Engineering and Operation Made Easy – A Semantics and Service Oriented Approach to Building Automation

Norbert Vicari, Egon Wuchner,
Arne Bröring, Christoph Niedermeier
Siemens AG, Corporate Technology, RTC
Munich, Germany

***Abstract*— Building automation systems control many aspects of today's buildings – lightning control, air condition, shading, access control, and surveillance, just to name a few. This diversity and the diversity of related technologies and protocols entails that the extension or integration of building automation systems requires a major effort. To address these challenges, the ITEA3 *Building as a Service* (BaaS) project developed a reference architecture that is based on a service oriented approach enhanced with semantic descriptions that aims at model based code generation and simple integration of legacy devices.**

In this paper, we present an overview of the BaaS reference architecture with a specific focus on the information model and on the envisioned way to take advantage of the semantic descriptions for information filtering, search and discovery throughout the lifecycle of a building automation system.

Keywords— building automation; service oriented architecture; reference architecture; information model; semantic descriptions; discovery;

I. INTRODUCTION

Today, the technologies used in building automation systems comprise a broad variety of devices, communication protocols (e.g., BACnet, KNX, or OPC-UA), and information management services [1]. The engineering of building automation systems is usually vendor-specific. There are no common engineering tools, and the meta-information required for extension of such systems is usually not revealed. In fact, different automation disciplines such as lighting control, HVAC (heating, ventilation and air-conditioning), safety and security are often handled separately. This creates vertical silos of technology stacks within a building's automation infrastructure. In consequence, tremendous engineering efforts are needed to create, integrate or extend building automation systems today [2].

Besides the heterogeneity in communication protocols, there are no shared semantic models among building automation technologies. Current approaches do not provide machine interpretable descriptions of the functionality of devices, services and data independent of particular protocols, data models and vendors. This lack of shared semantic models for the building automation domain has recently been recognized. Although first research activities to solve the issue are under way (see e.g., [3] and [4]), there are still no established approaches.

Thereby, the above described challenges are relevant for the entirety of the *lifecycle* of a building automation system. This is of particular importance, since buildings and all their contained infrastructure are planned and maintained for quite a long lifespan. Hence, IT solutions need to comprehend all phases (i.e., development, engineering, commissioning, operation, and optimization) of a building automation system's lifecycle, which is currently not state of the art. Today, building information modeling (BIM) [5] is a popular way of capturing a digital representation of a building's structure and properties. However, the capabilities of BIM to handle building automation functionalities are still limited.

To finally support all roles interacting with a building automation system (e.g., developer, system engineer, operator) a new approach needs to provide the required tools and functionalities in a way that minimizes the efforts for the user in conducting their particular tasks. For example, this includes the minimization of manual coding efforts to implement control and management services. A novel model-based specification of such services is required. This involves an automated generation of building automation services, which thereby goes beyond existing semantic building automation approaches such as designed in the SCUBA project ([6], [7]). The services itself need to be enabled to link the devices and controllers of a building utilizing Web of Things technologies based on Internet communication and semantic technologies (e.g., CoAP, JSON, RDF) to enable platform independent discovery and interaction.

This work tackles the above described challenges and in this paper presents a reference architecture developed as part of the *Building as a Service* (BaaS) project. Our approach first describes the needed reference architecture including an information model and a dedicated view on the lifecycle of building automation systems (Section II). Second, we analyze and subsequently design needed tools and functional building blocks of a BaaS system (Section III). Finally, conclusions and an outlook for our research work are presented (Section IV).

978-1-4673-7929-8/15/$31.00 ©2015 IEEE

This work has been achieved in the ITEA3 project "Building as a Service" (BaaS) and has been funded by the German Federal Ministry of Education and Research (BMBF) under reference number 01IS13019A.

II. Reference Architecture for Building Automation Systems

To address the challenges of modern building automation systems, the BaaS project defines a reference architecture that provides architectural patterns and a blue print for

- a flexible open building service platform that facilitates the generation and deployment of value-added building services at a considerably lower cost;
- a BaaS data model that provides additional meta-information to simplify the engineering of value added services and applications for the BaaS system and the integration of/with legacy systems;
- model-based mechanisms for analysis, aggregation and transformation of data according to the meta-information provided in the BaaS data model;
- methods for integration of existing and novel sources of information to create a "building information sphere" considering all stakeholders of the building.

A. Architectural Approach

The BaaS reference architecture is described following a 'views and perspectives' approach [8]. The following views were selected to illustrate the architecture:

- The Lifecycle View defines the activities in the different phases of the BaaS lifecycle, the related artifacts (e.g., models, software) in each phase and the transitions from one phase to the other.
- The Information View describes the information model used for the description of the data points, related to the building automation domain and the BaaS services which addresses the IT domain aspects.
- The Functional View defines functional building blocks of the reference architecture and it specifies its responsibilities and relations to other functional building blocks. The functional view captures the interactions of the functional building blocks as triggered by the actors during the lifecycle.
- The Behavioral View focuses on the interaction patterns for the data exchange between BaaS Services.

Finally, the qualitative requirements as cross-cutting issues are addressed in the Security and Dependability perspectives in order to cover all of the relevant aspects of a BaaS system.

In the following, we will provide a more detailed explanation of the lifecycle and the information model.

B. Lifecycle View

In our model, the lifecycle of a building automation system consists of the phases design, development, engineering, commissioning, operation and optimization. As depicted in Figure 1, the system is described by models, descriptions and software artifacts in all phases of the lifecycle.

During the design phase, the data points needed to implement the system are described. In the development phase, this data point model is the basis for modeling and implementing service types, which represent the data points as software artifacts. In the subsequent engineering phase, the service types are instantiated and wired to implement a SOA choreography that allows the services to execute their intended functionality. The composed service instances are then deployed in the field in the commissioning phase. During the operation phase, control information and operation data is collected, which is in turn analyzed in the optimization phase. Depending on the result of the optimization phase, the lifecycle might be entered again in the design, development or engineering phases. In the following, we will describe the path through the lifecycle in form of use cases.

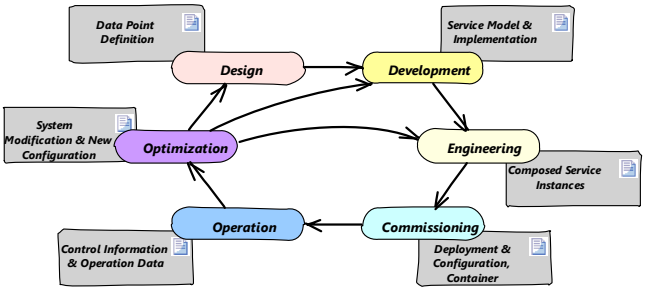

Figure 1: Lifecycle of a building automation system and related artifacts

We enter the lifecycle by looking at the design and development phase first (Figure 2). In this phase, the relevant user roles are the "domain expert" and the "software engineer". Key use cases are the maintenance of a set of data point types and the maintenance of a set of service types. These use cases include the (1) creation of new types, (2) the modification of existing types, and (3) the deletion of existing types. The existing data point and service type definitions are the required artifacts for the development phase. The maintenance of those types aims at fulfilling a predefined set of business case definitions, which have been previously defined and are input to this phase.

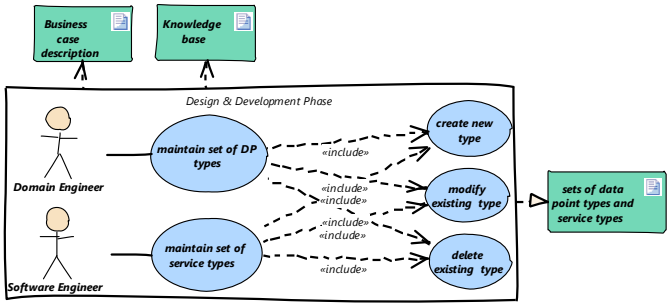

Figure 2: Use cases during development phase

Looking next at the engineering phase (Figure 3), the responsible user role is the system engineer. The key use case of this role is the creation of the functional model that describes the building automation system to be deployed in the next phase. This use case includes several related use cases, namely (1) the analysis of the existing building infrastructure, (2) the selection and instantiation of service types, (3) the parameterization of services, and (4) the wiring of service instances.

For all these use cases, the discovery of data point and service types is a prerequisite. To enable the user to define the functional model, she needs to be able to search over the types defined in the preceding development phase. For example, the user needs to be able to use all concepts of the underlying data point and service model as search criteria. In addition, existing service instances need to be discoverable to enable the wiring.

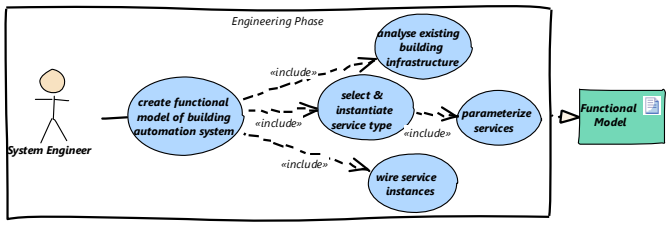

Figure 3: Use cases during engineering phase

The commissioning phase (Figure 4) deploys the previously defined functional model. The key role of this phase is the system installer, who has the primary use case of deploying and setting up the services to commission the building automation system. This use case entails (1) the analysis of the operational requirements of services, (2) the specification of the deployment and setup configuration, and (3) carrying out the deployment of all required service instances on the building infrastructure.

The input to these use cases is the functional model defined in the engineering phase. The system installer maps the defined functional model to devices and servers within the building. The existing infrastructure needs to be discoverable.

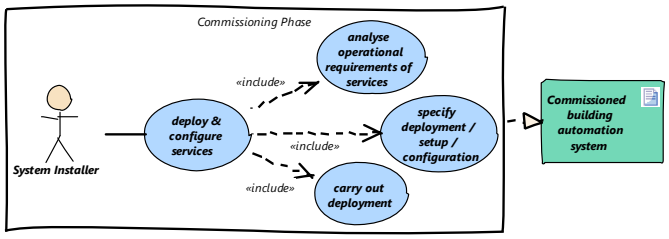

Figure 4: Use cases during commissioning phase

In the operation phase (Figure 5), the key user roles are the automated 'technical management system' and the operator. Their use cases are the maintenance of the deployed system, the monitoring of the system, and to add new services to the running system. In addition, external applications may access the building automation system. For example, this could be a smartphone app that allows interacting with the deployed building automation system.

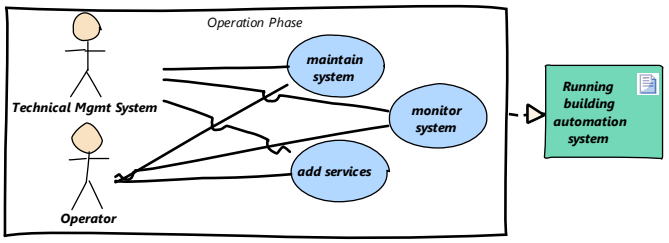

Figure 5: Use cases during operation phase

During the optimization phase (Figure 6), the key user roles are a system performing automated diagnosis and optimization and an engineering expert. They have the use cases (1) analysis of monitoring data and (2) adjustment of operational qualities.

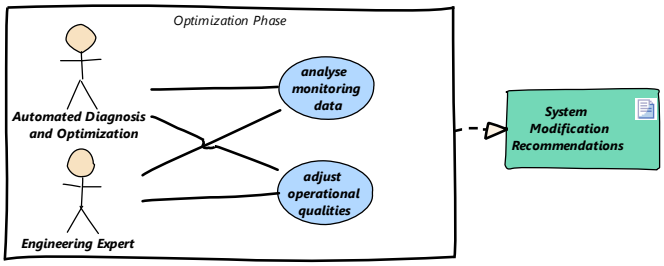

Figure 6: Use cases during optimization phase

C. *Information Model*

The BaaS Information Model provides a description of the data and metadata required for a BaaS Service throughout its lifecycle. The core concept of the BaaS Information Model is the BaaS Data Point. A BaaS Data Point serves as a blueprint for a BaaS Service in the early phases of its lifecycle, i.e. during development and engineering, and provides the information required for semantic discovery of BaaS Services at operation time. There is a one-to-one relationship between a BaaS Data Point and the BaaS Service that implements it.

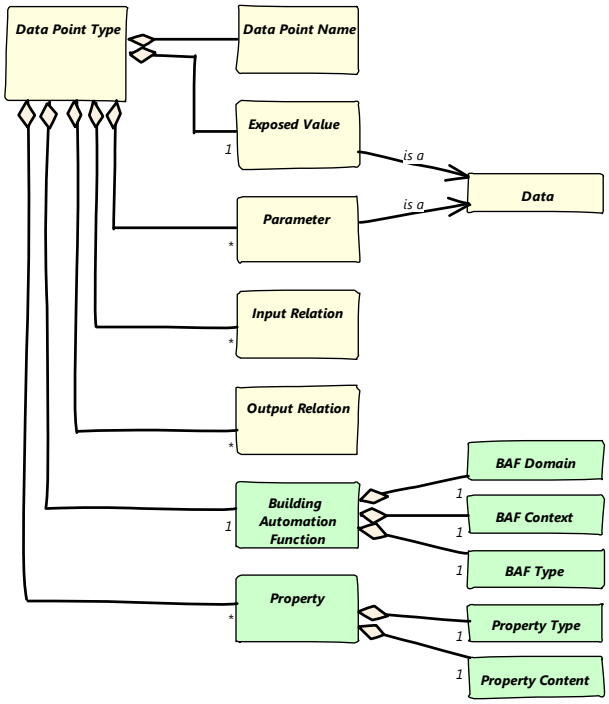

Figure 7: Conceptual BaaS Data Point model

A BaaS Data Point describes all data handled by its corresponding BaaS Service as well as the metadata characterizing the data handled by the service, the function of the service, the operation context of the service and the relationships to other services. In the following, the BaaS Data Point and related concepts are described in detail.

Figure 7 provides a model of a BaaS Data Point and its various properties. At the top of the diagram, the data represented by a BaaS Data Point are shown. The Exposed Value is usually a complex data structure that contains all data exposed to other BaaS Services and the BaaS Data Points implemented by those services, respectively. Elements of the Exposed Value are either read-only or writeable for other services. The detailed structure of such data is explained further below. Parameters have a name and can hold any kind of data. A data point may have an arbitrary number of Parameters that are defined at design time. Values are assigned to Parameters at engineering time; they are used by the algorithms implementing the functionality of the data point at operation time.

In the lower part of the diagram, the elements of the description of the Building Automation Function (BAF) that the data point represents are specified. The main elements of the BAF are:

- BAF Domain: the application domain that the function belongs to, e.g., Lighting, Heating, Air Conditioning.
- BAF Context: the subsystem the function is assigned to, e.g., in case of a heating: a boiler, a central heating return flow, a hot water storage tank.
- BAF Type: the type of the device or function represented by the data point, e.g., in case of a heating: a water temperature sensor, a central heating pump, a heating valve actuator.

In addition to the semantic description of the BAF, other metadata may be provided using so-called Properties. A Property has a name denoting the aspect it represents, and a content providing a concrete description of the aspect. For instance, a Property called "Location" may be introduced for a data point that represents either a physical device that is located at a particular location or a function that is associated with a particular range of a room or building. While arbitrary Properties may be defined for a data point at design time, it usually is useful to provide a standard set of Properties for all data points (at least as far as it makes sense).

In the middle of the diagram, two concepts that describe relations to other BaaS Services and their respective data points are mentioned: An Input Relation describes which values from which external data points are required as input values for the function implemented by the data point. An Output Relation describes which elements of the Exposed Value are provided as inputs to which external data points. A data point may have an arbitrary number of Input and Output Relations. They are defined at design time but instantiated with concrete references to data points only at engineering time when the relations between the Services of a BaaS system are established. In some cases, the references may be specified as matching rules that are applied at operation time to determine the actual data points to be referenced.

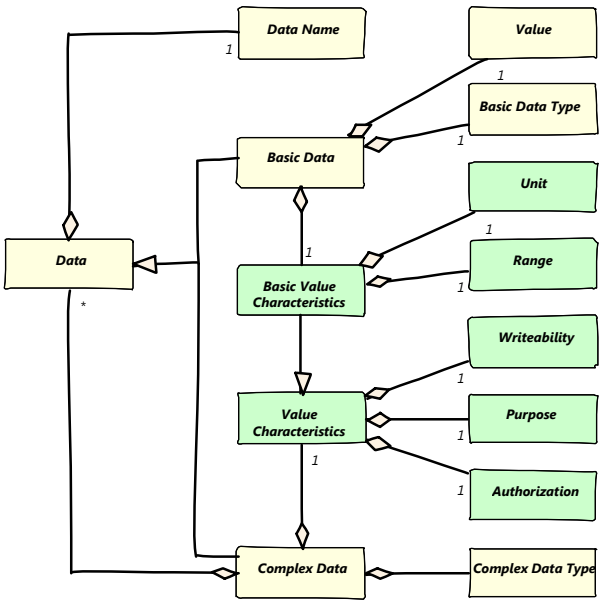

Figure 8: Conceptual model of Data with annotation

Figure 8 provides a model of the data used in the BaaS Data Point model for the Exposed Value and for Parameters. The diagram shows that data have a Name and a Data Type which in case of Basic Data is a Basic Data Type while in case of Complex Data it is a Complex Data Type. A Basic Data Type is equivalent to what is called a primitive data type in many programming languages (e.g., integer, float, string, or an enumeration of integer values) while a Complex Data Type describes a composition of Data (e.g., a sequence of different kinds of data, or an array of a particular kind of data). As the Data contained in a Complex Data Type may be Complex Data themselves, nested data structures are possible. Basic Data must also have an assigned (numeric) Value.

As opposed to ordinary type systems used in programming languages, BaaS Data also have metadata describing their semantics. Each data element at each level of the data hierarchy has so-called Value Characteristics. In general, they specify the meaning (purpose) of the element, whether it is writable, and which authorization is required for accessing the data element. Basic Data in addition have the following properties (as far as applicable): a description of the allowed range of the data (e.g., a minimum and a maximum value) as well as a description of the unit of measure, usually derived from the International System of Units (SI) [9].

All metadata used to describe the semantics of a BaaS Data Point and its data may be specified as references to concepts of ontologies. For instance, the Location Property may refer to an instance of a location concept described in a location ontology (e.g., a geographic coordinate). The attributes of the BAF may reference concepts provided by a building automation ontology. Units of measure are described in the QUDT ontology [10] provided by NASA. The way how such ontologies should be established is currently under investigation in the BaaS project.

D. Modeling Example for Legacy Integration

To illustrate the model described above we sketch the most relevant concepts with an extremely simplified example consisting of an outdoor temperature sensor represented on a BACnet controller and a heating control unit that makes use of this temperature sensor.

The temperature sensor is modeled to provide an exposed value of type temperature, i.e. a REAL value that has the unit °C and the purpose "temperature". The BAF describes the function of the data point as temperature sensor. Since the data point refers to a BACnet system, we add a legacy property that describes the origin of the data with its BACnet addressing and access method, i.e. *ReadProperty* to the *Present_Value* of a specific *Analog Input Object* located on a *Device Instance*. This property description is sufficient to generate code that accesses the BACnet device.

The heating control unit is specified to expose the result of the control algorithm and the related descriptions. Moreover, we specify an input relation to the exposed value of the temperature sensor data point. This input relation can be utilized to discover potential input temperature data points in the engineering phase and thereby reduce the complexity for the engineer.

III. Development and Engineering with the BaaS System

This section describes the tools and other main functional building blocks used in a BaaS system and how they are utilized during development, engineering and commissioning. First, we present the tools and functional building blocks used during each phase of the lifecycle of a BaaS system and show how they address the challenges mentioned in Section I. Secondly, we describe the BaaS Service Runtime used to run services of a BaaS system. These tools and functional building blocks presented here are currently developed as part of the BaaS project.

A. Tools and Functional Building Blocks of the BaaS system

A BaaS system is mainly organized as a system of inter-connected BaaS Services exchanging data. Each BaaS Service exposes Communication Resources representing the exposed value structure of a BaaS Data Point Type specification. A BaaS Service runs on top of a BaaS Service Runtime which takes care of operational properties like the communication protocol, the communication pattern, the activity model of the BaaS Service and other aspects.

The development phase includes two stakeholders, the domain engineer specifying a set of BaaS Data Point Types according to the information model specified in Section II.C. The software developer has to manually implement the Service Core behind a data point. The Service Core is supposed to provide the data according to the exposed value structure. This could imply the access to hardware such as sensors and actuators or the computation of derived data from such sensor measurements or the calculation of individual actor settings from a certain user input. The software developer also creates BaaS Service Types which make up templates to be used by the system engineer to instantiate and wire BaaS Service instances. A BaaS Service Type includes additional specifications like the communication protocol, the communication pattern or the activity model of the BaaS Service. This implies the generation of Adaptors (e.g., a Communication Adaptor) and glue code around the manually implemented BaaS Service Core.

The domain engineer mainly uses the BaaS Data Point editor as depicted by the following figure:

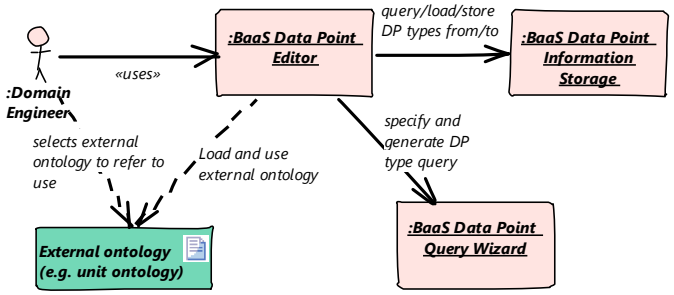

Figure 9: Specifying Data Points

The BaaS Data Point Editor allows creating/ modifying/deleting a BaaS-specific Data Point Type and specifying its properties according to the Information Model presented in Section II.C. For instance, the domain engineer uses this editor to create the heating control and temperature data points mentioned in Section II.D.

The Domain Engineer is also supported in browsing the existing set of Data Point types. The Data Point Editor uses the BaaS Data Point Query Wizard to support the Domain Engineer in expressing a data point query. The Data Point Editor uses the BaaS Data Point Information Storage to perform the query and present the query results to the Domain Engineer.

The Domain Engineer uses the Query Wizard for other purposes as well. For instance, she also uses the Query Wizard to express a data point query as the input or output relation of a new Data Point type to other data points (instances). This query is not performed on the BaaS Data Point Information Storage at development time; rather, it is applied at engineering time on the data points (instances) associated with BaaS Service instances. Thus, the query may already refer to context information of the data point like the information on the location (e.g., a query to retrieve all light switch data points on a certain office/floor of the building). When specifying the heating control data point from Section II.D, the domain engineer uses a parameterized query to specify the input relation to temperature data point with the location property as a query parameter. The parameter value is specified by the system engineer at engineering time in order to search for temperature data points providing temperature values.

The BaaS Data Point Editor also provides support for using concepts from externally defined ontologies in order to specify some properties of a Data Point Type like the building automation function or the value characteristics of parts of the exposed value structure. This is either achieved by importing a complete ontology into the editor, or by selecting and importing only required concepts from an external ontology.

In contrast, the software developer is mainly interacting with the BaaS Service Editor in addition to his normal development environment, e.g., an IDE. Both editors, the BaaS Data Point and the BaaS Service Editor, are based on several domain-specific languages and the underlying domain-specific language framework called Meta-Programming System (MPS)[1]. The interaction of the software developer with the BaaS Service Editor and the other functional building blocks is outlined in Figure 10.

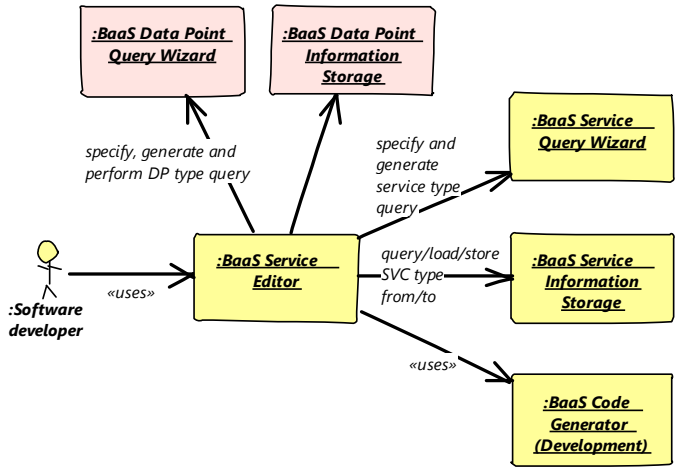

Figure 10: Specifying BaaS Service types

The software developer determines the Data Point Type(s) to be exposed by the new Service Type by using the BaaS Service Editor. In turn, the service editor uses the Data Point Query Wizard to compose a data point query to be performed by the BaaS Data Point Information Storage. The Service Editor supports the software developer in specifying several Service properties. As a third step the same or some other software developer triggers the code generation of the service editor to create the service core skeleton class and the service core handler interfaces. Section III.B explains these concepts in more detail. The software developer uses the generated artifacts to implement the skeleton and handlers, create the implementation artifacts manually by using his favorite IDE and stores these as part of the BaaS Service Type in the BaaS Service Information Storage.

The software developer uses the service editor and the BaaS Service Information Storage to create a library of re-usable service types to be used repeatedly by the system engineer. The following properties of a Service type are examples that can be chosen from a set of pre-defined options and be possibly extended/modified by the system engineer:

- the communication protocol of a BaaS Service Type (e.g., CoAP, DPWS) (preferably, the same protocol is used throughout an entire BaaS system),
- the communication pattern of a BaaS Service type with respect to its exposed data point (e.g., get/set, publish/subscribe, unsolicited set),
- the activity model of the BaaS Service Type, e.g.:
  - active (the value computation of a Data Point is triggered on a regular basis independently from any specific request),
  - passive (the value computation is triggered on demand by the request of another BaaS Service).

Other properties like the required or provided access permission of BaaS Services are mainly configured at engineering time when knowing the actual access model:

- the access control model for BaaS Service instances and their exposed data points (e.g., by specifying the required role(s) in order to get access to a data point of a BaaS service),
- usage of a particular authorization mechanism like OAuth to control the access of requests to a data point of a BaaS Service (after identifying the originator of a request by a suitable authentication mechanism).

Section III.B describes the generated Service Adaptor(s) and glue code to integrate adaptors with the service core of a BaaS Service Type.

During the engineering phase the system engineer creates a functional model of a BaaS system by creating/ modifying/deleting certain entities and their relationships according to certain composition rules and constraints by using the BaaS Engineering Tool. She

- creates BaaS Service instances (and indirectly their contained BaaS Data Point instances) by querying the BaaS Service Information Storage for service types,
- sets the configuration parameters for BaaS Service instances and their data points, and
- wires BaaS Service instances to each other according to the BaaS Data Point (input/output) relations.

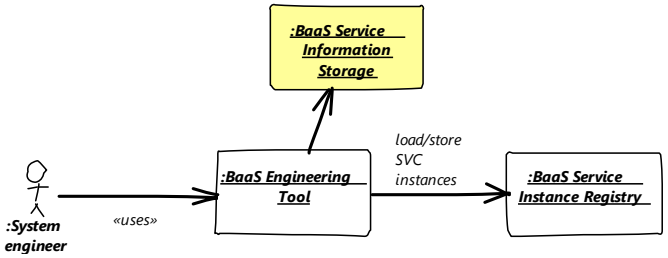

Figure 11: Create BaaS Service instances

The BaaS Engineering Tool allows generating deployable BaaS Service packages to deploy the BaaS Service instances during commissioning. Such a package contains

- the executable program of the BaaS Service instance, and
- the configuration of a BaaS Service instance, e.g., specifying the (relative) address (e.g., a URI) of the BaaS Service and its exposed resources.

B. Main Components of the BaaS Service Runtime

The BaaS Service Runtime hosts a running BaaS Service and the resources it exposes. For instance, the heating control data point from Section II.D is represented by an own BaaS

[1] https://www.jetbrains.com/mps/

Service (instance) exposing the different control settings as separate resources. The temperature data point is a resource which could be exposed by a second BaaS Service (instance). Figure 12 outlines the main components provided by the Service Runtime to each BaaS Service according to the specification of a service type and its data point. The BaaS Service Runtime handles operational properties like the communication protocol, the communication pattern, the activity model of the BaaS Service by generated components, their glue code and some manually implemented components. Thus, a BaaS Service is composed of several components. The Core component contains the manually implemented Core handlers needed to read, write and possibly compute and derive the values of the complex data structure of the exposed value of a data point.

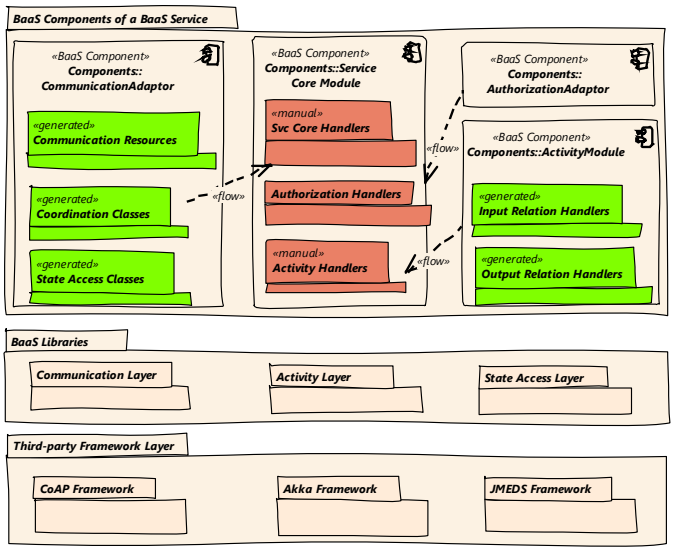

Figure 12: Components of a Service

There are several other components, e.g., the Communication Adaptor, the Activity Component, and Authorization Adaptor of a BaaS Service. The general goal is to generate all of them according to the data point type and service type specifications as described in Section III.A. Figure 12 marks all manually implemented parts of a service as red, whereas generated code is shown as green. The code generation leverages several BaaS libraries, which in turn are based on third-party framework. For instance, the BaaS Communication Layer library is an abstraction layer across different third-party CoAP frameworks like Californium [11] and JCoAP [12]. The remaining discussion focuses on the Communication Adaptor component and the Core component only, which are captured by the next figure.

The upper part of Figure 13 sketches the design of the Communication Adpator component. The lower part covers the Service Core component. The Communication Adaptor uses CommunicationClient and CommunicationResource classes of the BaaS Communication Layer. The BaaS Service Editor generates a configuration description of the hierarchy of CommunicationResources exposing the values of the complex data structure of the exposed value of the data point of a service. At startup time of the service the description is used by the Communication Adaptor component in order to instantiate the respective tree of CommunicationResource instances.

In addition, instances of the generated service Coordinator classes (SvcReadCoordinator and SvcUpdateCoordinater in Figure 13) are hooked into the Communication Resources to handle the incoming get/put/post/observe requests on a CommunicationResource. A Coordinator also uses a generated SvcDeMarshaler class to de-marshal the CoAP body payload into respective Java objects. It also coordinates the calls of the manually implemented Core handlers (SpecificReadHandler, SpecificUpdateHandler and SpecificDeriveHandler in Figure 13). All these handlers have to comply with specifically generated interfaces for a respective service (called SvcReadHandler, SvcUpdateHandler and SvcDeriveHandler).

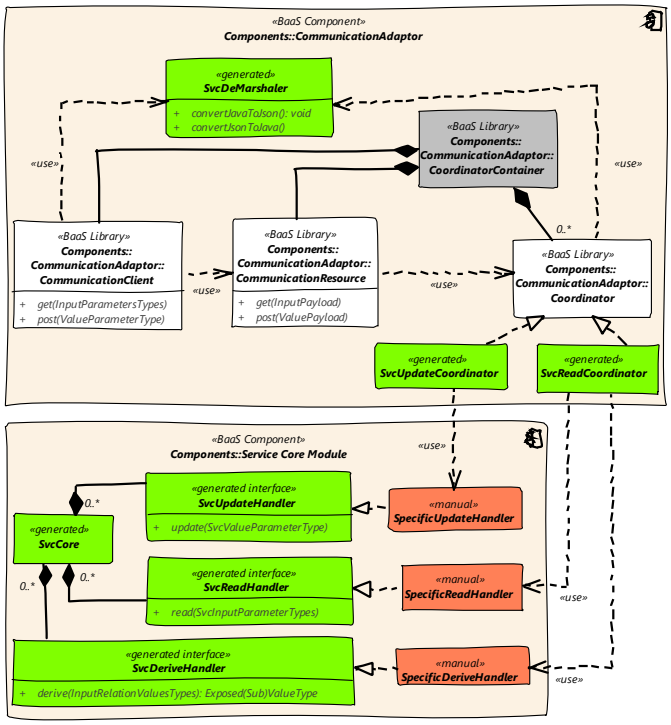

Figure 13: Design of the Core and CommunicationAdaptor components

C. *Discovery and Information Filtering*

For all the interactions between the tools and artifacts of the BaaS system as described above, an efficient *discovery* of resources is a key prerequisite. In fact, all use cases of the different lifecycle phases (Section II.B) of a building automation system are based on some sort of discovery. For example, in the engineering phase, where the service types are to be instantiated to define the functional model, the engineering tool must provide means to search for service types using various filter criteria. This *information discovery* is enabled through query requests against the information storage. On the other hand, during runtime of the building automation system, specific devices and service instances need to be discovered. For example, in the operation phase, a service representing a light switch needs to find the service representing the light to be switched on (or off). We call this form of discovery *service discovery*.

In the BaaS approach, we implement the information discovery through SPARQL [13] endpoints offered by the BaaS Data Point Information Storage and the Service Information Storage (Section III.A). Since the proposed information model (Section II.C) is represented in OWL/RDF [14], we implement the information storages as RDF triple stores. All created descriptions of data point and service types are inserted into the triple store by the according editor tools. Since this triple store exposes a SPARQL endpoint, we can flexibly query for the contained information.

The service discovery at runtime requires two key features: (1) we need to be able to resolve known IDs of running service instances in order to receive their addresses, and (2) we need to be able to search for running services by keywords (or "tags"). Both features are realized by leveraging the CoAP protocol. Therefore, we implement the BaaS Service Instance registry (Figure 11), where all service instances are registered, as a so-called *CoRE Resource Directory* [15]. The resource directory hosts links to all resources available in the building automation system. Links to registered resources are provided through the standardized *CoRE Link Format* [16]. For all resources, resource types are specified, similar to keywords or tags, but as URIs linking to concepts contained in the information storages, which are used for information discovery. Then, the resource directory allows searching over all registered resources using the associated keywords.

IV. Conclusions and Future Work

This paper presents a reference architecture for building automation systems that is based on semantic and service oriented approaches. We put special focus on the information model and show, how this information model is used during the lifecycle to make the development, engineering and operation of building automation systems more efficient.

The information model is centered on the concept of the BaaS Data Point that relates to the concept of data points handled by building automation engineers. We embody this concept in a twofold way: as a semantic model and as a software model used by model based code generation tools.

Further, we show how the tools of the BaaS Reference Architecture are used to define data points and iteratively generate executable services to perform building automation tasks. During the development and engineering a semantic search in the BaaS information storage, i.e. a filtering based on the semantic description of data points, can be applied to reduce the amount of information the engineer has to deal with. This enhances the engineering process independently of the used legacy technologies. During operation, the semantic description of data points can be used to resolve identifiers and thereby make the building automation system independent of specific IT infrastructures. The semantic descriptions can also be used to discover services at runtime and to achieve a late binding allowing for more flexibility or fault tolerant behavior.

In the future, we will look into ways to improve the usage of the semantic descriptions for discovery. In the first step, we aim at tag based search for services. We will investigate the usage of more advanced mechanisms for a semantic search, following approaches that have already been applied for linked data on the Web [17].

Acknowledgment

This paper builds strongly on the results of the ITEA3 BaaS Project – we would like to thank all the authors who contributed to the relevant sections of the deliverable on the BaaS Reference Architecture.

References

[1] 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.

[2] 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.

[3] 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.

[4] B. Butzin, F. Golatowski, C. Niedermeier, N. Vicari, and E. Wuchner, "A model based development approach for building automation systems," in *8th International Workshop on Service-Oriented Cyber-Physical Systems in Converging Networked Environments (SOCNE)*, Barcelona, Spain, 2014.

[5] C. Eastman, P. Teicholz, R. Sacks, and K. Liston, *BIM Handbook. A quido to Building Information Modeling. For owners, managers, designers, engineers, and contractors.* Wiley, 2011.

[6] A. McGibney, S. Rea, M. Lehmann, S. Thior, S. Lesecq, M. Hendriks, C. Gardeux, T. L. Mai, F. Pacull, J. Ploennigs, T. Basten, and D. Pesch, "A systematic engineering tool chain approach for self-organizing building automation systems," in *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, 2013, pp. 7696–7701.

[7] J. Ploennigs, B. Hensel, H. Dibowski, and K. Kabitzsch, "BASont - A modular, adaptive building automation system ontology," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 4827–4833.

[8] N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Pearson Education, 2011.

[9] "International System of Units." [Online]. Available: http://en.wikipedia.org/wiki/International_System_of_Units.

[10] Ralph Hodgson, Paul J. Keller, Jack Hodges, and Jack Spivak, "QUDT - Quantities, Units, Dimensions and Data Types Ontologies." [Online]. Available: http://www.qudt.org/.

[11] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the internet of things with coap," in *Proceedings of the 4th International Conference on the Internet of Things (IoT 2014)*, 2014.

[12] "jcoap - jCoAP is a Java Library implementing the Constrained Application Protocol (CoAP)." [Online]. Available: https://code.google.com/p/jcoap/. [Accessed: 29-May-2015].

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

[14] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "OWL 2 Web Ontology Language Primer," World Wide Web Consortium, W3C Recommendation, Oct. 2009.

[15] Z. Shelby and C. Bormann, "CoRE Resource Directory," 2014. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-resource-directory/. [Accessed: 20-May-2014].

[16] Z. Shelby, *RFC 6690 - Constrained RESTful Environments (CoRE) Link Format*. Internet Engineering Task Force (IETF), 2012.

[17] O. Hartig, C. Bizer, and J.-C. Freytag, "Executing SPARQL Queries over the Web of Linked Data," in *The Semantic Web - ISWC 2009*, 2009, pp. 293–309.