A model based development approach for building automation systems

Björn Butzin, Frank Golatowski
Institute of Applied Microelectronics and Computer Engineering, University of Rostock, Faculty of Computer Science and Electrical Engineering
Rostock, Germany
Bjoern.Butzin@uni-rostock.de
Frank.Golatowski@uni-rostock.de

Christoph Niedermeier, Norbert Vicari, Egon Wuchner
Siemens AG, Corporate Technology,
Research Technology Center
Munich, Germany
Christoph.Niedermeier@siemens.com
Norbert.Vicari@siemens.com
Egon.Wuchner@siemens.com

***Abstract* — As of today, building automation systems are present in almost any commercial building. They perform climate control, lightning control, access control, surveillance, and quite a few other tasks. As a result of their evolutionary development, building automation systems are divided into separate silos of disciplines that are not well integrated with each other. As of today, a variety of communication protocols, data models and engineering approaches are used by different vendors. Existing standardized building automation protocols as BACnet or KNX allow integration of some disciplines on the communication level but fail to provide means for common description of devices, services and data on the semantic level. This means that building automation applications that span multiple disciplines require a high effort for development, engineering and maintenance. If devices from multiple vendors are integrated in one installation, a set of different engineering tools and vendor-specific knowledge is required. In the ITEA "Building as a Service" (BaaS) project we try to overcome these deficiencies and define a common way to develop, engineer, commission, operate and maintain building automation systems following a service oriented approach. The whole process will be supported by semantic models to reduce costs and time-to-market, which is a quite new approach. In this paper we will present the current state of the work with special regard to domain modeling and model driven processes that are currently being specified for the BaaS platform.**

Keywords—building automation; service oriented architecture; data model; semantic model; domain specific language

I. INTRODUCTION

Today's commercial buildings have different automation, management and information systems built-in and running. These systems are responsible for controlling different building functions like heating and cooling, lighting, access control, surveillance and many more. All the data collected and all the possibilities of the built-in devices are separated in silos of disciplines [1]. Even within a discipline it can be hard to use products of more than one vendor. There are several reasons for this. The first reason is the variety of, partly standardized data models that can be used e.g. BACnet, KNX, OPC-UA and many more. In [2] an overview of the state of the art in applied BAS communication systems can be found. Furthermore there are different communication interfaces like AN, WS + oBix or EEBUS and additionally the vendors of building automation solutions also have different engineering approaches. With all these differences it requires a high engineering effort to integrate all or at least some devices of different silos or vendors [3]. The building automation technologies that are used today lack a common semantic model that allows the description of the purpose and features of devices, services and data independent of particular protocols, data models and vendors. This lack of a semantics model and ontologies for building automation equipment has recently been recognized; first research activities to solve the issue are under way [4].

II. VISION

Our vision of the "Building as a Service" (BaaS) project is an evolution of building automation systems that are using standard IT infrastructure and protocols in a service oriented manner to become interoperable with each other [5]. The goal of the project is to simplify the development, engineering, commissioning and maintenance of multi-discipline, multi-vendor building automation installations, and thus reducing cost and time-to-market. The approach must be able to facilitate integration of legacy building automation components and systems in a seamless way. The use of the standard IT infrastructure and protocols together with the increased interoperability shall create opportunities for novel value-added applications to be built on top of existing solutions. These applications can take advantage of all information collected in the different disciplines and achieve new purposes (e.g. goal-oriented building control) by employing advanced mechanisms and algorithms. To reach this goal we need uniform data models as well as semantic knowledge about each building automation device that can be used within current and future buildings.

III. CHALLENGES

There are mainly three challenges that we need to face and that are described below:

A. Overcome traditional silo structures

Between the different domains there are evolutionary grown gaps that prohibit a comprehensive data analysis and therefore

the creation of new knowledge. The systems within a certain domain are engineered for nothing more than its purpose using domain specific attribution, naming and specialized protocols. Interactions between different silos is still possible under these circumstances but requires a high effort for creating and maintaining custom adapters or gateways to bridge the gap between incompatible protocols and vendors.

B. Avoid proprietary engineering approach

With a proprietary engineering approach many standard solutions have to be done multiple times with respect to technologies and languages. Moreover, using proprietary engineering approaches does not only mean to reinvent the wheel. It also causes a lack of standardized interfaces. This in turn makes these systems inflexible in maintainability and extensibility. Creating new applications would require the vendor to be involved in the development process or at least the interfaces, configuration- and system models have to be known, but this often isn't the case. Furthermore to engineer, commission and operate systems of multiple technologies requires a host of different tools and knowledge and is considerably more error prone. The overhead could be reduced by evolving a common approach for the development and engineering of such systems. Then a common set of tools could be used that is independent of vendors. At the same time, the complexity of the engineering task and the time required to perform it will be reduced.

C. Introduce common semantic model

Currently BA systems need to be engineered and commissioned by highly qualified individuals that are experts in the respective BA system domains. This is due the lack of semantic descriptions as well as a common information model. Semantic descriptions will enable the automatic discovery and binding of devices and services and thus reduce the manual engineering effort. The lack of a common information model prevents the easy integration of currently available devices in the IT infrastructure.

IV. APPROACH

Our mission is to face the mentioned challenges in order to have an interoperable way to develop, engineer, commission, operate and maintain building automation systems with reduced cost regarding time and expertise. There are five major areas that need to be covered. These areas are:

A. Common communication abstractions

To have a common ground for developing, maintaining and extending BA systems, the communication has to be modelled following a common abstraction. The common communication abstraction describes the available communication primitives independent of specific technologies and protocols. This abstraction can be used to specify the interface of a service and to utilize arbitrary Web service communication protocols during operation. This approach follows the well-known architectural principle of "separation of concerns" that allows to perform the decision to use a particular technology at a later point in time or delegate it to another stakeholder.

B. Comprehensive data models

Comprehensive data models allow the sharing of data among different disciplines and help to overcome the traditional silo structures in current BA systems. Comprehensive models facilitate the creation of new applications using sensors and actors of different domains and vendors and therefore open new business fields that are not available yet. Together with the data models the associated data representations need to be described in a protocol independent way.

C. Semantic models

To overcome the traditional silo structures in current BA systems the devices and services not only need to communicate. They also need to understand each other. Therefore the resulting data models and their representations need to be enriched by semantic information. By providing semantic information data sources and consumers could be able to search for the required data without having someone to preselect or configure this relation. Additionally the resulting system could have the ability to perform semantic reasoning on the available data. The reasoning will allow extracting or combining data to get certain information. This allows to dynamically reconfiguring BA systems in order to add new functionality or to optimize parts of it. Additionally the overall systems robustness and fault tolerance is increased, because faulty system parts could be replaced by semantically identical or similar parts. To achieve this, the BaaS platform is planned to maintain and extend an ontology of the BA domain.

D. Tools for legacy integration

Most of the building automation systems are planned, engineered, and commissioned for several decades of operation time. Therefore already existing systems cannot be left behind. New buildings could be equipped with this new technology but most of the buildings can be only extended and upgraded. Thus, the integration of legacy BA installations is an important feature of BaaS. The legacy integration should be designed to be able to embrace the systems, data models, and interfaces of solutions that are currently available on the market. Besides the development of gateways that provide access to legacy systems, the data points in legacy devices have to be described in a semantic way. This description with together with the needed protocol transformation allows the seamless access to legacy BA systems in the BaaS approach.

E. Development tools

Furthermore tools for the development, engineering, integration and commission of services and devices need to be provided. Development tools that make use of the BaaS abstractions and semantic annotations reduce the required time and system knowledge to create solutions within the domain. It also ensures to stick to the mentioned constraints due to the communication abstraction, data model, semantic model and legacy integration. For that reason the development tool should hide these aspects from the developer and turn the

focus on the service structure, privacy and security constraints, as well as the business logic.

Our idea is a service oriented platform for the building domain consisting of a runtime environment and a software development kit. The runtime environment will provide common capabilities to BaaS services. For instance, these are functionalities that manage services, provide access to low level resources of the hardware, or ensure the availability and robustness of service execution.

For the software development kit domain specific editors and tools based on a domain specific language will be developed. This shall provide the ability to model services, annotate devices, services and data points with semantic information and hide the underlying technologies for communication and configuration. In a next step the tool chain uses these domain specific models to generate at least service skeletons, only missing the business logic or service compositions. Furthermore tools and mechanisms for service deployment, configuration and maintenance will be provided. For that purpose the inherent semantic information can be used to perform automatic advertisement and discovery of services, devices, and data points at engineering, commissioning and operation. To integrate legacy devices, the same tools shall be used. In contrast to native services, legacy devices require gateways and related services to communicate in the BaaS system. These shall be developed once for each technology and then be configured for the designated purpose by providing semantic annotation.

The capabilities of the BaaS platform will be demonstrated and validated by a set of services covering relevant examples of BA functionality.

V. Related Work

There are several projects dealing with similar problems regarding the Internet of Things and building- or home automation. The recently established Open Interconnect Consortium [6] is focusing on defining a common communication framework for wireless data transmission and

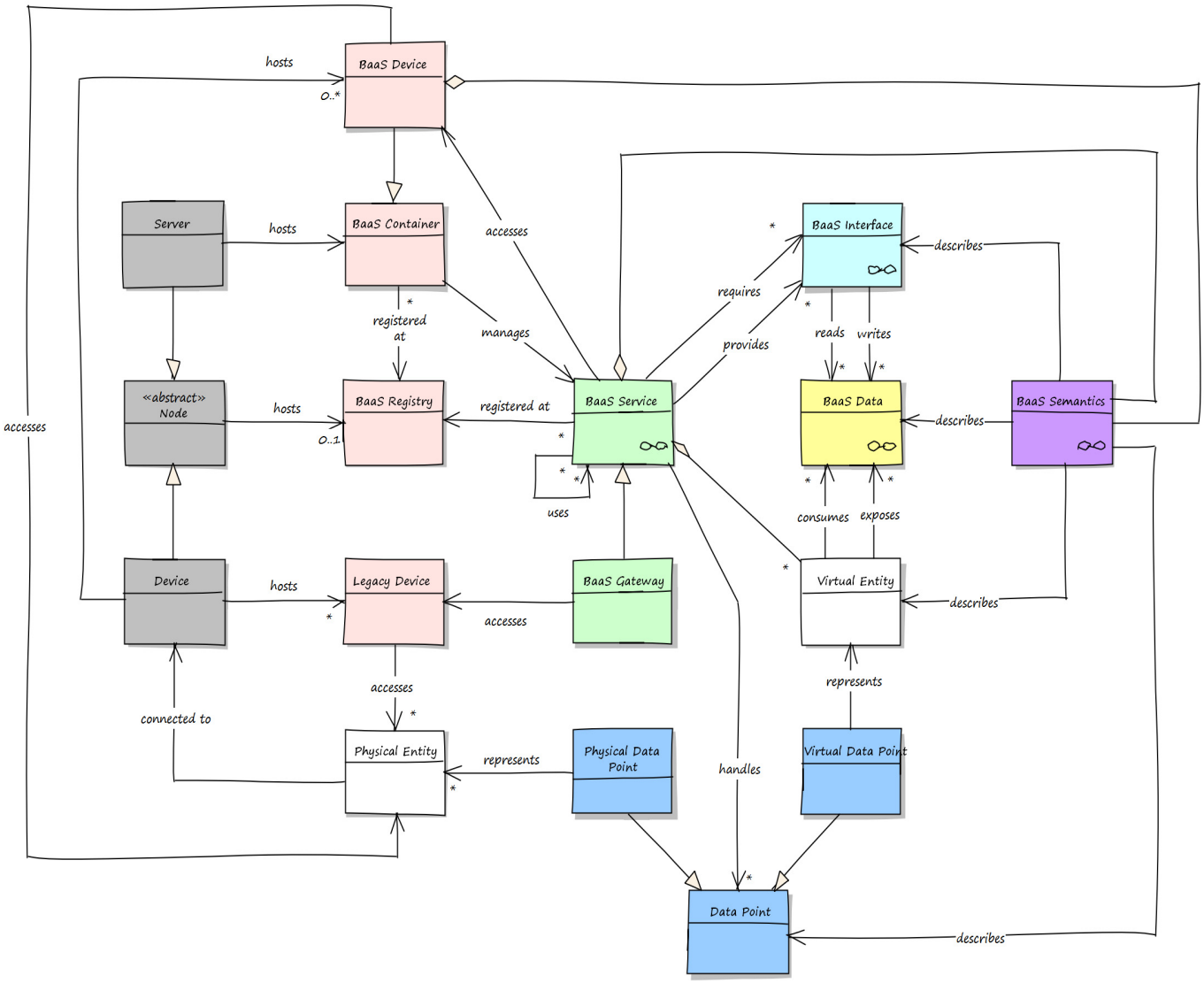

Figure 1 BaaS Domain Model

information management. The AllSeen Alliance [7] and Connected Living [8] are focused on home automation. The Thread Group [9] is also targeting at the home sector but uses mesh network technologies. The EEBus initiative [10] tries to evolve EEBus towards a networking concept for almost all smart devices. The OM2M project[11] and the OData [12] standard are more generally looking at machine to machine communication without any special regards to a domain. The BaaS Project [13] of the seventh framework programme (FP7) is, in contrast to our project, aiming to optimize the energy performance of buildings. A taxonomy of tags for building automation domain is developed in the project Haystack [14].

VI. Domain model

At the current state we are elaborating a domain model for the building automation system domain and the BaaS vision. A domain model in our case is a conceptual model and structural view of important topics related to a specific problem. It describes participating entities, their attributes, and relationships as well as underlying concepts and constraints. We want to use this domain model as a clear depiction to ensure that all stakeholders can agree on the scope and meaning of the concepts in the building domain.

In the BaaS Domain Model, as depicted in Figure 1, Data Points are used to describe input and output functions in the building automation domain. According to EN ISO 16484-2 [15], 3.61 such Data Points are semantically completely described. This description may contain functionality, parameters, and variables representing current state but also characteristic curves, units, and allowed ranges. Data Points may be Physical Data Points or Virtual Data Points. Physical Data Points are related to existing hardware or physical actuators/sensors. Virtual Data Points can be derived or aggregated functions. Data Points represent the building automation entities in the BaaS domain model.

A core concept in BaaS is the BaaS Service. A BaaS Service handles Data Points and contains Virtual Entities which are a digital representation of building automation functions, i.e. Virtual Data Points, in the BaaS system.

A BaaS Service is accessible and communicates via its BaaS Interface, i.e. it provides and requires the BaaS Interface to communicate with other BaaS Services. The BaaS Virtual Entity consumes and provides BaaS Data, which is read or written over the BaaS Interface.

A BaaS Service has a semantic description, which is contained in the BaaS Semantics. The BaaS Service is described by ontologies describing the Virtual Entity and the semantic meaning of the related Data Point. Further ontologies describe the data types of BaaS Data and the semantic of the BaaS Interfaces of a BaaS service.

BaaS Services may be used and may use other BaaS Services.

The physical world is reflected in the BaaS Domain model by the concept of an “abstract” Node. Such Node can be a Device that is associated with a Physical Entity, i.e. a sensor or actuator. Devices might take the form of Legacy Devices, e.g. BACnet or KNX devices, or the form of native BaaS Devices. A Node can also be a Server that hosts a BaaS Container which manages BaaS Services, i.e. a BaaS Service is deployed in a BaaS Container. This approach allows running BaaS services on sensors or actors if they have enough resources and otherwise outsourcing it to more powerful systems that are not directly associated with a physical entity. BaaS devices are always accessed via BaaS Services. A BaaS Device provides the interface between real hardware, such as sensors, and BaaS Services. A BaaS Device is semantically described by ontology in the BaaS Semantics.

BaaS Services and BaaS Containers resp. BaaS Devices are registered at a BaaS Registry which is hosted on a Node. The BaaS Registry is used to query and resolve BaaS entities in the different phases of the BaaS lifecycle. The BaaS domain model does not limit the number of BaaS registries and therefore centralized or decentralized architectures can be derived.

Legacy Devices are accessed by a BaaS Gateway which is a specific type of BaaS Service. From the view of a Baas Service there is no difference to another service. From the view of a developer a BaaS Gateway provides a connector to legacy systems such as BACnet or KNX. Additionally the data will be connected to semantic models by the system engineer in order to be accessed by other BaaS services.

A. Relation to IoT-A Domain Model

The FP7 project IoT-A developed an Internet of Things Architectural Reference Model (IoT ARM). The IoT domain model as part of the IoT reference model [16] defined in the IoT-A project, influenced the design of the BaaS domain model. Yet, some significant differentiations have been made to reflect the specifics of BaaS.

Main differences consist in the addition of Data Points, which provides the building automation language and semantics between Physical and Virtual Entities. Further, the focus on semantic descriptions is emphasized by the introduction of the BaaS Semantics which is a placeholder for all ontologies needed in the BaaS system.

Similarities are in the modelling of hardware components, even if the focus is more directed on the deployment of BaaS components in the BaaS domain model. Resources are depicted as BaaS Interface and BaaS Data putting more emphasis on the description of the resource, compared to the location in IoT-A. Finally, the User is not modelled in the BaaS domain model, but the Service in IoT-A and the BaaS Service are both a User.

VII. Domain Specific Language

The whole model is not only for the depiction of the used names, concepts and relations in the building domain it also serves as foundation for the usage of domain specific languages and ontologies related to the building automation domain. A domain specific language is a formal language that is able to describe the problems, solutions, entities and relationships for a particular domain in a specialized way [17].

Therefore it is in contrast to a general purpose language that is able to do that for any domain but in a general way.

VIII. Combined Approach of Domain Specific Languages and Ontologies

For each phase we intend to equip the respective stakeholder (software engineer, system engineer, domain engineer etc.) with domain specific language editors [17] making use of ontologies also in order to fulfil some of the specific tasks of each phase. In the following we only sketch the approach for the development phase of new services.

The software engineer aims at creating reusable services by using the flexibility of a general purpose programming language. At the same time the approach intends to build in first-level support for higher-level building specific concepts directly into the programming language. The software engineer should be able to refer to semantic descriptions of building specific data points (and possibly their possible data formats) a service is supposed to provide by its implementation. The integration of legacy building automation systems should be captured by semantic descriptions of the data points of the installed legacy building automation system as well. On the other hand, the software engineer does still want to use state-of-the-art Integrated Development Tools being used to features like syntax coloring, code completions and support for refactoring). Textual domain specific languages and frameworks like Meta-Programming Systems which we use naturally include this kind of extended IDE capabilities.

The following picture (Figure 2) depicts the goal of a general combined approach of using a domain specific language editor and ontologies.

The software engineer uses a DSL editor called “Service Creator”. This DSL editor in turn includes query support for semantic descriptions of building specific data points e.g. SPARQL [18]. These data points have been specified by a domain engineer as ontology individuals which are stored as part of a “Data Point Repository”. Thus, the software engineer is able to annotate a new service with references to the semantic description of Data Points it will provide. Possibly, the reference might also include the syntactic data format of the data point which the service will implement.

Additionally, a second “Service Repository” keeps hold of all created services, their descriptions and could also be queried by the DSL Service Creator. This query mechanism is used to search for already created services providing certain Data Points required by a service to provide data of its own supported Data Points. Furthermore, the DSL Service Creator transforms any new service specification into a semantic format to be stored by the Service Registry.

The Service Registry and all its content is meant to be also available for the engineering Phase. Therefore, the DSL editor used by the system engineer is able to search all implemented services, look for specific ones from the development phase and continue instantiating and connecting them for engineering purposes.

The DSL service creation editor allows handling technical concerns like communication styles (request/reply versus publish/subscribe, data-centric versus event-based design), IT communication protocols (e.g. DPWS, CoAP) or security mechanisms to be supported by the software engineer as well. Some of these concerns also have to be part of the service description to be used by the following phases. For example, the system engineer needs to know which communication protocol a service supports since only service instances of services using the same communication protocol and style can be connected to each other at engineering time.

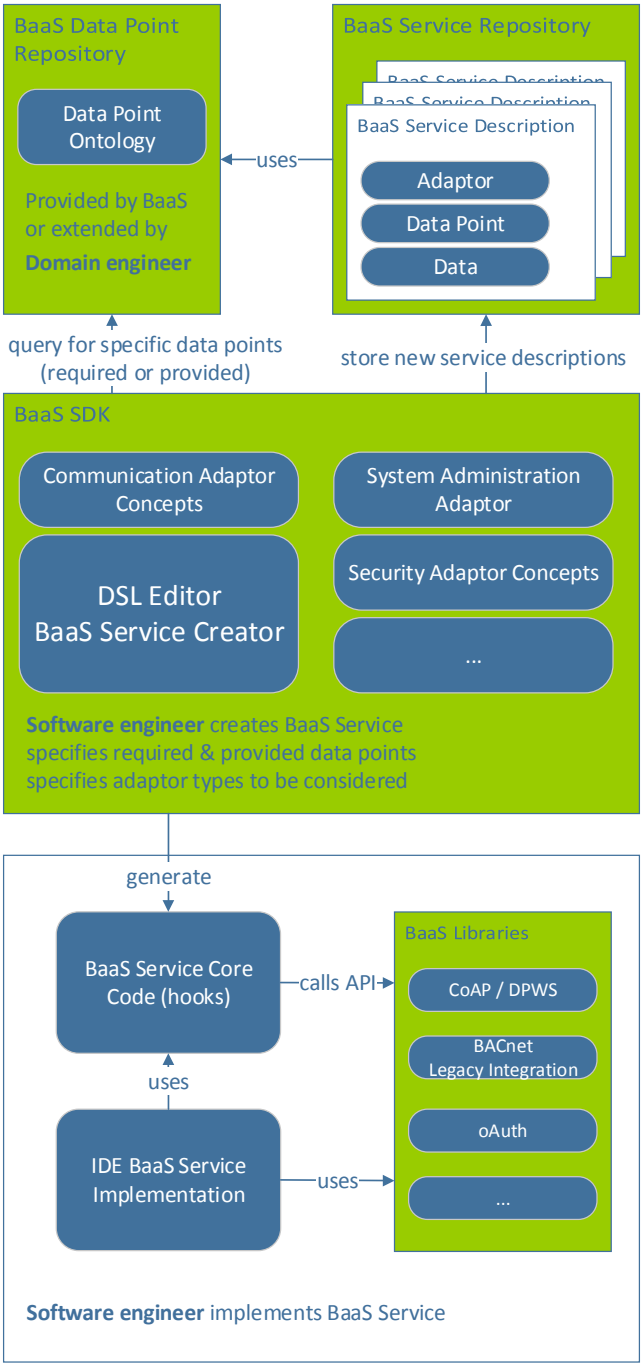

Figure 2 General combined approach of using a domain specific language editor and ontologies

The software engineer specifies these technical characteristics of a service and the DSL Service Creator generates code skeletons of the service. The generated code leverages certain libraries containing base functionality of communication styles/protocols and other technical concerns.

As an example we currently working on a domain specific language to exemplarily describe web services without care about underlying technologies like xml and the web service description language or to be used programming languages. It is based on basic web service implementations for providing and consuming just simple data like sensor readings and the BaaS domain model. The approach aims to have the service structure easily visible in the code and reduce the amount of repetitive implementation work. As a exemplary test bed we generate services following the devices profile for web services (DPWS) [19] standard, using the JEMEDS (Java) stack[20]. In the future also c, python or .net implementations could be generated out of the same model and different communication mechanisms could be used. If the basic model for services is stable there is the idea of including semantic meta information into the model as a first approach to define the provided and consumed data of a service.

IX. Conclusion and Future Work

We briefly introduced the vision and challenges of the "Building as a Service" project to overcome traditional silo structures of current building automation systems by means of used protocols, models and tools and ease the development, engineering, commissioning, operation, and maintenance of building automation services. To achieve this, BaaS will create development tools to model semantic annotated building automation services with communication and data representation abstractions. The tooling will generate deployable code from these models which reduces standard coding tasks and the required knowledge of the used technologies. The generated code will be based on web service communication protocols and corresponding data models. Semantic descriptions may be used for the discovery of needed information in different phases of the lifecycle of the BaaS platform.

We presented the current domain model of BaaS and the concept of combining domain specific languages and ontologies to develop services on the BaaS platform. These concepts serve as foundation of the BaaS reference architecture, which will be developed in the BaaS project. As next steps towards the architecture, we will specify the details of the BaaS runtime environment and the BaaS software development kit. The feasibility and benefits of the selected approach and the developed platform will be demonstrated in a real world deployment featuring a set of innovative services running on top of an existing legacy building automation system.

Acknowledgment

This work has been achieved in the European ITEA project "Building as a Service" (BaaS) and has been funded by the German Federal Ministry of Education and Research (BMBF) under reference numbers 01IS13019H and 01IS13019A.

We want to thank all partners in the BaaS project for the stimulating discussions and their contributions to the project. Especially Heiko Krumm, Malte Burkert (TU Dortmund), Marc-Oliver Pahl, Benjamin Hof (TU Munich), Mario Schuster, Armin Wolf (Fraunhofer FOKUS) Franz-Josef Stewing, Ingo Lück (MATERNA), Juergen Maass, Michael Christmann, Michael Peter Schmidt (Kieback & Peter), Andreas Müller, Martin Neubauer (TWT) and many more.

All project partners can be found on http://baas-itea2.eu.

References

[1] M. Kovatsch, M. Weiss, and D. Guinard, "Embedding internet technology for home automation," in *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010.

[2] W. Kastner and G. Neugschwandtner, "Data Communications for Distributed Building Automation," in *Embedded Systems Handbook: Networked Embedded Systems*, Boca Raton, Florida: Editor Richard Zurawski, CRC Press, 2009, pp. 29–1 – 29–34.

[3] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner, "Web Services in Building Automation: Mapping KNX to oBIX," in *2007 5th IEEE International Conference on Industrial Informatics*, 2007, vol. 1, pp. 87–92.

[4] A. Andrushevich, M. Staub, R. Kistler, and A. Klapproth, "Towards semantic buildings: Goal-driven approach for building automation service allocation and control," in *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010, pp. 1–6.

[5] F. Jammes, B. Bony, P. Nappey, A. W. Colombo, J. Delsing, J. Eliasson, R. Kyusakov, S. Karnouskos, P. Stluka, and M. Till, "Technologies for SOA-based distributed large scale process monitoring and control systems," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 5799–5804.

[6] "Open Interconnect Consortium." [Online]. Available: http://www.openinterconnect.org/. [Accessed: 15-Jul-2014].

[7] "Open Source IoT to advance the Internet of Everything - AllSeen Alliance." [Online]. Available: https://allseenalliance.org/. [Accessed: 15-Jul-2014].

[8] "Connected Living e.V.: Home." [Online]. Available: http://www.connected-living.org/. [Accessed: 15-Jul-2014].

[9] "Thread Group." [Online]. Available: http://www.threadgroup.org/. [Accessed: 17-Jul-2014].

[10] "EEBus Initiative e.V." [Online]. Available: http://www.eebus.org/eebus-initiative-ev/. [Accessed: 15-Jul-2014].

[11] "OM2M - Open Source platform for M2M communication." [Online]. Available: http://eclipse.org/om2m/. [Accessed: 15-Jul-2014].

[12] "OData Version 4.0 Part 1: Protocol." [Online]. Available: http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html. [Accessed: 15-Jul-2014].

[13] "About BaaS." [Online]. Available: https://www.baas-project.eu/. [Accessed: 15-Jul-2014].

[14] "Home – Project Haystack." [Online]. Available: http://project-haystack.org/. [Accessed: 18-Jul-2014].

[15] "ISO 16484-2:2004 Building automation and control systems (BACS) -- Part 2: Hardware." .

[16] M. Bauer, N. Bui, J. De Loof, C. Magerkurth, A. Nettsträter, J. Stefa, and J. Walewski, "IoT Reference Model," in *Enabling Things to Talk*, A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, S. Lange, and S. Meissner, Eds. Springer Berlin Heidelberg, 2013, pp. 113–162.

[17] M. Fowler, *Domain-specific languages*. Upper Saddle River, NJ: Addison-Wesley, 2011.

[18] W. The W3C SPARQL Working Group, Ed., *SPARQL Query Language for RDF*. 2013.

[19] "OASIS Devices Profile for Web Services (DPWS)." [Online]. Available: http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01. [Accessed: 16-Jun-2014].

[20] "Stack: WS4D-JMEDS (Java) | Web Services for Devices." .