic conditions as well as an intelligent reorganization of the energy supply. Further objectives are to support a self-reliant life at an old age, sustainable mobility, security, as well as internet based services [1].

Considering the energy efficiency aspects within a city or a city district we inevitably address the research area of smart grids. Ref. [2] defined that a "smart grid is the use of sensors, communications, computational ability and control in some form to enhance the overall functionality of the electric power delivery system". Main objectives are to automatically apply intelligent and to optimize control mechanisms for continuously adjusting the overall system or distinct parts of them. Consequently, automation should facilitate more efficient energy consumption, an improved maintainability, and reliability that consequently results in an overall cost-optimization with environmental friendliness.

Other important aspects besides the reduction of costs, which can be achieved by intelligent automation, are an increased living comfort and the support of a self-reliant life especially for elderly people, particularly addressed as smart home.

Considering not only a single home for automation but also complex buildings within complex districts as parts of a smart grid we can expose huge potential for applying energy efficient optimization as presented for performing the dynamic hydraulic balance in domestic buildings [3]. Furthermore, residential buildings are not only energy consumers anymore. Nowadays, they can also produce and store energy locally e.g. by installing photovoltaic or solar plants and energy storage components. Modern devices involved in the outlined application areas, provide communication interfaces and network support today. Fundamentally, the current development realizes the intention of Ubiquitous or Pervasive Computing which addresses "computer systems infusing the physical world and human and social environments" [4].

All in all we have a wide diversity of application areas, components and smart devices providing computational opportunities to develop intelligent systems today. In fact, the variety of applicable components and devices similarly causes a great challenge for the development of flexible software components for controlling and optimizing their interaction. We have a heterogeneous world of similar devices possibly with quite distinct functionality from different vendors using several communication technologies and protocols like DALI, Meter-Bus, Modbus, BACNet, KNX, Zigbee, Powerline, Z-Wave and many more. Consequently we require a flexible and intelligent solution to achieve a continuous integration from a single device to smart home and finally smart grids.

This paper gives an insight into our developed solution for energy efficient building automation in view of the future integration within an entire district. Therefore we have organized this article as follows. First, we present our general virtualization approach in section two. Afterwards, we provide a detailed insight into our abstract data structures for a general component virtualization within a heterogeneous world and discuss several design intentions in section three. Finally, we provide a short overview about related works in section four.

II. VIRTUALIZATION ARCHITECTURE

Functionality of installed hardware devices like sensors and actors as well as their interconnection and communication with each other provide a fundamental basis for the realization of an energy efficient automation solution. Traditionally, we can install and configure automation devices directly, to increase comfort or to save energy in the private home area. Higher energy saving potentials reveal

the consideration of complex domestic building instead of single homes. In the traditional way, this requires that devices can directly exchange information with each other by communicating over the same bus system and using an identic protocol, which means they are speaking the same language. Otherwise, cost-intensive converters and additional configuration efforts are necessary for the interconnection.

The implementation of equal or similar functionality in different accommodations requires redundant and intensive programming efforts. Furthermore, manufacturer-specific solutions are limited concerning their capabilities of automatic control and optimization as well as the implementation of complex monitoring processes. Especially focusing the process optimization, settlement scenarios and monitoring solutions within the scope of smart grids, we require the development of flexible software solutions in general. There exists a potential risk of directly involving manufacturer-specific devices and properties directly into functions and algorithms during individual software development for automation technology, or functions are adjusted on specific bus systems. Consequently, this causes non-reusable software components.

An eventual installation of new devices e.g. from different vendors with similar functionality or integration of new bus systems and communication protocols may cause an unpredictable refactoring effort of any existing software. Consequently, it is impossible to develop general software services which could interact with any arbitrary type of hardware like data recording, operation monitoring or device-controlling.

A. Overview

First of all, the development of a flexible software solution requires knowing objectives and views from different involved actors, as shown in Fig. 1. Initially there is a main focus on the client or customer, who is defining his purposes concerning increased comfort or energy saving aspects. An optional consultant could be included e.g. for energetic aspects. Finally, the contractor is responsible for installation, implementation and realization of individual customer requirements.

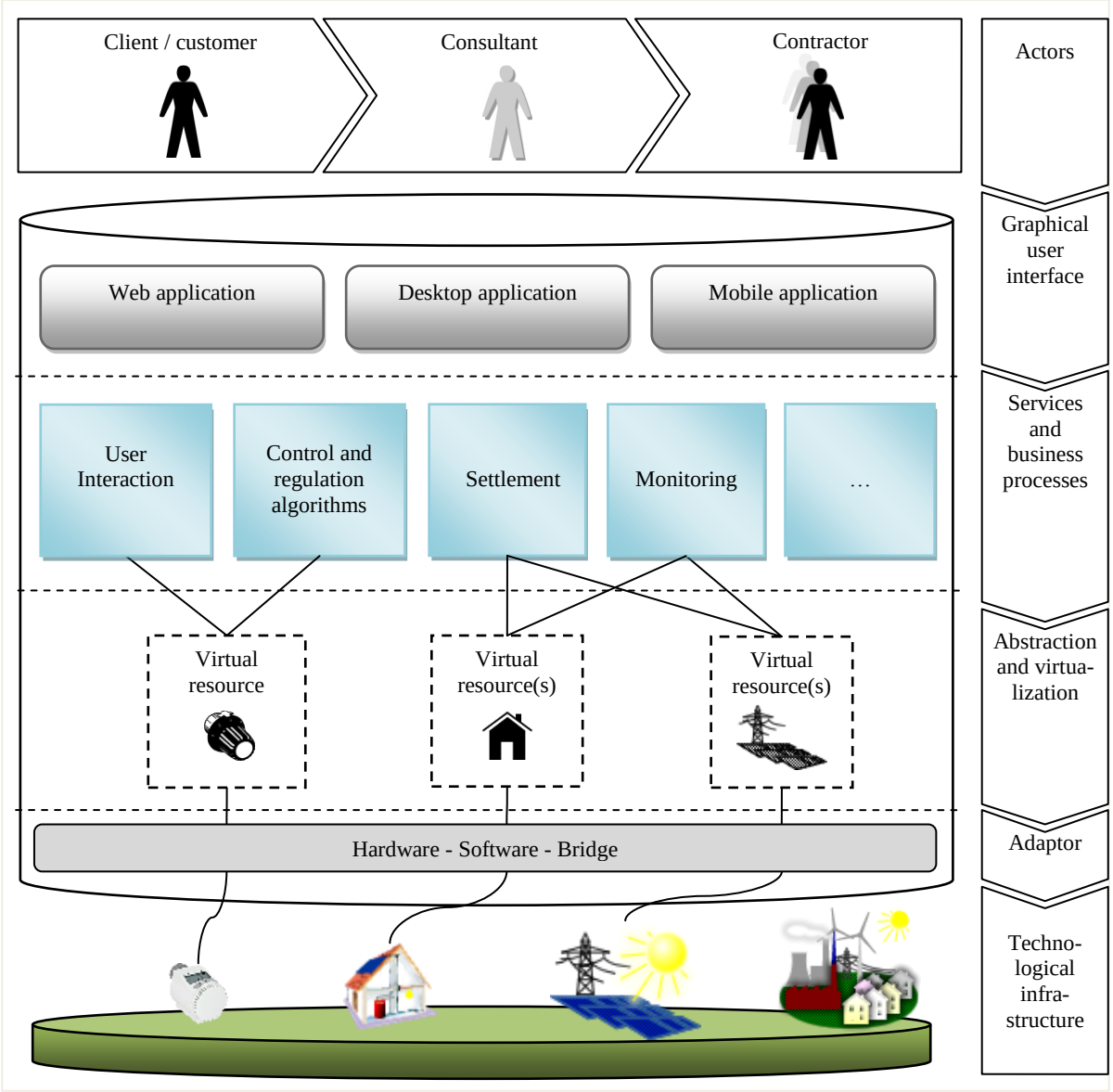

Fig. 1 Virtualization architecture

On the next layer in Fig. 1, a graphical user interface provides access to different services e.g. for user interaction, monitoring or settlement processes. Services can be accessed by desktop, web or mobile applications. The flexibility of our architecture is resulting from the application of an OSGi platform. An OSGi framework is a dynamic Java based application platform which supports the modularization of software by a component model and a service registry as well as by restrictive implementation requirements. It supports the collection of services and service implementations into bundles as well as a dynamic, individual and demand-based software configuration in a modular way [5].

Furthermore, if we want to develop application logic that should be flexibly used in different system environments with distinct automation technology; we completely have to decouple service implementation from underlying hardware devices and bus systems. Therefore, we describe devices in an abstract form and map them into a uniform representation, as illustrated as virtualization layer in Fig. 1. Additionally, abstract devices should not contain any functionality but behave like pure data structures. Any decision or interaction logic is implemented by services and business processes. Finally, our Hardware-Software-Bridge contains adaptors which are responsible for the transformation of vendor specific hardware components from the technological infrastructure into our device virtualization layer and vice versa.

B. Abstraction and Virtualization

Completely decoupling of embedded application logic from their real physical hardware component, the bus system, or hardware interface, is fundamental for realizing flexible software components. Consequently, we need a uniform interface for communicating with hardware devices and a generalizing data structure for the description of any arbitrary component within the current system. Thus it is required to abstract and unify considerable properties of specific device categories. Therefore we have to map device specific attributes and parameters into generalized data structures of virtual proxy objects. This transformation process is done by hardware-specific adaptors. Adaptors are realizing a uniform connection between hard- and software by retrieving and mapping device properties as well as submitting information from virtual proxy objects to them.

Another positive aspect is achieved by treading devices as data structures without any functionality. Because of the extraction of any device specific application logic into the service layer we can completely abandon any time-consuming hardware programming which has to be done redundantly for similar types of devices. The only remaining start-up process is a device initialization which supports the communication and data exchange with virtual components as well as optional emergency programs, if the software crashes or is unattainable based on technical reasons. To connect different bus systems and to unify different device types we require hardware specific device adaptors which are similar to device drivers of an operating system. Consequently, those adaptors realize the Hardware-Software-Bridge as shown in Fig. 1. The adaptor development is necessary once for every device type because they transform device specific properties and pieces of information into standardized parameters of virtual proxy objects. Implemented adaptors contain only communication and transformation logic without any device specific application logic.

C. Services and Business Processes

By virtue of generalizing the specific hardware components from the technological infrastructure into standardized virtual objects, we can develop completely reusable services as well as business processes. Services can now realize the same functionality for any arbitrary device of identic category that was traditionally redundant and individually programmed into hardware devices.

Besides the user interaction, services also implement control and regulation algorithms as well as complex settlement and monitoring processes. Furthermore, they decouple virtual components completely from any communication aspects. Virtual proxy objects only represent the current state of real physical components from the technological infrastructure or submit value changes to them.

A service which is interacting with devices can be dynamically connected to virtual device instances and use them as parameter input or to output results. Hence a service is completely responsible for controlling the communication between devices. Additionally, services can access the OSGi service registry to communicate with other services or use middleware platforms to exchange information in distributed environments. Instead of the direct bus communication and control, we can integrate security mechanisms in the software layer like encryption technologies for remote communication as it is urgently required for communication in smart grid environments.

D. Configuration

Decoupling device and bus specific communication logic from service implementations by transformation into a uniform data representation supports the development of reusable and flexibly configurable software components. In fact, our virtualization approach causes a higher configuration effort.

First of all, we have to initialize the devices of the technological infrastructure with a minimum of required functionality and set them up for bus communication. Furthermore, virtual adaptor instances need to be associated with the real addressable physical device for communication and data transformation. Finally, services have to be connected with concrete virtual objects to retrieve input and submit output parameters.

Basically we can derive each configuration layer from each other. Therefore we have to define required services and available hardware devices. Furthermore, we need information about the building topology that could be retrieved from CAD drawings. Afterwards the configuration can be generated completely automated with defined default scenarios.

III. Virtualization Approach

The development of very general mapping of physical devices and their current state representation is the main objective for our virtualization approach. Hence we can develop services that are applicable within different environments.

E. Virtual Components and Observer Pattern

Initially, we consider virtual devices as specializations of virtual components. This aspect is modeled as inheritance between the abstract class virtual component and virtual device as illustrated in the UML class diagram in Fig. 2. A virtual component is primarily used to implement the observer pattern by using realization of the interface virtual component listener. This behavioural pattern is required to inform registered listener instances about update changes of the virtual component [6]. Fundamentally, we required this pattern to actively inform implemented services and functions about externally triggered state changes. While processing information, communication errors could raise, especially during data exchange between virtual devices and technological infrastructure. To forward details about occurred errors, we have derived listeners from the exception handler interface.

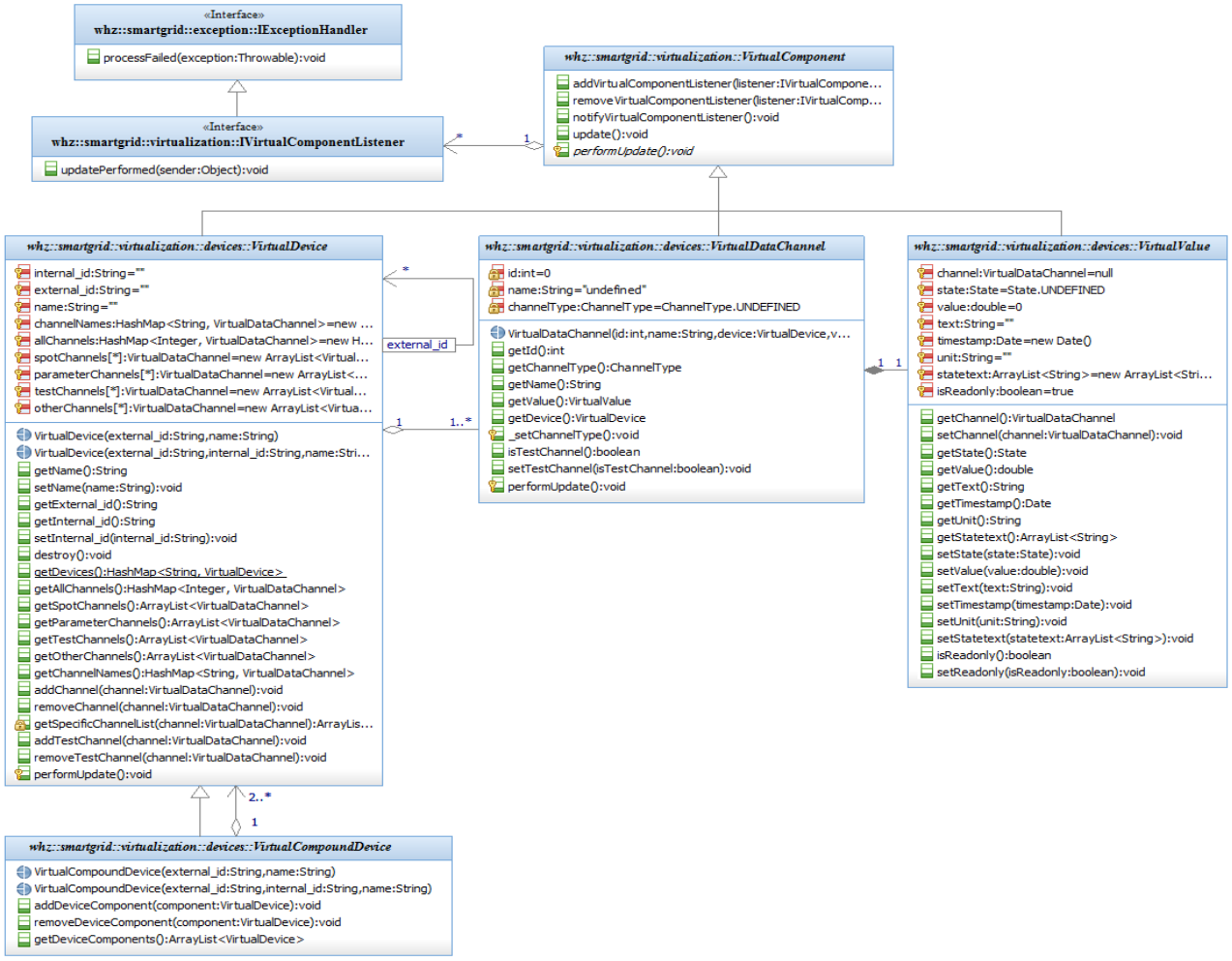

Fig. 2 Description of virtual components

F. Abstract Device Mapping

Within the virtualization layer we define two identifiers for devices. Initially, we require an internal identifier to directly address a physical device on the technological infrastructure. The selected character string is flexible enough to represent different address types like a simple bus address, a specific device handle as well IPv4 or IPv6 network addresses. Additionally, there is an external identifier which is a globally unique device identifier that supports unambiguous virtual component identification as required by smart grid applications. Thus services cannot only realize local functions but also interact globally within a distributed environment.

Multifunctional devices which aggregate functionality of different single devices are mapped as a virtual compound device with the abstract class definition virtual compound device. The advantage of this distinction is that we still map the functionality as single devices in a standardized manner. Furthermore, the aggregating instance provides the ability to define new properties which result from the combination of the single devices.

G. Data Channels

A general and abstract mapping of real devices into virtual devices requires a uniform representation of their properties. Virtual devices always represent the current state of real components or inform about state changes of each

other. Adaptors of the Hardware-Software-Bridge transform device specific information into a standardized form. Consequently, each service can interpret them in a unique way. Basically we have reduced the device functionality to input and output operations. Input and output properties are mapped as virtual data channels as illustrated in Fig. 2.

The basic idea of this mapping was inspired during the development of wrapper classes for porting a programming library (YASDI) via the Java Native Interface (JNI). It is a library completely written in C language for accessing inverters of photovoltaic plants and energy storages via RS232, RS485 and power line [7]. Data channels determine available properties of devices from the technological infrastructure that could be read or written. An exact association is achieved by the definition of unique channel names and an identification number. The great advantage of this identification is that absolutely different devices of an identic category can be mapped to identic channels. Therefore we have a unique signature for similar devices. Data channels are distinguished as spot and parameter channels. We associate read-only values e.g. temperature value from a temperature sensor, with spot channels. Parameter channels are used for device configuration. Hence they have read and write permissions. Test channels are intended as placeholder for check-up accesses. Finally, other channels are an optional extension mapping of device specific properties which should not be standardized. Because of the defined inheritance from virtual component class we can also observe selected data channels of a device.

H. Virtual Values

Virtual values are directly linked with a virtual data channel and hence define the characteristics of this channel. Because of unambiguous data channel identification we can associate those values with concrete physical device properties. Additionally instances of the Hardware-Software-Bridge as well as functions of the service layer can interpret those values in a uniform way.

Numerical values are mapped to the double field called “value” of virtual value instances. The underlying domain of the value type is derived from the channel name and optionally specialized with the unit field. Most degrees of freedom are provided by the additional text field for mapping any arbitrary information like Boolean, Byte and any character sequences. Additionally there are state details about the value like the timestamp of last modification or current processing state.

I. Instantiation and Update Handling

Creating instances of the virtualization layer is implemented with the fabric method pattern which belongs to the category of creational patterns [6]. Base functionality for creating new instances is provided by the abstract class virtual device factory. Dependent on the complexity of the virtualized system and the required initialization effort, the factory either works synchronous or asynchronous. State changes during asynchronous processing are notified by the observer pattern again. This is realized by virtual device factory listener interface implementations and notifies about state changes and completed object creation.

Additionally this interface has to be an exception handler because object creation could already raise errors. Implementations of a class virtual device factory create virtual components from given configurations and connect them to the real physical device.

Instantiation errors could be caused by configuration failures or initial communication failures with the technological infrastructure. Adaptor instances of the Hardware-Software-Bridge are hardware specific data transformers that can be associated one-to-one to a real physical device. Consequently the adapter logic is directly appended to a virtual device by inheritance. After successfully instantiating the virtual proxy objects, it is necessary to update their data channels with corresponding physical devices.

The realization of this process is strongly dependent on the physical device and the bus system. On the one hand there are devices and bus systems that require an active request of current data. This pattern is called pull model because the receiver is responsible for actively retrieving information [8]. Fundamental threading functionality for cyclic information requests are provided by an update manager as illustrated in Fig. 3. On the other hand, physical devices can actively inform about state changes via the bus connection. This is called push model and hence the receiver has to wait for state changes after connection initiation.

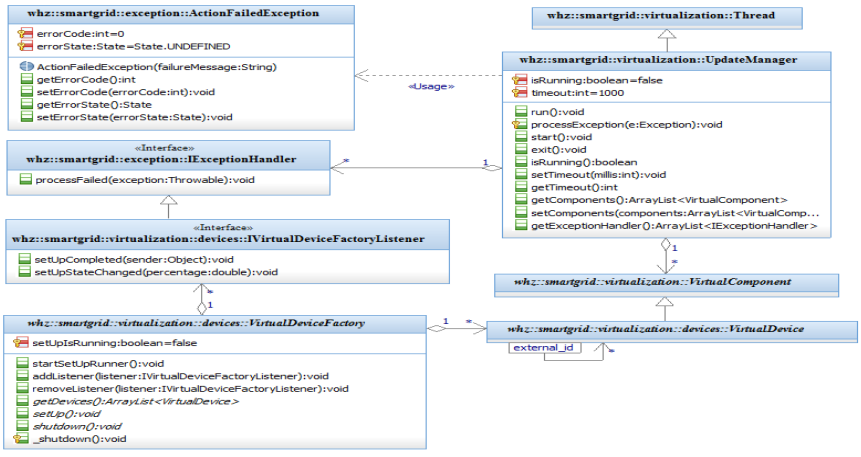

Fig. 3 Instantiation and update handling

IV. RELATED WORK

Connecting heterogeneous technologies and software landscapes is a common issue. Solutions like CORBA or web services realize integration independent from any system, platform or programming language by transforming application specific information into a standardized form that can be interpreted by any involved system [9].

Ref. [10] discussed an interface-based approach for the virtual device abstraction by defining common methods for specific device categories. Abstract classes also define methods that must be implemented by specializations. Additionally we provide basic functionality which is generally required like a virtualization framework.

Ref. [11] elucidated the importance of virtualization for implementing security concepts using the example of virtual machines in smart grid environments.

V. CONCLUSION

This article presented our approach for abstracting the complexity of a heterogeneous automation landscape by mapping device properties into a generalized data structure. Our approach for device virtualization supports the development of flexibly configurable and reusable software components for energy efficient control and monitoring.

An important, but rarely mentioned aspect is the standardized mapping of channel identifiers for identic or similar device categories. The development of specific factory classes requires standardized templates for associating unique channel names. We are currently developing a database which defines the signature for specific types. Channel names should be based on several standards e.g. IEC 61400 or 61850 which define common parameters and communication objects for smart grid environments.

REFERENCES

[1] Federal Ministry of Education and Research (BMBF), Hightech-Strategie 2020 für Deutschland, Bonn, Berlin, 2010.

[2] C. W. Gellings, The Smart Grid: Enabling Energy Efficiency and Demand Response, Crc Pr Inc, 2009.

[3] T. Teich, D. Szendrei, M. Schrader, F. Jahn and S. Franke, „Feasibility of Integrating Heating Valve Drivers with KNX-standard for Performing Dynamic Hydraulic Balance in Domestic Buildings", World Academy of Science, Engineering and Technology, Issue 49, 2011, pp. 367 – 372.

[4] S. Poslad, Ubiquitous Computing: Smart Devices, Environments and Interactions, John Wiley & Sons, 2009.

[5] J. McAffer, P. Vanderlei and S. Archer, OSGi and Equinox: Creating Highly Modular Java Systems, Addison-Wesley, 2010.

[6] E. Gamma, R. Helm and R. E. Johnson, Design patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley Longman, Amsterdam, 1995.

[7] Yasdi API Dokumentation (YASDI-TDE084020), SMA Solar Technology AG, 2008.

[8] F. Buschmann, K. Henney and D. C. Schmidt, Pattern Oriented Software Architecture: On Patterns and Pattern Languages, 5th ed., John Wiley & Sons, 2007.

[9] D. Kretz, J. Militzer, T. Neumann, C. Soika and T. Teich, "CORBA based Architecture for Feature Based Design with ISO Standard 10303 Part 224", in Proc. Digital Enterprise Technology 2011, Athens, Greece, pp. 29 – 38.

[10] T. Bodhuin, G. Canfora, R. Preziosi, and M. Tortorella, "An Extensible Ubiquitous Architecture for Networked Devices in Smart Living Environments", Embedded and Ubiquitous Computing – Lecture Notes in Computer Science, 2005, Volume 3823/2005.

[11] B. Graham, Securing Smart Grid Devices Using Virtualization to Protect the Grid, Intel Corporation, 2010.