Building as a Service - BaaS

Deliverable

D05 – BaaS Reference Architecture

Editor:

Norbert Vicari, Siemens

May 20, 2016

ITEA 2 Project 12011

Document properties

Distribution	Public
Version	Version 2.0, May 20, 2016 Distribution changed to public as of November 2016
Editor	Norbert Vicari, Siemens
Authors/ Contributors	Gunes Karabulut Kurt, Defne Berna Ors Yalcin, Defne Francisco de Borja Ortiz De la Orden, Everis Christoph Fiehe, Materna Anna Litvina, Materna Darko Anicic, Siemens Michael Bahr, Siemens Sebastian Käbisch, Siemens Christoph Niedermeier, Siemens Jan Seeger, Siemens Egon Wuchner, Siemens Norbert Vicari, Siemens Marc-Oliver Pahl, TUM Benjamin Hof, TUM Malte Burkert, TUDO Andreas Müller, TWT Martin Neubauer, TWT Nacho Mansanet, UPV Joan Fons, UPV Björn Butzin, URO
Pages	144

Abstract

This document describes the second iteration of the BaaS reference architecture. Its main purpose is to provide guidance for the development of architectures. In particular, the reference architecture outlines the main functional groups and components and provides a blue-print for the specification of concrete service platforms and the related services.

The BaaS project [1] is targeted to establish a generic service platform for commercial buildings, that integrates traditional building automation and management systems with ICT infrastructures. The BaaS reference architecture follows a service-oriented approach and puts special focus on the definition, generation and deployment of novel building automation services and the integration of legacy services.

The BaaS reference architecture is described following a set of 'views and perspectives' approach. After a clarification of the terminology of the BaaS project in form of the BaaS domain model and the BaaS platform, we address the functional requirements of the BaaS architecture in four views:

- The Lifecycle View defines the activities in the different phases of the BaaS lifecycle, the related artifacts (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 and communication patterns of data exchanged between BaaS services.

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

The second iteration of the BaaS reference architecture uses feedback from the implementation of the BaaS platform for a major overhaul of the Information View, consolidates the information provided on the Technical Management as perspective in a single place and introduces several clarifications on the Lifecycle View and Security Perspective.

Table of Contents

1 Motivation and Structure

The main goal of the BaaS project [1] is to establish a generic service platform for commercial buildings that integrates traditional building automation and management systems with ICT infrastructures. This platform supports the development and deployment of novel valued added services and applications that take advantage of an integrated model of novel and legacy building systems and the data provided and consumed by those kinds of systems.

In detail, BaaS targets four technical objectives aiming at conceptualizing and developing

· a flexible open building service platform that facilitates the generation and deployment of value-added building services at a considerably lower cost compared to the state of the art;
· 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 the integration of existing and novel sources of information to create a "building information sphere" considering all stakeholders of the building.

Among the expected outcomes of the project are (1) a reference architecture for an implementation of the *BaaS* platform, (2) mechanisms to integrate with legacy building systems, (3) a set of basic platform mechanisms for the generation, deployment and composition of services, and (4) prototypical implementations of value-added services for validating and demonstrating the BaaS service platform.

This deliverable targets the first objective of the BaaS project in describing the BaaS reference architecture.

1.1 Why do we need a Reference Architecture?

The BaaS reference architecture is defined as an architectural design pattern that indicates how an abstract set of mechanisms and relationships addresses the set of BaaS requirements. The main purpose of a reference architecture is to provide guidance for the development of architectures.

According to the definition, the BaaS reference architecture establishes the basic principles of the BaaS concepts and describes the essential elements required for meeting the architecture requirements [2]. These elements are described independently of specific technologies. In particular, the reference architecture outlines the main functional groups and components and provides a blue-print for the specification of concrete service platforms and the related services.

1.2 Description of the Approach

The reference architecture is described based on the 'views and perspectives' approach as presented in [3]. This approach allows separating the different important concerns regarding the reference architecture, where views address the functional requirements of the reference architecture while the perspectives address the cross-cutting qualitative aspects of the system. In particular, we decided to describe the following views and perspectives:

- The Lifecycle View gives an overall overview of the lifecycle of a BaaS system. Since the BaaS approach aims at model based generation of services for the building automation domain, the development, engineering and commissioning phases are given special attention besides the operation phase. The Lifecycle View defines the activities in the phases, related artifacts (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 BaaS services. These services represent data points which are the entities building automation domain experts use to describe building automation systems. The information model also describes the methodology for the semantic description of the data points and BaaS services.
- 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 and communication patterns of data exchanged between BaaS services. This description is based on an analysis of the BaaS application cases by identifying communication patterns and requirements.
- The Dependability Perspective deals with the availability and reliability of the underlying system that enables the BaaS services to work correctly and with the needed performance.
- The Security Perspective describes the needed principles for confidentiality, integrity, authentication and authorization in order to protect the assets of a BaaS system.
- The Technical Management Perspective describes mechanisms that monitor and manage the deployed software components of a BaaS system in a way that the needed reliability is achieved.

Overall, the reference architecture provides concepts and mechanisms how the following aspects can be addressed:

- semantic models facilitating the description of functional and non-functional properties of the data to be exchanged between data providers and consumers,
- models and model-based methods that facilitate the automated generation, composition and deployment of value added building services,
- communication patterns covering the inherently massively distributed and usually heterogeneous nature of building systems,
- models and mechanisms for the integration of legacy building automation systems (e.g. BACnet),
- models on the management and reliability of the system.

1.3 Structure of the Deliverable

After the introduction, Chapter 2 lists the glossary of the BaaS project. The BaaS domain model and an overview of the BaaS platform and its technical management system is given in Chapter 3. The main part of the document is formed by Chapter 4 introducing the Views and Chapter 5 describing the Perspectives. The deliverable is concluded by a summary.

2 Glossary

The following table provides a glossary of specific terms used in the description of the BaaS reference architecture.

Term	Description
Application Case	Description of an application scenario enabled by the BaaS platform. Breaks down into a number of user stories. More details to be found in the BaaS Deliverable on use cases [4].
Architecture View	Principle: It is not possible to capture the functional features and quality properties of a complex system in a single comprehensible model that is understandable by and of value to all stakeholders. We need to represent complex systems in a way that is manageable and comprehensible by a range of business and technical stakeholders. A widely used approach - the only successful one we have found - is to face the problem from different directions simultaneously. In this approach, the architecture document is partitioned into a number of separate but interrelated views, each of which describes a separate aspect of the architecture. Collectively, the views describe the whole system. Definition: A view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders [3].
Automation Function	An open-loop or closed-loop control function or a locking function acting upon a process [5].
Artifact	Artifacts are result of a specific step in the life cycle, e.g. a document/specification, a concrete model at the end of modelling process, source code generated from the model, or object code as a result of a compilation process. Artifacts are used as basis for the development, engineering, commissioning, operation and optimization of the services in a BaaS system.
BaaS Container	BaaS containers provide an abstraction from the underlying hardware. BaaS services are deployed in BaaS containers.
BaaS Data	A set of data (values) together with a description of its type (e.g. array of integers)
BaaS Data Point	A BaaS data point provides the semantics to exchange VALUES between services.
BaaS Device	An entity that provides a hardware abstraction layer (HAL) to physical devices and connected sensors/actuators.
BaaS Gateway	A BaaS service that represents physical data points from legacy devices in the BaaS system.
BaaS Interface	The BaaS interface describes the access to BaaS data.
BaaS Ontology	The BaaS ontology is a common vocabulary (semantic model) which formally describes concepts in the BaaS system in machine interpretable and human understandable form.
BaaS Constraint	The BaaS Constraint is a mean to model certain qualities of non-functional

Term	Description
	requirements in the development phase. It consists of different qualities which can be assigned to the BaaS Constraint and the corresponding management tasks and measures to enforce the demanded quality at runtime. A BaaS Constraint can be applied to BaaS services and BaaS platform services during the engineering phase.
BaaS Platform	The BaaS platform consists of tools (DSL editors, model IDEs, runtime management tools, etc.), artifacts (basic models & ontologies), a repository to maintain and query these artifacts and a runtime to operate BaaS services. The platform is extendable in the sense that it allows the creation and operation of BaaS services. This might include the extension of the basic models and ontologies. The BaaS platform addresses requirements from the BaaS reference architecture.
BaaS Reference Architecture	The BaaS reference architecture is an architectural design pattern that indicates how an abstract set of mechanisms and relationships realizes the set of BaaS requirements. The main purpose of a reference architecture is to provide guidance for the development of architectures.
BaaS Registry	The BaaS registry is used to query/lookup BaaS services according their properties.
BaaS Semantics	BaaS semantics is a placeholder for the semantic descriptions needed in the BaaS system. These descriptions might be in different formats and tools, e.g. DSLs in a modeling tool/editor or ontologies in RDF format.
BaaS Service	A BaaS service executes building automation tasks. It is created and operated in the context of the BaaS platform.
BaaS Platform Service	A BaaS platform service is a service providing management capabilities, information and semantics about BaaS data points, BaaS services, BaaS container and BaaS devices. The BaaS platform services create an ecosystem without the operation of BaaS services would be much harder if not impossible.
Behavioral View	The process view describes the communication aspects of the BaaS system. These communication aspects have to be taken into account by the design and implementation of the BaaS platform. It refers to different communication approaches (e.g. event-based versus data-centric communication), their respective Quality-of-Service (QoS) aspects like reliability of delivery and relates these QoS aspects to the respective Application Cases from D02. Examples related to BaaS: communication of BaaS services over different communication protocols (e.g. CoAP), communication using different communication styles (e.g. request/reply, event-based communication, data-centric communication triggered periodically) [6].
Component View	Specifies the software components realizing the functional building blocks of the Functional View.
Data Point	A data point is an input/output function consisting of all assigned information describing fully the point's meaning (semantic). The data point's information includes the present value and/or state and parameters (properties and attributes), e.g. signal type, signal characteristics, measured range, unit, and state texts. There are physical and virtual data points. A physical data point is related to a direct or network connected field device within a homogeneous system. A virtual data point can be derived from the result of a processing function, or it is related to a device within a different system as a shared (networked) data point. A parameter having its own user address is also a virtual data point [7].

Term	Description
Functional View	Describes the system's functional elements, their responsibilities, interfaces, and primary interactions. A Functional view is the cornerstone of most ADs and is often the first part of the description that stakeholders try to read. It drives the shape of other system structures such as the information structure, concurrency structure, deployment structure, and so on. It also has a significant impact on the system's quality properties such as its ability to change, its ability to be secured, and its runtime performance. Examples related to BaaS: domain-specific BaaS editors, repositories of semantic descriptions of BaaS services, BaaS data points, BaaS SDK tools for code generation, BaaS runtime libraries covering communication, service management, security, BaaS BACnet gateway [3].
Information View	Describes the way that the system stores, manipulates, manages, and distributes information. The ultimate purpose of virtually any computer system is to manipulate information in some form, and this viewpoint develops a complete but high-level view of static data structure and information flow. The objective of this analysis is to answer the big questions around content, structure, ownership, latency, references, and data migration. Examples related to BaaS: content and structure examples of the semantic description of BaaS data points and services, data type examples of BaaS data points, etc. [3].
Lifecycle Phase	The service lifecycle of the BaaS system consists of the following phases: development, engineering, commissioning, operation, optimization, and decommissioning.
Lifecycle View	The Lifecyle View covers the user activities of each BaaS lifecycle phase of a BaaS system. It also describes the transitions between the phases with respect to the artifacts being exchanged from one phase to another. The Lifecylce View has to take into consideration that the activities of the BaaS lifecycle phases do not make up a linear sequence from one phase to the next. It also elaborates on the cycles of user activities as soon as a BaaS system has been initially engineered and commissioned and goes into operation. For example a BaaS system should enable stakeholders to operate additional, newly developed BaaS services during the whole lifetime of a building automation system.
Service Lifecycle	Describes the lifecycle of a BaaS service consisting of a number of lifecycle phases. More details to be found in common understanding of the lifecycle of a building automation system and related aspects as defined by the BaaS project.
Set Point	A set point is a special data point used to specify the desired process output that an automatic control system is supposed to reach. For example, a boiler might have a temperature set point, that is the temperature the control system aims to attain [8].
Use Case	Description of the generic interaction between external actors and the system to accomplish a certain generic goal. More details to be found in Terms and their Relations for structuring and defining BaaS use cases.
User Story	Detailed description of specific interaction steps (correct and complete) to achieve a concrete goal. More details to be found in Terms and their Relations for structuring and defining BaaS use cases.

Term	Description
Value Added Service	A service that uses several basic services like sensors and actors or other value added services to provide an added value to the installation. This especially includes the combination of solutions of the different building domains (HVAC, lighting, building management, safety, security).
Virtual Data Point	see data point.

Table 2-1: BaaS Glossary

3 BaaS Domain Description

3.1 Domain Model

The BaaS domain model (BDM) consists of a collection of concepts that have been used throughout the elaboration of the application cases as well as the derivation of the requirements. Thus, the BDM provides the basis for the elaboration of the BaaS reference architecture. It describes essential concepts from the building automation domain as well as from service-oriented architectures and some relations between those concepts. In the following, we will first explain the approach of domain modeling, and the relevance of the BDM for the development of the BaaS reference architecture. Then, we will give an overview of the concepts and relations described in the BaaS domain modeland have a closer look on some of the most relevant concepts.

3.1.1 What is a Domain Model?

A domain model [9] in software engineering is a conceptual model of all the topics related to a specific problem. It is used to represent and explain the vocabulary and key concepts of the problem domain. Furthermore, it also describes the roles of entities and the relationships among entities within the problem domain. In addition, important attributes of the described entities may be modeled. Usually, the domain model does not describe solutions to the problem. However, some aspects of possible solutions may be addressed. The domain model provides a structural view of the domain. It may be complemented by dynamic views. In particular, use case models are usually combined with a domain model.

A domain model is well suited to verify and to validate the understanding of the problem domain among different stakeholders. Therefore, it provides a good starting point for any kind of architectural analysis because it defines a common vocabulary and it may serve as a communication tool. In domain-driven design, the domain model covers all layers involved in modeling a business domain, including service layer, business layer, and data access layer. Thus, it can ensure effective communication at all levels of engineering.

The ultimate purpose of domain modeling is to capture the semantics of a domain rather than considering any technical details relevant for designing and implementing solutions. The vocabulary that is identified during domain modeling is commonly used as input for the definition of a glossary of terms which constitutes a useful tool throughout the architecture design and software development process. In particular, the glossary provides a foundation for the modeling of use cases.

3.1.2 Purpose of the BaaS Domain Model (BDM)

The BaaS domain model (BDM) has been initiated in the very beginning of the BaaS project, even before the discussion on application cases and user stories was started. It was used to identify some of the vocabulary that was needed for the specification of the application cases and user stories described in deliverable D01 [4].

First of all, the BDM addresses an essential concept from the building automation domain, the data point. According to ISO 16484-2 [10], a data point is “an input/output function consisting of all assigned information”. A more detailed definition of “data point” can be found in the version of the BaaS glossary (see Section 2). In the BDM, the concept of “data point" is used as anchor towards the building automation domain; other concepts in the model exhibit relationships with the data point and its specializations.

Among the most important concepts in the BDM are:

- Hardware entities (devices, servers) constituting the physical foundation of building automation systems;
- Middleware components (containers, registries) providing the basis for deployment and registration of building automation services;
- Building automation services (including services representing legacy devices) and associated concepts as data and interfaces;
- Data points (physical and virtual data points) capturing the functions of building automation systems;
- Semantic descriptions representing the meta-information characterizing the entities and data in building automation systems.

The detailed meaning and relationships of these concepts is presented further below.

The purpose of the BaaS domain model is to provide the basis for the elaboration of the BaaS reference architecture. In particular, the description of application cases and user stories as well as the description of architectural requirements has been elaborated with reference to terms and relationships from the BDM. Furthermore, the elaboration of the architectural views (see Section 4) has been performed with regard to the concepts described in the BDM.

The BaaS domain modelwas inspired by the domain modelof the IoT-A project that elaborated an architecture reference model for the Internet of Things [11].

3.1.3 General Overview on the BDM

The BaaS domain model is represented by several views (a top-level view and several sub-model views); not all of them being described in this document. In the following, a general overview of the model is given. Furthermore, two sub-models, namely the data sub-model and the ontology sub-model, are briefly described to give an insight into some more detailed concepts.

The top level view of the BaaS domain model is depicted in Figure 3-1. In the model, data points are used to describe input and output functions in the building automation domain. According to EN ISO 16484-2 [10] 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 represent the building automation entities in the BaaS domain model.

Data points may be physical data points or virtual data points. Physical data points are related to existing hardware that is usually associated with physical sensors or actuators. Virtual data points are derived or aggregated functions. In particular, virtual data points do not need any direct relationship to physical sensors or actuators. Rather, they are usually realized as a software function that may reside anywhere in the system, also on a server or in the cloud.

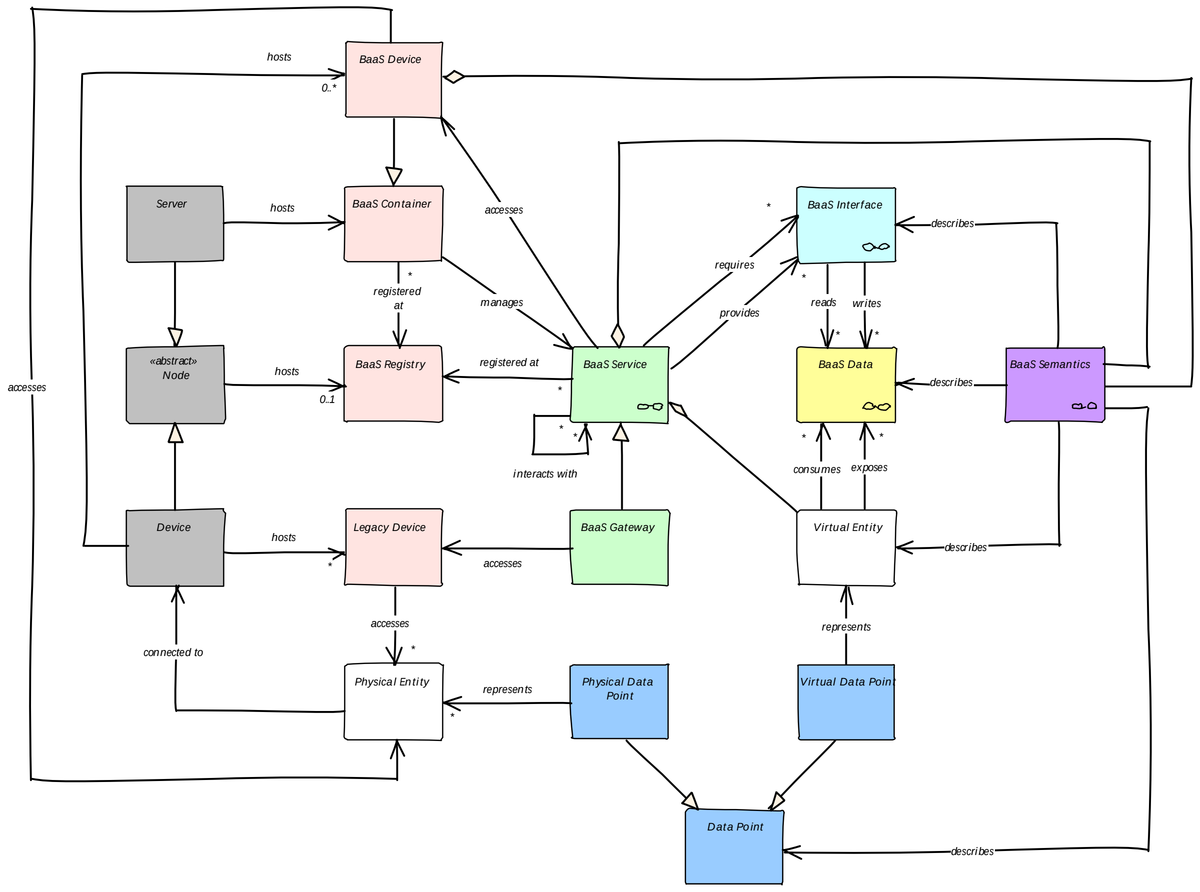


Figure 3-1: BaaS Domain Model - Top Level View

A core concept in BaaS is the BaaS service. A BaaS service 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 BaaS Interfaces, i.e. it provides BaaS Interfaces to other BaaS services and uses (requires) BaaS Interfaces of other BaaS services. A BaaS service is not necessarily a passive entity; it may also exhibit built-in active behavior (i.e. at times communicate with other services without requiring an external trigger).

A virtual entity represents a building automation function implemented according to the specification of the BaaS virtual data point associated with it. A virtual entity consumes and provides BaaS data that is read or written via BaaS Interfaces.

A BaaS service has a semantic description, which is represented by the concept of BaaS Semantics. BaaS Semantics are based on ontologies providing semantic concepts, properties and relationships for the characterization of data points and virtual entities. Furthermore, the ontologies also provide means to describe the semantics of BaaS data and the semantics of BaaS Interfaces of a BaaS service.

BaaS services may have relationships to other BaaS services, i.e. 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 them 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 has associated BaaS Semantics that are based on a device ontology.

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 both centralized and decentralized architectures can be specified.

Legacy devices are accessed by a BaaS gateway which is a specific type of BaaS service. From the view of another Baas service communicating with a BaaS gateway there is no difference to any other BaaS service. From the view of a developer a BaaS gateway provides a connector to legacy systems such as BACnet or KNX. Additionally the data obtained from the legacy device will be associated with semantic descriptions by the system engineer in order to be discoverable and interpretable by other BaaS services in the same way as all other data in a BaaS system.

3.1.4 Detailed Discussion of Selected Concepts

In the following, two examples of deep dives on particular concepts are presented. The description of these examples is non-exhaustive and is intended to provide guidance for the elaboration of the architectural views.

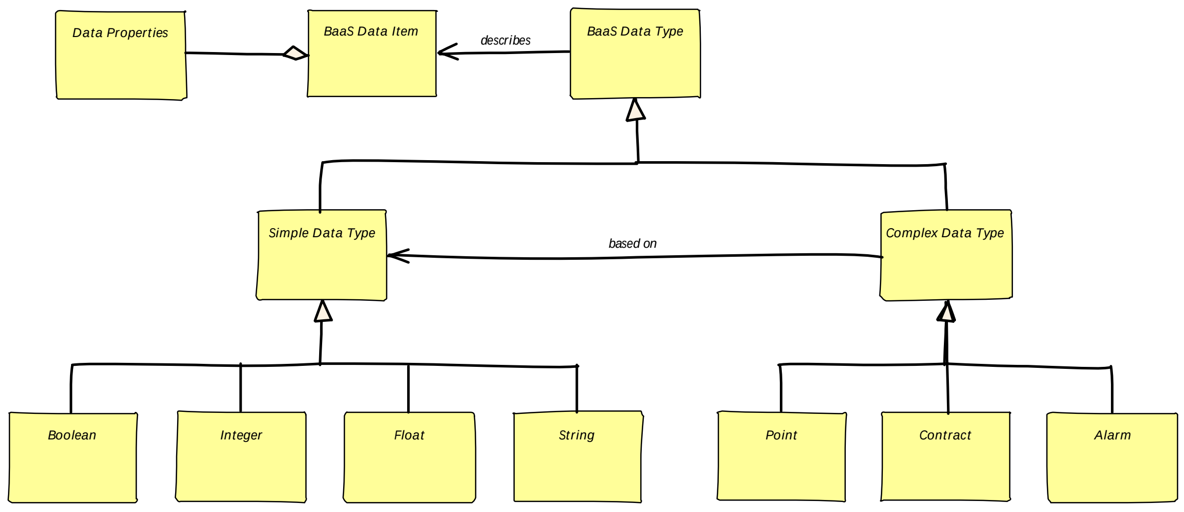


Figure 3-2: BaaS Domain Model - Data Sub-Model

In Figure 3-2, the data sub-model of the BDM is shown. It describes properties and concepts related with BaaS data. In particular, the concept of a BaaS data type is introduced, and specializations of this concept are provided. The concept simple data type represents a number of elementary data types (Boolean, Integer, Float, String) commonly used in type systems of programming languages. The concept composite data type is based on the simple data types and represents complex data types. As examples for those, the oBIX data types Point, Contract and Alarm are given. The data sub-model is just an example how a type system could be constructed; a more detailed elaboration of this is performed as part of the Information View.

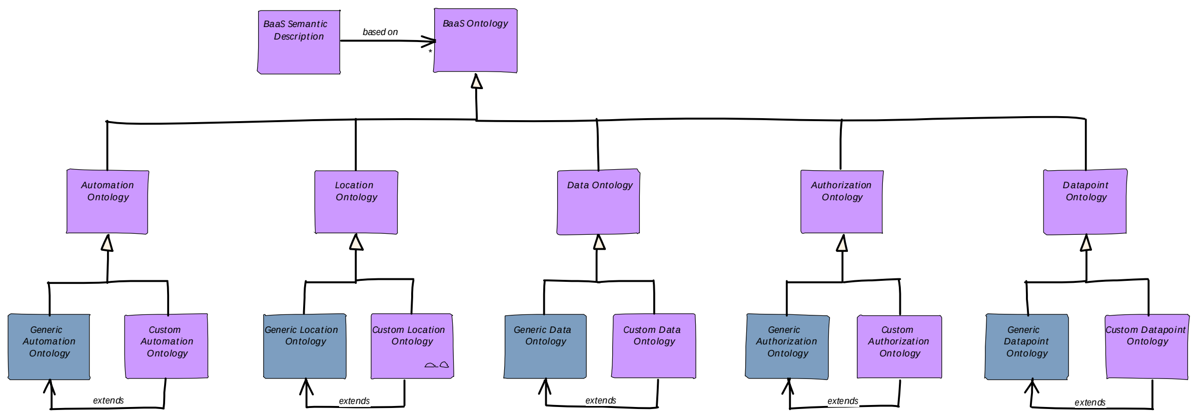

BaaS Semantic Description
based on
BaaS Ontology
Automation Ontology
Location Ontology
Data Ontology
Authorization Ontology
Datapoint Ontology
Generic Automation Ontology
Custom Automation Ontology
extends
Generic Location Ontology
Custom Location Ontology
extends
Generic Data Ontology
Custom Data Ontology
extends
Generic Authorization Ontology
Custom Authorization Ontology
extends
Generic Datapoint Ontology
Custom Datapoint Ontology
extends

Figure 3-3: BaaS Domain Model - Ontology Sub-Model

Figure 3-3 shows the ontology sub-model of the BDM. The diagram shows that BaaS Semantics are based on BaaS Ontologies. Several examples of BaaS Ontologies are provided, among them a

- Location ontology providing concepts for the semantic description of locations in buildings;
- Data ontology providing concepts for the semantic description of (building automation) data;
- Authorization ontology providing concepts of the semantic description of authorization related concepts as roles and permissions;
- Data point ontology providing concepts for the semantic description of physical and virtual data points.

Each of these ontologies has a generic part and a custom part. The term “generic ontology” denotes a standardized ontology that is the basis for extensions that are performed ad-hoc by vendors or installers of building automation systems without prior standardization. These extensions are denoted as “custom ontology”.

The ontology sub-model is just an example which ontologies may be needed for BaaS; the actual decision on specification and implementation in the project is subject to further study.

3.2 BaaS Platform

The goal of the reference architecture is to provide guidance for the proper planning and implementation of a specific BaaS platform. So the question arises what a BaaS platform is and what it does.

The platform itself consists of three parts that can be separated. The first is the BaaS framework consisting of several tools and methodologies that deals with the design time of the BaaS building automation system. The second part is the runtime environment that provides the capabilities needed to run BaaS services. The third part is the technical management system, which monitors the running services and assures the services are running properly.

3.2.1 BaaS Framework

During the design time phases: development, engineering and commissioning, the platform is responsible for the support of the stakeholders to achieve their tasks while designing a building automation installation. Every component, tool or methodology in these phases is part of the platform. Those help for example to create and modify data points which are already mentioned in the domain model section and to store them into a repository. The tools and methodologies also try to ensure that the generated interfaces of the created services always match the standards of the BaaS architecture and therefore ensuring compatibility between the services of different parties. Furthermore the security and privacy aspects of the target installation can be modeled and integrated. The tools are constructed not to collaborate together at the same time and thus they exchange their data in a document based manner.

3.2.2 BaaS Service Runtime

At runtime not everything that is present in the BaaS installation is also part of the platform. Only components that are required to run services like registries, container or devices belong to the platform. They are used to provide the basic mechanisms to start, stop and monitor services as well as to discover and bind them.

More detailed information about the platform its components, functions and behavior can be found in the architectural views section of this document.

3.2.3 Technical Management

The technical management maintains availability, stability and performance of the software components and IT infrastructure of a building automation system. It enforces the Quality of Service (QoS) and ensures that all the Service Level Agreements (SLAs) are met. This approach requires that applications, services, communication networks, field buses, devices and computing systems of the BAS are closely monitored in order to provide an efficient and dependable environment. The management process must be highly automated so that deviations in the predefined system behavior can be detected and compensated in a timely manner without any need for human intervention.

Further insight into the technical management, its challenges and its implementation is provided as perspective, c.f. Section 5.3.

4 Architectural Views

The reference architecture is described based on the 'views and perspectives' approach as presented in [3]. In the following chapter, the views addressing the functional requirements of the reference architecture are described. In particular, we decided to describe the following views:

- The Lifecycle View gives an overall overview of the lifecycle of a BaaS system.
- The Information View describes the information model used for the description of BaaS services.
- The Functional View defines functional building blocks of the reference architecture and it specifies its responsibilities and relations to other functional building blocks.
- The Behavioral View focuses on the interaction and communication patterns of data exchanged between BaaS services.

4.1 Lifecycle View

4.1.1 Introduction

The Chapter on the BaaS Lifecycle View describes the different lifecycle phases of the design time and runtime of a BaaS building automation system. It defines the activities in the different phases, related artifacts (models, software, ...) in each phase and the transitions from one phase to the other.

4.1.2 BAS System Lifecycle Revisited

In the following we recapitulate the lifecycle as given in D01 [4]. In difference to the former document we split the development phase into a design phase and the implementation related development phase.

Design

In the design phase data points are defined as the central structure in the BaaS system. Since data points are the entities used in the building automation domain language to describe building automation systems, the lifecycle starts with the definition of data point types, which include the semantic description of its own capabilities in a specified format. This semantic description includes the kind of data that will be exposed by the service, the control options it supports, the value characteristics (e.g. frequency and accuracy of the data), operational requirements (e.g. the minimum network bandwidth it needs in order to operate as expected). A more detailed modeling of data points and their features is described in the Information View, c.f. Section 4.2.1.

Development

During the development phase a new service type is created based on specified requirements and constraints. A service type is always based on a data point type. In case of dependencies, the new service type also specifies which other service types and semantic properties it requires in order to provide its own functionality. The data point types and service types (which include their semantic descriptions) are made available for the engineering phase.

BaaS constraints describe the demanded quality of services at runtime, e.g. reliability and availability. The BaaS constraints that will be applied in the engineering phase are (at least partially) specified in the development phase. This includes the corresponding qualities and the respective measures. As the result of the development phase, the system engineer creating/modifying the model of the BaaS building automation system has to be able to apply the demanded constraints and qualities to the respective BaaS services or to the whole model without any knowledge about the enforcement.

Engineering

In the engineering phase BaaS service types are instantiated and wired according to their specified semantic descriptions and in compliance to the input/output relations of the data point instances contained in the corresponding BaaS service instances. The engineering of BaaS services potentially makes use of the information about the building structure and relevant parts of the automation concept of the traditional/legacy building automation system. Engineering yields a partial functional model of the installation of the building automation system and the overall functional model of the additional BaaS services to be used in the commissioning phase.

In order to achieve BaaS services running within defined constraints the system engineer needs to specify them in the engineering phase. The constraints and corresponding qualities have been defined in the development phase. The system engineer creates the system model of the BaaS building automation system and applies the requested constraints accordingly. Consistency checks determine whether dependencies between services violate the modeled constraints.

Management rules are derived from the demanded constraints for the technical management system as input for the commissioning phase. The management rules are derived according to the measures for each constraint and quality defined in the development phase.

They are executed by the technical management system at operation time through a rule engine.

Commissioning

In the commissioning phase, the deployment of services to BaaS containers and their setup in the building is determined and performed based on a functional model. The commissioning phase might use the information from the development phase about the operational requirements and BaaS constraints of BaaS services. The relationships between the services identified in the engineering stage are mapped to configurations of traditional/legacy services, BaaS services, traditional/legacy automation devices and BaaS devices. Commissioning yields to a deployment model of the installation of the traditional/legacy building automation system and the additional BaaS services.

Different artifacts generated in the foregoing phases are integrated into the BaaS system in order to deploy, distribute, and configure BaaS services and BaaS platform services.

Operation

In the operation phase, the BaaS building automation system performs the configured functionalities. In addition, the BaaS services and BaaS containers are monitored to ensure their correct operation. In case of failures or anomalies, corrective and/or maintenance actions are performed either automatically by the technical management system or manually by the operator. Status information of BaaS services and BaaS containers and performance indicators are captured in the operational model.

Furthermore, new BaaS services and/or BaaS devices can be added in an ad-hoc manner during operation. This results in applying methods of other phases and updating of the models of the corresponding phases. The changes can be applied while the BaaS system is running and operating.

Analysis/Optimization

Buildings require adjustment of their operational qualities on a regular basis. This is achieved on the basis of performance indicators obtained by the analysis of monitoring data. In addition, changes in the BaaS building integration system or the building structure itself have to be taken into account. Adjustment of the qualities requires a modification of the engineering model resulting in changes of the deployment model. This is either achieved by the application of automated diagnosis and optimization procedures or manually by engineering experts. New BaaS services might be developed based on the results delivered by the operation of BaaS Monitoring and Analysis services. Other value-added BaaS services like the optimization of the energy consumption of the building according to the varying costs of energy provisioning (e.g. during different times of the day) could be devised and developed based on BaaS Monitoring and Analysis services.

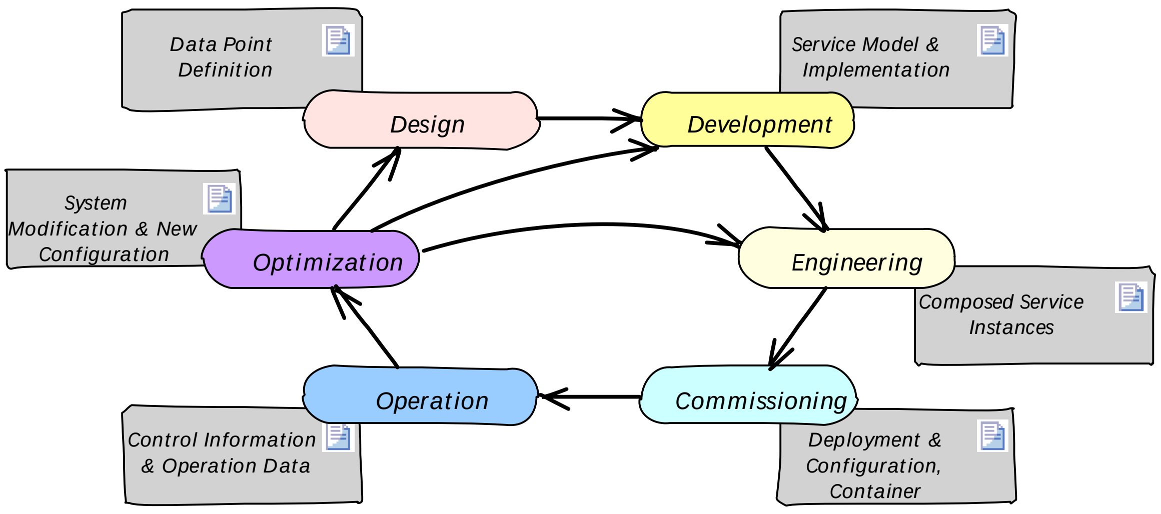

Data Point Definition
Service Model & Implementation
Design
Development
System Modification & New Configuration
Optimization
Engineering
Composed Service Instances
Operation
Commissioning
Control Information & Operation Data
Deployment & Configuration, Container

Figure 4-1: BaaS System Lifecycle

4.1.3 Activities throughout the BaaS Lifecycle

4.1.3.1 From BaaS Data Point Types to BaaS Service Instances

This section gives a brief overview of the metamorphosis of the basic artifact through the lifecycle phases as illustrated in Figure 4-1. The description in this section is useful background knowledge, which helps in more quickly understanding the structured textual description of the BaaS lifecycle phases starting in Section 4.1.3.4 with the design phase.

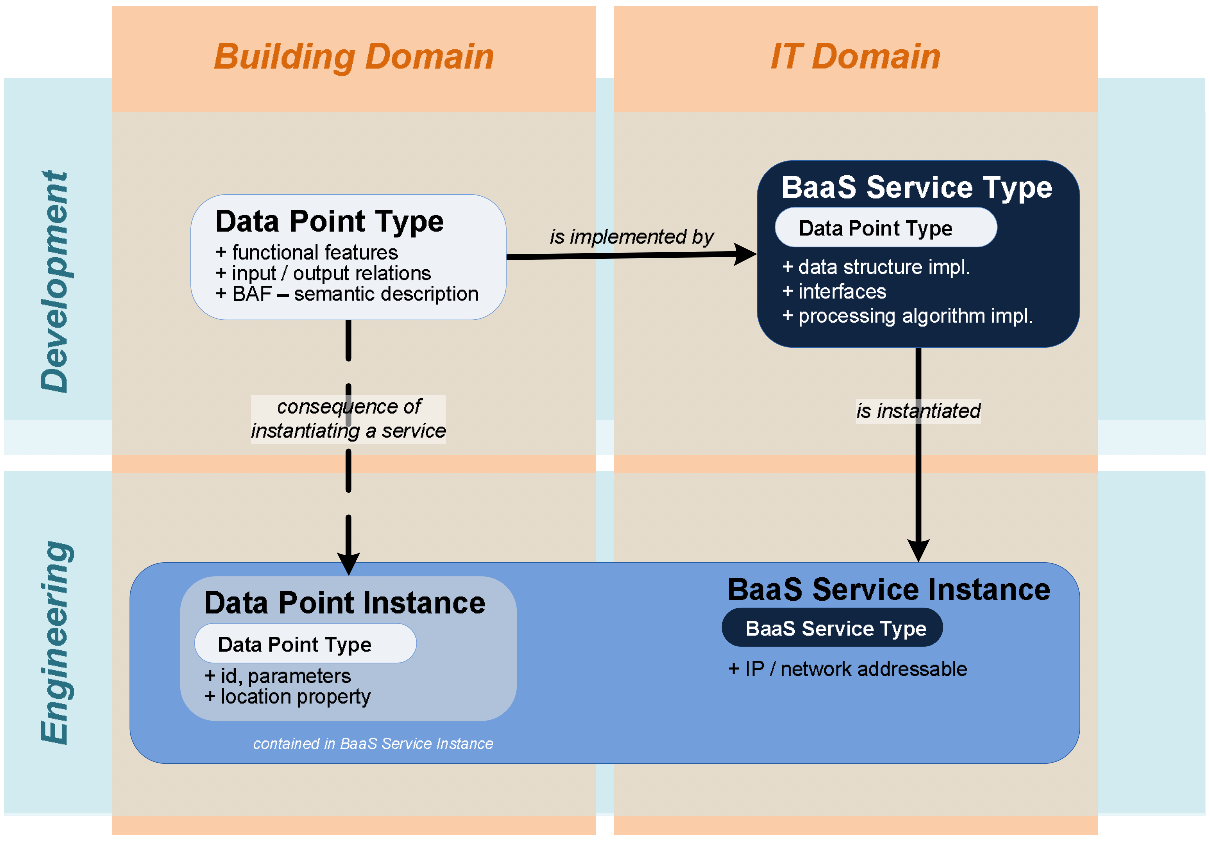


Figure 4-2: From BaaS Data Point Types to BaaS Service Instances

The building domain provides all information related to the description and modeling of the data points, i.e. the functional features, input/output relations and the semantic building automation function description (cf. Figure 4-2, left).

The IT domain provides all information necessary for a processing and handling of the data and information of the building domain by the information technology. The artifacts of the IT domain represent the corresponding artifacts of the building domain: The BaaS service implements the data point (cf. Figure 4-2, right).

The development is concerned with the specific concepts (or types) of the building automation and the IT domain. The domain engineer can concentrate on the concepts of the building domain in the BaaS development (see also Section 4.1.3.4). The software engineer implements and integrates the building domain-specific concepts (data point types) in the respective concepts of the IT domain (BaaS service types) (see also Section 4.1.3.5) (cf. Figure 4-2, top).

The engineering generates the instances from the concepts of the development and adds instance-specific information such as identifier, location, and network addressing. Note, only BaaS service instances are generated in the BaaS engineering. The information and data of data points are instantiated implicitly by the BaaS service instances. The information and data of the data point are contained and represented by the BaaS service instance (cf. Figure 4-2, bottom).

The data point type is the basic artifact at the beginning of the BaaS development. It is located in the building domain since it describes the building automation function (cf. Figure 4-2, top left).

The data point type is implemented in the IT domain through the BaaS service type. A BaaS service type represents the corresponding data point type. It contains the data point type besides additional information for a BaaS service type, and especially, the BaaS service type implements the represented data point type (cf. Figure 4-2, top right).

In the engineering, the instances are created from the developed types. It is important to understand, that only BaaS services are instantiated in the BaaS engineering. The BaaS service instance contains all necessary information from both the building domain (data point) and IT domain (cf. Figure 4-2, bottom).

The data point is only instantiated implicitly by the instantiation of the BaaS service type. The BaaS service instance contains all information of a data point instance (cf. Figure 4-2, bottom left).

4.1.3.2 Artifacts Provided by the BaaS Platform

The BaaS platform provides the runtime, tools, and models needed for the implementation of a BaaS building automation system (see Section 3.2 for further information on BaaS platform). Regarding the lifecycle view, the platform provides artifacts that are used as basis for the development, engineering, commissioning, operation and optimization of the services in a BaaS system. The following artifacts are provided:

- Basic data point types and selected BaaS ontologies that provide a basic vocabulary for the description of data points. These are partially based on external knowledge artifacts (standards and community accepted knowledge artifacts, e.g., parts of the SSN ontology or of Haystack, etc.). They might also represent basic functional patterns, such as sensor, actuator, or history.
- Information storages, registries and the related lookup (if needed reasoning) mechanisms.
- BaaS containers/BaaS devices
- Technical management system. The technical management system is described in detail in Section 3.2.3.
- Basic service skeletons, that define the structure of services and needed interfaces towards the containers and the technical management system
- Deployment Interfaces

4.1.3.3 Artifacts Provided by Customer Solution Specification

The term “Customer Solution Specification” has to be interpreted in a very broad meaning. It contains ideas, considerations, requirements, and results of all pre-design/development phases, i.e. requirements of the solution /installation before entering the lifecycle. It covers all input external to the BaaS system that guides the stakeholders of the BaaS LiveCycle phases in the selection of the corresponding elements.

The customer solution specification provides the following artifacts to the BaaS lifecycle phases

- List of needed data point types
- List of needed service types
- List of requirements on BaaS services

4.1.3.4 Design Phase – Domain Engineer

This section describes the tasks of the domain engineer in the development phase. See Section 4.1.2 for an overview of the design phase.

The main task of the domain engineer in the design phase is the definition of an initial description of the data point types. This description is based on semantic concepts and models provided by the BaaS platform. The description might be extended or refined during the BaaS lifecycle. In the design phase, the focus is on a functional description of data point types from the perspective of the building automation domain. This description will use the specific information model of the building automation context, which is contained in the information view/information model and is described in detail in Section 4.2.

The goal is to provide all the necessary information for the software engineer in the development phase. The software engineer must be able to provide all libraries and models so that the executable code for the BaaS services can be generated after the instantiation and parameterization of the services (representing their contained data points) by the system engineer in the engineering phase.

Required Artifacts

The following artifacts are required:

from the BaaS platform (The BaaS platform is described in detail in Section 3.2, the artifacts provided by the BaaS platform are listed in 4.1.3.2.):

- Semantic concepts and models, including
  - Generic data point skeleton, i.e. the definition what belongs to a data point
  - Building automation ontology, that describes the common building automation functions

from the customer solution specification:

- List of needed data point types

from the optimization phase:

- List of data point types to be changed or added (only used in development phase for optimization by domain engineer; see also Section 4.1.3.9 on optimization phase)

Tasks

The domain engineer performs the following tasks in the design phase:

Specification of data point types and their properties: The data point types and their properties are specified in a description which covers all elements as given in the information view / information model for the data point type in Section 4.2. The description of the data point types and their properties contains the following specifications and assignments:

- building automation functionality of the data point type in a semantic way
- values provided by the data point types
- value characteristics, that is, the data type, the unit, etc. of the values of the data point type. The data type defines the structure of the value (data type description) and is globally defined for reuse between different data point types.

- input relation to needed input values from other data point types: This might be a simple reference (which is also a semantic query) to a data point type or might be a complex semantic query that contains several criteria in order to determine during runtime the set of data points instances being part of the input relation.
- output relation to other data point types: The other data point types use the value of this data point type as input value. The output relation might be a simple reference (which is also a semantic query) to a data point type or might be a complex semantic query that contains several criteria in order to determine during runtime the set of data point instances being part of the output relation.
- "processing mechanism" describing how the output value is derived from the input values in the building automation functionality: A description of the processing mechanism is needed for the software engineer to implement the service representing the data point type and can be provided in plain text (no ontology or programming code needed).
- configuration properties of instances of the data point type (e.g. time to keep the stored data in a history database)
- gateway properties: They are needed if instances derived from a data point type are represented by a BaaS service with gateway functionality. This is the case if the original source is in a (domain-) specific protocol and behind a gateway (protocol translator).

  Remark: Probably, there will be the need to develop some specific software for this kind of BaaS gateway. If information on the original source is not available to the software engineer during the development phase and it is not possible to automate / generate the needed implementation in the engineering phase, the lifecycle might go back from the engineering phase to the development phase for this BaaS gateway.

- assign qualitative requirements for dependability (e.g. measure of a system's availability, reliability, maintainability, safety)
- assign generic security needs, e.g. encryption, authorization, or confidentiality levels

  Remark: The actual use and specific rights/roles might be decided at the engineering phase.

  Remark: The requirements and security needs might create constraints for the general hardware selection. The hardware selection and installation is outside the BaaS scope.

Check consistency of the data point type descriptions: The resulting set of data point type descriptions has to be consistent. For instance, data point types of an input relation have to be compatible. The consistency check is presumably a feature of the editor and might be done automatically and tool-supported.

Artifacts

The following artifacts are the result of the work of the domain engineer in the design phase:

- Set of data point type descriptions that describe the different available data point types in the system.

  The structure and the content of a data point type description is given in the Information View in Section 4.2.

Remark: At this point, the description does not refer to services and specifies only logical input/output relations to other data point types or sets of data point instances during runtime from which to retrieve or to which to provide values. However, real connectivity will be established by means of services.

- Set of data type descriptions that are reusable for the definition of data point value types and interfaces.

4.1.3.5 Development Phase – Software Engineer

This section describes the tasks of the software engineer in the development phase of the BaaS lifecycle. See Section 4.1.2 for an overview of the development phase.

The software engineer in the development phase provides all libraries and models so that the executable code for the BaaS services can be generated after the instantiation and parameterization of the services representing the data points by the system engineer in the engineering phase. Furthermore, the software engineer in the development phase specifies which qualities in terms of constraints could be applied to service type instances in the engineering phase. This so called BaaS constraint includes the name of the constraint (e.g. availability), the qualities the constraint could have (e.g. 99.99%, 99.999%) and the measures enforcing the respective qualities in the form of Event-Condition-Action rules (e.g. on service lifecycle change – if service has fallen out - deploy service to another computing node, start the service, repair the former service instance, try to set the former service instance to cold standby).

Required Artifacts

The following artifacts are required

from the domain engineer in the design phase (see 4.1.3.4):

- Data point type descriptions from the information storage
- Data types

from the customer solution specification:

- List of needed services types for access of data point types
- List of requirements on BaaS services

from the optimization phase:

- List of BaaS service types to be changed or added (only used in development phase for optimization by software engineer; see also Section 4.1.3.9 on optimization phase)

Tasks

The software engineer performs the following tasks in the development phase:

Creation of BaaS service types in a model driven way: The creation of the Baas service types is done in a model-driven way by the software engineer. It is based on the available data point types and data types in the corresponding information storages and on the needs of the customer solution specification, that is, the list of needed service types (see paragraph *Required Artifacts*).

Note, the BaaS service types are specific to the data point types they represent.

The following has to be defined, specified, or implemented during the creation of a BaaS service type:

- define data structures for the values of the represented data points based on the given data types in the information storage
- implement processing code. This processing code implements the building automation function in executable software.
- define interfaces
  - specify access to the provided values of data points represented by the BaaS services
  - specify reception of the required values of (other) data points. ('Understand' the interface of the BaaS service types providing input; this may depend on the used communication pattern.)
  - specify exchange of authorization information
- Implement "glue code" for security, communication, etc. The decisions on specific mechanisms for security, communication, and so on are done at the engineering phase.

  Remark: the communication patterns are decided during the engineering phase. The implementation of the service model must allow generating the software artifacts expressing all patterns as needed.

- Specify resource requirements of service types. An update might be required if code generation or decisions in the engineering phase significantly change the requirements.
- Definition of BaaS constraints consisting of name, acceptable qualities and automatic measures being executed at runtime through the technical management system on occurrence of certain system changes (e.g. network down, device offline, service failure, etc.). The measures can be defined through patterns given by the platform. The measures have the form of the aforementioned Event-Condition-Action-Rules

Artifacts

The following artifacts are the result of the work of the software engineer in the development phase:

- Set of service types available in an information storage.
  The service types are model-based and should allow to create executable code on the target device and follows a well-defined formal structure.
- Libraries needed by model in order to generate compiled code
- Additional resource files needed for BaaS service (html files, java script files, etc.)
- BaaS Constraints describing the acceptable qualities of constraints and respective behaviors enforced by the technical management system

All artifacts, including their semantic descriptions, should be available in an information storage.

Remark: The artifacts in the information storages are a data point centric representation/description of concepts that are partially contained in the BaaS ontology. This means that the BaaS ontology is represented in the information storage. A tool is needed to show this information in human-readable form.

4.1.3.6 Engineering Phase – System Engineer

This section describes the tasks of the different stakeholders in the engineering phase of the BaaS lifecycle. The system engineer is involved in the engineering phase. See Section 4.1.2 for an overview of the engineering phase.

The main task of the system engineer in the engineering phase is the instantiation of the BaaS services (representing their contained data points). This includes the parameterization of the BaaS services and their contained data points. The system engineer completes the model of the BaaS building automation system, so that it is deployable after the final code generation. All the still missing, necessary extensions and refinements to the description of the BaaS services and their contained data points will be done. In the engineering phase, the focus is on a the design of the actual BaaS building automation system from the set of provided BaaS service types in order to provide the final model of the actual BaaS building automation system.

The goals are to instantiate and parameterize the BaaS services (representing their contained data points) and to provide all the executable code, in terms of a model of the actual BaaS building automation system, for the BaaS services for the commissioning phase.

Required Artifacts

The following artifacts are required

from the development phase (see 4.1.3.5):

· Service types in a model-driven way and available in an information storage (the model should allow to create executable code on the target device and follows a well-defined formal structure)
· Libraries needed by the modeled service types in order to generate compiled code
· Additional resource files needed for BaaS services (html files, java script files, images, etc.)
· BaaS Constraints defined by the software engineer

The service types have available all information of their represented data point types.

from the optimization phase:

· Parts of the model of the BaaS building automation system to be changed or added, such as lists of installation-wide parameters and BaaS service instances and technical management system configuration (only used in engineering phase for optimization by system engineer; see also Section 4.1.3.9 on optimization phase).

Tasks

The system engineer performs the following tasks in the engineering phase

for the BaaS services:

· *Definition of installation-wide parameters* such as encryption method, authorization method, communication protocol, and access roles.
· *Creation of BaaS service instances* by model-driven methodology and by using the service types available in the information storage.

- *Specification of bindings between BaaS service instances*. The meta-information of the BaaS data point types is accessible via the BaaS service instances and guides the wiring of BaaS service instances.
- *Assign communication patterns and activity model* to the BaaS service instances. Active BaaS services have an internal activity loop or timer and can trigger other BaaS services.
- *Specification of values of configuration parameters*
- *Parameterization of BaaS service instances*: The BaaS service instances are parameterized in a way that takes the dependability constrains into account.
- *Configuration of BaaS gateway service instances*: BaaS gateway service instances have to be configured with the address of the corresponding legacy automation object.
- *Assigning BaaS Constraints* with particular parameters to BaaS services instances, groups of BaaS service instances or the whole BaaS service landscape
- *Consistency check of constrained BaaS service instances* that have dependencies with other BaaS service instances.

for the technical management system:

- *Generation of management artifacts* for configuration, monitoring, service distribution, tests, and adaptation to system changes. This includes
  - *Specification of QoS requirements* of BaaS service instances
  - *Specification of management agents and technical management system managers*
  - *Specification of the technical management system manager hierarchy*, if required
- *Derivation of management rules* in accordance with the constraints and requirements of the BaaS building automation system

Artifacts

The following artifacts are the result of the work of the system engineer in the engineering phase:

Model of BaaS building automation system, deployable after final code generation. The model of the BaaS building automation system contains
- Installation-wide parameters such as encryption method, authorization method, communication protocol, and access roles.
- BaaS service instances. The BaaS service instances have been configured with
  - bindings between BaaS service instances
  - all needed configuration values as provided during the tasks by the system engineer
  - communication pattern and activity model
  - BaaS constraints applied to BaaS service instances
  - resource requirements of BaaS service instances
  - addresses of the legacy automation objects, if the BaaS service instance is a BaaS gateway service instance.

Configuration for the technical management system containing
- set of management agents
- set of technical management system managers
- Management rules for the technical management system

Remark: During the operation and optimization phases, new BaaS services might be defined (delta engineering). This will lead to new BaaS service instances added to the overall model.

Only these changes to the artifacts will be deployed in the commissioning phase succeeding the definition of new BaaS services during the operation and optimization phases. The BaaS system might be running and operating during the application of these (delta) changes.

4.1.3.7 Commissioning Phase – System Installer

This section describes the tasks of the different stakeholders in the commissioning phase of the BaaS lifecycle. The system installer is involved in the commissioning phase. See Section 4.1.2 for an overview of the commissioning phase.

The system installer in the commissioning phase deploys the executable and configured software from the model of the BaaS building automation system to the actual hardware of the BaaS building automation system.

Required Artifacts

The following artifacts are required

from the BaaS platform (The BaaS platform is described in detail in 3.2, the artifacts provided by the BaaS platform are listed in 4.1.3.2.):

- BaaS containers/BaaS devices

from the engineering phase (see 4.1.3.6):

- Model of BaaS building automation system, containing amongst others
  - BaaS service instances and their bindings
  - Configuration values for BaaS service instances
- Configuration for the technical management system, containing
  - Management agents and technical management system managers
  - Management rules

from the development phase (see 4.1.3.5):

- Libraries and additional resource files organized as service bundles

from the BaaS platform (The BaaS platform is described in detail in 3.2, the artifacts provided by the BaaS platform are listed in 4.1.3.2.):

- Deployment interfaces

Tasks

The system installer performs the following tasks in the commissioning phase:

- *Generation of final, executable code* from model of BaaS building automation system. This includes the deployment of the BaaS containers and the final, executable code of the BaaS service instances (the BaaS service software packages).
- *Assignment of BaaS service instances to BaaS containers* (possibly there are suggestions from the engineering phase)
- *Deployment of BaaS service software packages on BaaS containers* using the deployment interfaces provided by the BaaS platform
- *Connect BaaS gateway services* to the corresponding legacy automation objects
- *Initial software/BaaS service test* in order to check whether the BaaS service is running correctly

- *Configuration of technical management system*: The technical management system is described in more detail in Section 3.2.3. The configuration of the technical management system includes the following tasks:
  - *Deploying the technical management system managers*: The specified technical management system managers are instantiated and deployed in the technical management system.
  - *Deploying the management agents*: The corresponding management agents are instantiated and deployed in the monitored BaaS building automation system.
  - *Application of corresponding technical management system configurations* to technical management system managers and management agents
  - *Deploying the management rules* to the technical management system managers

Artifacts

The following artifacts are the result of the work of the system installer in the Commissioning phase:

Commissioned BaaS building automation system containing

- Configured BaaS containers
- Executable, configured software of BaaS service instances deployed on BaaS containers
- BaaS gateway services connected to legacy automation objects
- Configured technical management system with management rules

4.1.3.8 Operation Phase – Facility Manager/Service Technician

This section describes the tasks of the different stakeholders in the operation phase of the BaaS lifecycle. The facility manager/service technician is involved in the operation phase. See Section 4.1.2 for an overview of the operation phase.

The facility manager/service technician is operating, monitoring, and maintaining the running, i.e. operating, deployed BaaS building automation system in the operation phase.

Required Artifacts

The following artifacts are required

from the commissioning phase (see 4.1.3.7):

- Commissioned BaaS building automation system

Tasks

The facility manager/service technician operating the BaaS building automation system performs the following tasks in the operation phase

for the BaaS building automation system:

- *Operating* the BaaS building automation system
- *Application of adjustments to configuration parameters* of the BaaS building automation system. In order to determine the necessary adjustments, the facility manager/service technician interacts with the BaaS services and the BaaS devices.
- *Accounting of used BaaS services* towards the inhabitants of the building. Only a few BaaS services are applicable for a monetary accounting, e.g. heating. This monetary accounting might be outsourced to third party service companies. Monetary accounting of used BaaS services is not part of this stage of the BaaS project.

- *Monitoring of devices*: The status of devices (e.g. sensors and actuators) and other information relevant for detecting aging, malfunctioning, or failing devices needs to be monitored (condition monitoring) so that the facility manager/service technician can act accordingly (predictive/ proactive and reactive maintenance).
- *Repairing of devices*: Malfunctioning and failing devices need to be repaired or substituted.
- *Tracking of maintenance actions*: The maintenance tasks follow specified maintenance processes that need to be obeyed. Maintenances actions and issues need to be tracked according to the defined maintenance processes, for instance, in log files, maintenance forms, and tracking tools.

for the technical management system

- *Monitors BaaS nodes, Baas containers and BaaS services* in order to keep track of critical parameters
- *Collection of relevant information* from the nodes, BaaS containers and BaaS services that are relevant for the technical management system
- Checks compliance against constraints and thresholds
- Reconfigures and adapts system behavior in case of violations via management rules

for the inhabitants of the building

- *Obtain information* about the environment, e.g. status of sensors and relevant information for them
- *Pay your bills*: The inhabitants of the building have to check and pay their bills for certain building management services such as heating.

Remark: The operation phase contains some tasks, such as adjustments to configuration parameters, which can be seen as tasks of the optimization phase conceptually. In a practical deployment of a building automation system, such optimizations or adjustments are rather obvious and are rather simple in their complexity. They may be done immediately by the facility manager/service technician who is the stakeholder of the operation phase. In fact, many people naturally see the adjustment of configuration parameters of building automation functions as part of the operation of a building automation system.

Artifacts

The following artifacts are the result of the work of the facility manager/service technician in the operation phase:

Running BaaS building automation system (leading to happy inhabitants **J** ¡)
from the BaaS building automation system:

- Logs of configuration changes and errors
- Protocols of operation, errors, and notifications of BaaS services
- Measurements of sensors
- Status data (reports) of operation of the building (e.g. temperature, energy usage)
- Usage numbers and statistics for monetary accounting information for billing (e.g. heating costs)

for the technical management system

- Status data (reports) reflecting performance and QoS indicators
- Logs of Notifications representing relevant system state changes

- Adjusted configurations of BaaS nodes, Baas containers and BaaS services

4.1.3.9 Optimization Phase

The optimization phase of the BaaS Lifecycle View is a special phase, since it is a post-operation lifecycle phase with feedback loops to pre-operation lifecycle phases. The optimization phase requires an operating BaaS building automation system and has a feedback loop to the design time lifecycle phases (development phase and engineering phase) as well as to the operation phase.

The goal of the optimization phase is to optimize, but also to extend the deployed and operating BaaS building automation system based on the collected logs, protocols, and reports of measurements and status data. The collected logs, protocols, and reports of measurements and status data come from the BaaS building automation system and corresponding technical management system and are artifacts of the operation phase (see 4.1.3.8).

Note: The extension of a BaaS building automation system can be seen as an optimization with respect to the building. In any case, both, optimization and extension of the BaaS building automation system require changes to the model of the BaaS building automation system and follow the same feedback processes of the lifecycle view and use the same methods.

The optimization phase requires

- a deployed and running BaaS building automation system from the operation phase (with the facility manager or some inhabitants being at least a little bit unhappy K! so that some improvement is needed).
- logs, protocols, and reports of measurements and status data from the deployed and running BaaS building automation system and its corresponding technical management system, so that the optimization can be determined and specified.

An optimization phase is needed, if

- some BaaS services or devices have insufficient performance and a performance improvement is needed in order to fulfill the requirements.
- energy efficiency is to be improved.
- devices are replaced by newer generations with more and improved functionality.
- the building is going to be refurbished or renovated.
- the BaaS building automation system is extended to further parts of the building or the site.
- Baas building automation functions of BaaS services are optimized.
- improvements of the relationships between the BaaS services are done.
- new added-value services are added to the BaaS building automation system.
- anything is done that requires a change to the BaaS model of the building automation system.

In the optimization phase, the to-be-done optimization is determined and specified based on the analyzed logs, protocols, and reports of measurements and status data from the deployed and running BaaS building automation system and its corresponding technical management system. This means, the optimization phase can be done in parallel to the operation phase – the BaaS building automation system can operate as usually while the tasks of the operation phase are performed.

The actual optimization is implemented by the corresponding stakeholders of the design time phases with the corresponding methods of these phases. The optimization phase has lifecycle feedback loops to the design time phases. Whether a feedback loop to the development, engineering, or operation phase is sufficient depends on which artifacts are optimized. In any case, the optimization has to pass through all lifecycle phases from the starting phase of the optimization to the operation phase. For example, if some artifacts of the development phase are going to be optimized (data point types or BaaS service types), the optimization phase gives feedback to the development phase and the development, engineering, and commissioning phase need to be passed through by the optimization in this order before it can enter the operation phase, again.

It depends on the degree of the optimization whether the whole model of the BaaS building automation system or only parts, even only small, minor parts of it need to be modified. So, the whole design process of the BaaS building automation system may be redone as an optimization, but often the optimization makes only changes to parts of an existing BaaS building automation system.

During the optimization, the deployed BaaS building automation system can operate as usual. Only the commissioning phase is critical in this respect: The amount of changes and the available tool support determine whether on the fly changes to the deployed and operating BaaS building automation system are possible.

4.2 Information View

Data plays a central role in BaaS. Data elements are exchanged between BaaS services and their functions. Structuring data is beneficial for different reasons, including the interoperability of services and the discovery of specific data, functions and services. The information view section describes how BaaS structures data, functions and services. The conceptual design can be mapped to different concrete data formats. Examples are the eXtended Modeling Language (XML) or the JavaScript Object Notation (JSON).

The central information model of BaaS is called BaaS data point. It refers to the term data point that is used to describe a functional entity in the building automation domain (see 4.2.1). This section explains the structure und elements of a BaaS data point.

The section starts with an overview on how data is structured in BaaS. Five information models for the BaaS data, the BaaS feature type, the BaaS data point type, the BaaS service type, and the BaaS service instance are introduced.

4.2.1 From Data Points in the Building Domain to BaaS Data Points

A data point is the central entity in the planning and engineering of building automation systems. The term is defined according to [10], 3.61 as: "*A data point is an input/output function consisting of all assigned information describing fully the points meaning (semantic). The data points' information includes the present value and/or state and parameters (properties and attributes), e.g. signal type, signal characteristics, measured range, unit, and state texts. There are physical and virtual data points. A physical data point is related to a direct or network connected field device within a homogeneous system. A virtual data point can be derived from the result of a processing function, or it is related to a device within a* *different system as a shared (networked) data point. A parameter having its own user address is also a virtual data point."*

In a BaaS system, BaaS data points are implemented by BaaS services.

From the BaaS point of view, the following aspects are pointed out:

- BaaS data points represent (building automation) functionality.
- BaaS data points use data of other BaaS data points as input and/or provide data as output.
- BaaS data points and their functions and data have semantics[1].

More details about BaaS data points are given in this part.

4.2.2 Information Flow between BaaS Services

This section gives an overview on how data is exchanged in BaaS.

The BaaS reference architecture comprises distributed services that offer different functionality such as providing a Hardware Abstraction Layer (HAL) for physical devices (BaaS device, BaaS gateway), or implementing workflows for managing devices in physical spaces (see Figure 3-1).

Figure 4-3 illustrates the communication between two BaaS services. Services can run on different machines. They communicate over a network. Each service offers an interface for remote access to a so-called resource tree. The interface can be implemented in different ways, e.g. using inter-service communication such as CoAP or DPWS. The methods of the interface are specified in the Functional View (Sec. 4.3). The resource tree allows addressing the data provided by a service.

The BaaS reference architecture targets interoperability of services. Interoperability especially includes that data offered by one BaaS service can be consumed and understood by another service. As defined in Section 4.3.2.2.3, all services use the same kind of interface for inter-service communication to access their data.

BaaS data is organized within so called BaaS features that represent functional elements of a BaaS service. Multiple features can be composed into a BaaS data point type. The BaaS data point and its contained BaaS features determine a major part of the resource tree. The BaaS data point specifies the data and functionality that a BaaS service implements. Additional specification information is represented in the so-called BaaS service type. It determines most of the information required to implement a service.

The description shows that each shell in Figure 4-4 adds more meta data. In the following subsections, the different kinds of meta data of a BaaS service are described more in detail.

[1] One of the research challenges of BaaS is the fact, that the semantics of a data point is only available in the know-how of the system engineer. A formal description is in most cases not available.

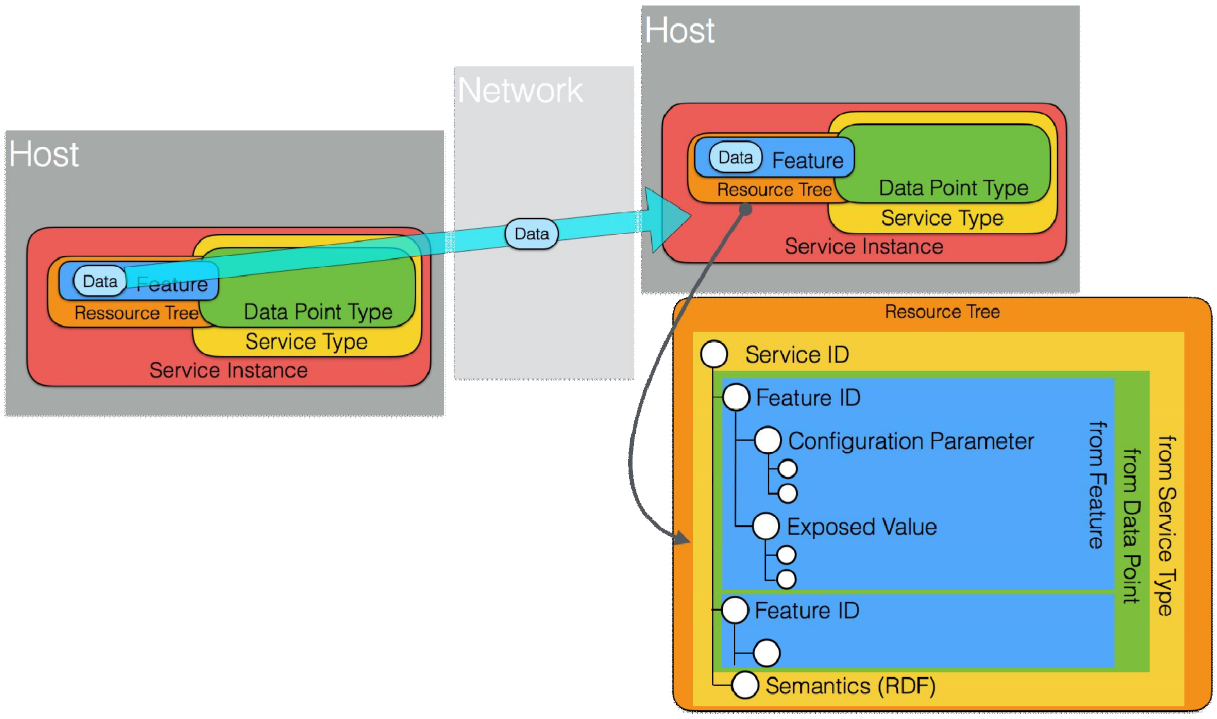


Figure 4-3: Interaction of BaaS Services by Exchanging Data via Resources

4.2.3 Information Model

To structure the information that is offered by a data point, the BaaS information model comprises different layers as shown in Figure 4-4. The upper two layers (basic data types, BaaS data type) model data. They define how data specified in a BaaS data point is structured.

The BaaS service type models a service. It defines which data a service implementing a BaaS data point exposes to other services.

The layers are described in more detail next.

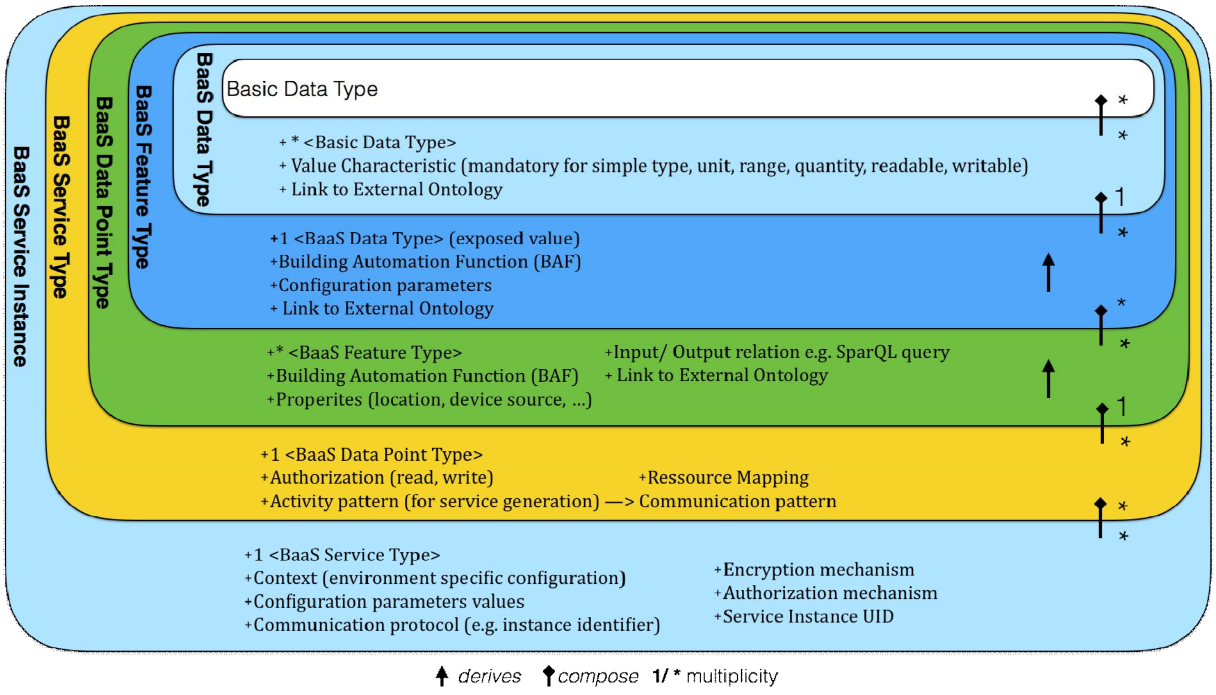

BaaS Service Instance
BaaS Service Type
BaaS Data Point Type
BaaS Feature Type
BaaS Data Type
Basic Data Type
+ * <Basic Data Type>
+ Value Characteristic (mandatory for simple type, unit, range, quantity, readable, writable)
+ Link to External Ontology
+1 <BaaS Data Type> (exposed value)
+Building Automation Function (BAF)
+Configuration parameters
+ Link to External Ontology
+* <BaaS Feature Type>
+Building Automation Function (BAF)
+Properites (location, device source, ...)
+Input/ Output relation e.g. SparQL query
+ Link to External Ontology
+1 <BaaS Data Point Type>
+Authorization (read, write)
+Activity pattern (for service generation) —> Communication pattern
+Ressource Mapping
+1 <BaaS Service Type>
+Context (environment specific configuration)
+Configuration parameters values
+Communication protocol (e.g. instance identifier)
+Encryption mechanism
+Authorization mechanism
+Service Instance UID
derives
compose
1/ * multiplicity

Figure 4-4: Different Abstraction Layers of the BaaS Information Model for Data

The BaaS information model consists of five layers of abstraction: BaaS data types, BaaS features, BaaS data point types, and BaaS services and BaaS service instances. Each layer composes one or multiple entities of the previous layer. The previous layer is shown as more inner layer in Figure 4-4.

In the following, the different layers from Figure 4-4 are introduced. First a description of the functional purpose of each layer is given. Then the additional semantic properties that the layer introduces are listed and explained. Then it is described if members of the layer can be derived (🠉) or composed (🠉). Finally the multiplicity of the composition relationship is described (1, *), giving information how the layers are interconnected.

The following description starts from the basic data types in the inside of Figure 4-4 and proceeds towards its outside to the BaaS service instance.

4.2.3.1 Basic Data Types

The name *basic data type* is used for the types that are provided by the data description language that is used for implementing a concrete data model. As the eXtended Markup Language (XML)/XML Schema languages provide a rich set of data types, it was decided to use the following subset as BaaS basic types:

- Integral numbers
- Fractional numbers
- Strings
- Booleans
- Date, Time and Duration values
- Binary values

Additionally, these types can be combined, renamed and constrained to create simple or complex types when the included basic types are not sufficient.

Some possible constraints on basic types are specifying minimum and maximum values, a regular expression that the value has to match, or a set of values that may be set.

The *basic data types* are used in the definitions of the *BaaS data types*. A *basic data type* can be used in many different *BaaS data types* (composition multiplicity *).

4.2.3.2 BaaS Data Types

The definition of *BaaS data types* is based on the basic data types described in the previous section They are augmented with semantic meta data, that describe the data as physical quanities with various properties (e.g. a unit and a range) and in addition specify whether the data are readable or writable (or both).

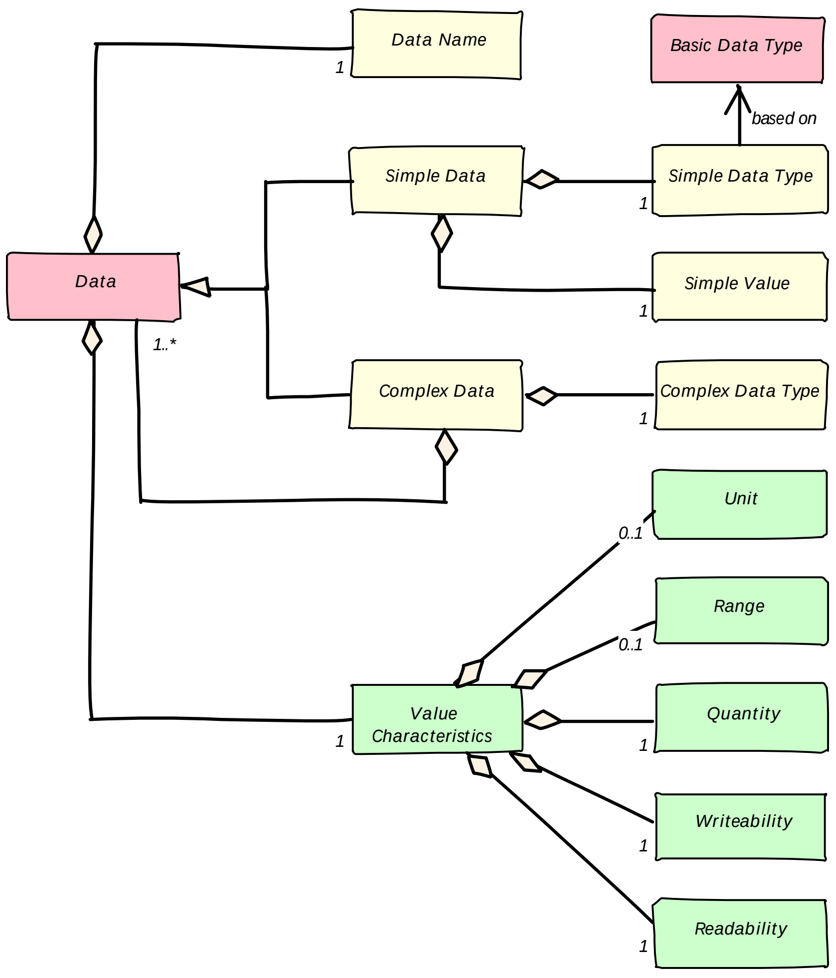


Figure 4-5: BaaS Data Information Model

The definition of *BaaS data,* shown in Figure 4-5, introduces the following semantic properties:

Data Name	Each data item has a globally unique name.
Data	Data is specified in a hierarchical manner: complex data is composed of other data (simple or complex) and have a complex data type. Simple data have a value and a simple data type that refers to a basic data type.

Value Characteristics	The value characteristics define how a value should be interpreted and which restrictions it has. Members of the *value characteristics* are · Unit: Defines the *Unit* of the value, e.g. °C. · Quantity: Defines the *Quantity* of the value, e.g. Temperature. · Range: Defines the *Range* of the value, e.g. 0-100. · Readability and Writeability: Whether the value can be read or written.
Link to external ontology	The BaaS information model defines an ontology that describes only a part of the semantic concepts used in the model. Additional concepts from other ontologies might be used as well. A *Link to external ontology* refers to the specification of a semantic concept that is external to the BaaS information model. For instance, units and quantities are semantic concepts that are adopted from the QUDT ontologies defined by NASA.

A complex *data type* can be based on multiple *basic data types*. Exactly one kind of *BaaS data type* is used as *exposed value* in a *BaaS feature*.

4.2.3.3 BaaS Feature Type

BaaS feature types package functionality provided by building automation components or software. Each feature uses a particular kind of BaaS data as so-called *exposed value*, and contains additional meta-information about the building automation functionality represented by the feature. In addition, a feature may specify parameters that can be used for configuring the feature at engineering time. Similar to an exposed value, each parameter is described as a particular kind of BaaS data.

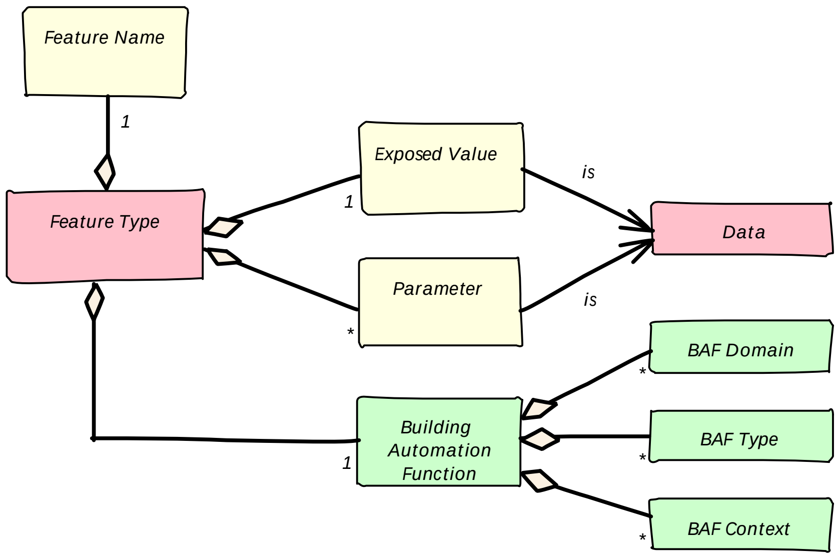


Figure 4-6: BaaS Feature Information Model

A BaaS feature has the following properties:

Feature Name	Each BaaS feature has a globally unique name.
Exposed Value	The exposed value characterizes the data that a feature exposes towards other services
Parameters	Parameters allow specifying configuration information required by the feature. Parameters are instantiated at engineering time.
Building Automation Function (BAF)	A tag-based semantic description of the functionality represented by this feature. The BAF contains three kinds of elements that may occur repeatedly: BAF Domain, BAF Type and BAF Context.
Link to external ontology	The BaaS information model defines an ontology that describes only a part of the semantic concepts used in the model. Additional concepts from other ontologies might be used as well. A *Link to external ontology* refers to the specification of a semantic concept that is external to the BaaS information model. For instance, the semantic tags used in a Building Automation Function (BAF) may be adopted from elsewhere (e.g. from Project Haystack).

A simple example for a feature would be a temperature sensor, which exposes a temperature. It would contain a read-only value of type Temperature, together with the BAF tags “temperature” and “sensing”. Additionally, a configuration parameter could be used to describe a sensor-specific control parameter, such as sensor gain.

4.2.3.4 BaaS Data Point Type

A *BaaS data point type* contains a collection of features. It may optionally be derived from another BaaS data point type. Data points describe a collection of functionalies that may either be provided by a physical device, or may be entirely virtual, i.e. being provided by an algorithm only and not (at least not directly) representing a physical quantity. An example for a virtual functionality is a feature providing an average room temperature that a service computes out of several physical data that are available via BaaS data points representing thermometers.

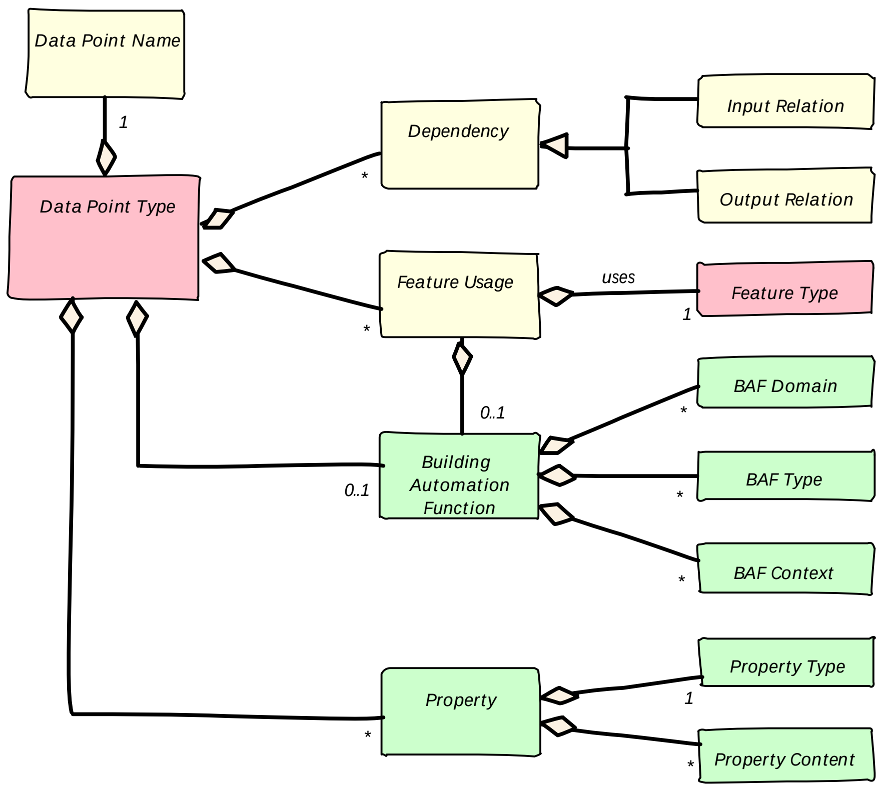


Figure 4-7: BaaS Data Point Information Model

Data Point Name	Each BaaS data point has a globally unique name.
Dependency	Dependencies describe the communication relations this BaaS data point has with other data points. Input relations characterize the data points this data point takes input from, while output relations characterize the data points that this data point sends data to.
Feature Usage	A BaaS data point type can implement one or several features.
Building Automation Function (BAF)	The BAF of the data point may contain additional semantic descriptions that apply to all contained feature alike. The BAF of a feature usage can extend the BAF of the used feature if necessary. The BAF contains three kinds of elements that may occur repeatedly: BAF Domain, BAF Type and BAF Context.
Properties	The properties contain additional semantic information that applies to the data point. An example for such a property would be a location. Properties must specify from which ontology the semantic concepts used for the property are adopted.
Link to external ontology	The BaaS information model defines an ontology that describes only a part of the semantic concepts used in the model.

	Additional concepts from other ontologies might be used as well. A *Link to external ontology* refers to the specification of a semantic concept that is external to the BaaS information model. For instance, the semantic tags used in a Building Automation Function (BAF) may be adopted from elsewhere (e.g. from Project Haystack).

As an example for a BaaS data type that implements multiple features, a single sensor that contains several integrated temperature sensors and a light sensor can be represented by a single data point containing several temperature sensing features and a single light sensor feature. The data point BAF can be used to describe the sensor and functionality in detail, while Properties can be added to describe extra semantic properties, such as the location or size of the referenced device.

4.2.3.5 BaaS Service Type

The *BaaS service type* describes the interaction between data points. This is done by means of exposing values (taken from features) in the form of a resource tree.

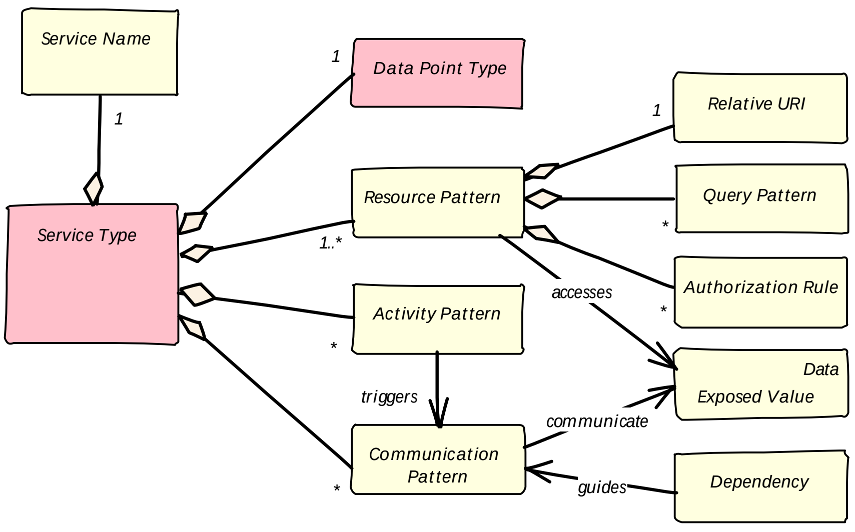


Figure 4-8: BaaS Service Information Model

Service Name	Each BaaS service has a unique name.
Resource Pattern	A Resource Pattern describes how an element of an exposed value is mapped to an element of the resource tree of the service. The collection of all resource patterns provides the specification of the part of the resource tree that represents all exposed values of the service. There are also other parts of the resource tree, e.g. those representing the parameters of the features of the service. These parts do not have to be described by resource patterns as there mapping can be automatically derived from the feature models. A resource pattern contains the following elements: a relative URI; optional query patterns (specifying sets of query parameters

	that may be used when reading or writing the respective resource); optional authorization rules that describe how access to the resource is authorized.
Activity Pattern	An Activity Pattern describe in which manner a service executes an associated communication pattern: · Periodically: The service will regularly execute the respective communication pattern. · Event-triggered: The service will execute a communication pattern only if a specified event occurs, i.e. if a certain element of an exposed value changes or if another service delivers particular data.
Communication Pattern	A Communication Pattern defines how communication on an associated element of an exposed value is performed: · Pull: Communication is initiated by the data sink. · Push: Communication is initiated by the data source.

4.2.3.6 BaaS Service Instance

A BaaS service instance describes the attributes of an instantiation of a BaaS service. Those attributes include specific values for configuration parameters as well as implementation-specific properties such as *Technology Bindings*.

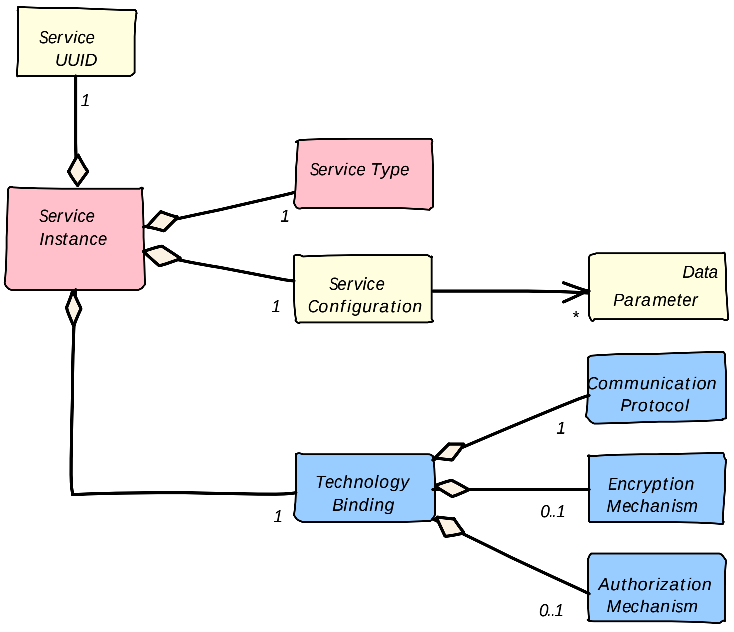


Figure 4-9: BaaS Service Instance Information Model

Service UUID	Each service instance has a unique identifier.

Service Type	The service type refers to the BaaS service type this service implements.
Service Configuration	The configuration values are the concrete values of the parameters defined in the respective BaaS features. Those parameters are either set in the commissioning phase or at run time.
Technology Binding	The technology binding describes the implementation specific technologies that were used. Examples are: · *Communication Protocol* describing the protocols used such as HTTPS, HTTP, or CoAPS, CoAP. · *Encryption Mechanism* describes the concrete encryption mechanisms provided such as Advanced Encryption Standard (AES). · *Authorization Mechanism* describes the supported authentication mechanism such as OAuth.

4.2.3.7 Modelling a Heating System - a Concrete Example

The next section gives a concrete example on modeling a real technical system. It describes the data types, features and data point types needed to represent a basic heating system with BaaS concepts.

The entire heating system (shown in Figure 4-10), encompassing both a hot-water and a heating circuit, consists of the following parts:

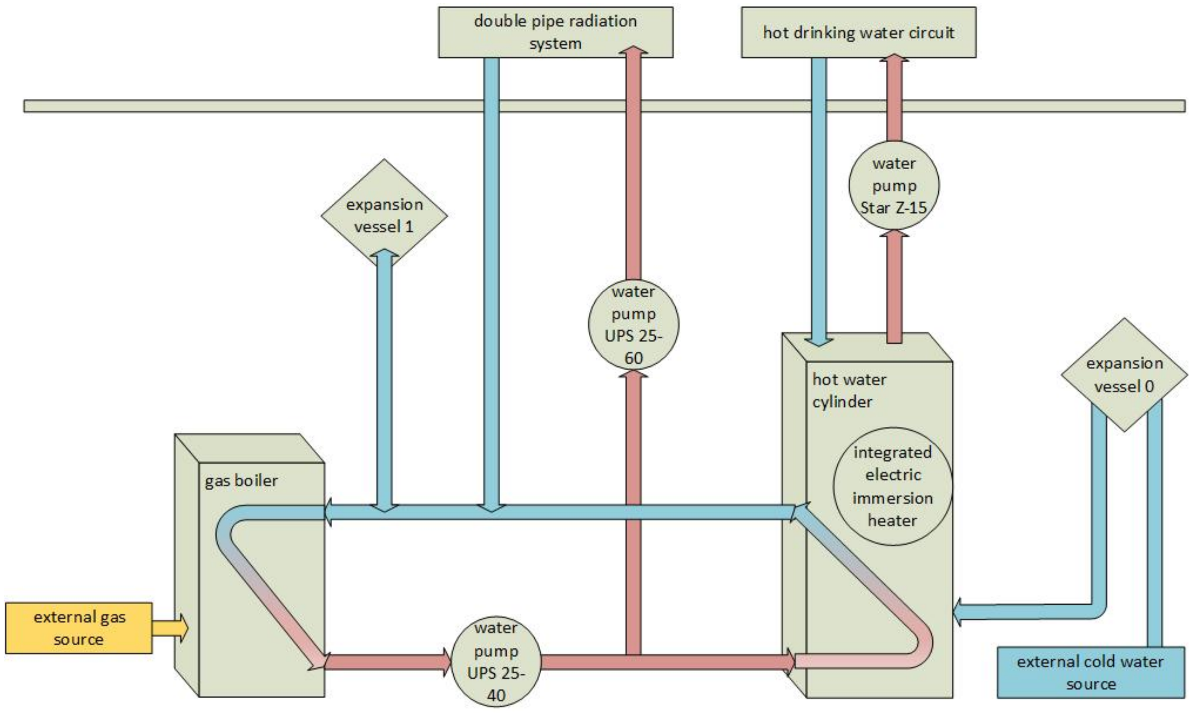


Figure 4-10: Overview of a Heating Circuit.

- Gas boiler (system boiler)
  - This type of boiler works together with a hot water cylinder, but doesn't require a cold water tank, as the system can be filled directly from the water mains. As the heating circuit is closed, the water in the heating system is not lost and will be pumped in circles all the time.
  - The gas boiler consists of several parts:
    - The gas burner is included in the device and is used to heat up the heating water.
    - Three temperature sensors are used to get information about the current system status: Feed flow, return flow, outside air temperature.
    - A controller is used to gather the collected temperature data and use them together with the configuration settings to control the gas burner.
- Hot water cylinder
  - The hot water cylinder is mainly heated by the heating water circuit, but can also use an integrated electric immersion heater. To control the temperature within the cylinder there is a temperature sensor and a small controller.
  - This type of system benefits from having an electric immersion heater within the cylinder which means you can still have hot water even if your boiler breaks down.
- Water pump (3x)
  - Three water pumps are installed throughout the system.
  - The first one (UPS 25-40) is used to pump the water between the gas boiler and the hot water cylinder.
  - The second (UPS 25-60) is used to pump the water up to the heating radiators in the first and second floor.
  - The third (Star Z-15) is used to pump the hot water to the taps on the first and second floor.
- Expansion vessels
  - Expansion vessels are used to keep the hydronic balance in the system

In this example a water pump is modeled. The pump used is a "Grundfos UPS 25-60". It is shown in Figure 4-11.

Figure 4-11: A Grundfos UPS 25-60.

In the following the required BaaS data models are defined for the instrumentation described above. For later use it can be expected that many data models are already defined and can be reused.

For the basic data types, definitions for both flow rate (for the pumps) and temperature (for temperature sensors) need to be defined. The OWL definition of these data types can be found in the Annex in Section A.1.

Using these data types, we define several features and data point types, beginning with the features.

4.2.3.7.1 Defined Features

We define a feature for each element of the system, taking care to not reimplement similar features multiple times. For the heating system, our required features are:

- A temperature sensor
- A temperature setpoint
- A flow sensor
- A flow setpoint

These features will then be used by data point types to implement functionality. The following tables show the representation of these features in the BaaS information model. We omit the description of the temperature setpoint and the flow sensor, since these features are very similar to each other (barring some different semantic tagging).

Name	TemperatureSensor	
Description	Measures a temperature.	
Exposed Value	Name	temperature
	Data Type	temperatureValueType
	Unit	DegreeCelsius
	Quantity	TemperatureUnit
	Writability	false
	Readability	true
BAF	Types	Sensing

Table 2: Temperature Sensor Feature

Name	FlowSetpoint	
Description	Describes the desired throughput of some water-processing device, such as a valve or a pump.	
Exposed Value	Name	throughput
	Data Type	troughputType
	Unit	CubicMeterPerHour
	Quantity	VolumePerTimeUnit
	Writability	true
	Readability	true
BAF	Types	SetPoint

Table 3: Flow Setpoint Feature

As is visible from Table 2, the temperature sensor feature exposes a single readable value which is tagged as containing a temperature value in degree Celsius.

Additionally, the feature is annotated with the BAF type “Sensing”.

A very similar feature is created for flow sensing, with a different unit for the exposed value.

The description of a setpoint is shown in Table 3. Here, the setpoint exposes a single value that is readable as well as writeable. Additionally, the setpoint is tagged with the BAF type “SetPoint” instead of “Sensing”.

4.2.3.7.2 Defined Data Point Types

Now, for every distinct element of the heating system, we define a data point type using the features described in the previous Section.

These features are:

- A boiler
- Three kinds of temperature sensors (feed, return, outside)
- A pump
- A hot water cylinder

The temperature sensors are implemented as three different types, since they differ in their semantic tagging. Since they differ only in a few semantic tags, we will only show the data point type of a single temperature sensor.

Data Point Name	FeedTemperatureSensor		
Description	A temperature sensor connected to the heating system feed pipe.		
Features	Feature	FeedTemperature	
	Reference	TemperatureSensor	
	BAF		
		Domains	Heating; Water
BAF	Types	FeedTemperatureSensor; Sensor	
	Contexts	HeatingSystem#1	

Table 4: Feed Temperature Sensor Model

Since it is the simplest data point type, we will show the model of the temperature sensor first. As can be seen in Table 4, the feed temperature sensor data point type references the *TemperatureSensor* feature, adding additional semantic tagging.

The added tags are BAF domains (Heating, Water), and a “FeedTemperatureSensor” BAF type. Additionally, to locate the sensor, a “HeatingSystem#1” context is given to the sensor. The return and outside sensors are very similar, but annotated with a “ReturnTemperatureSensor” and “OutsideTemperatureSensor” BAF types respectively.

Data Point Name	Pump	
Description	A water pump including a controller and a flow sensor, allowing setting a desired flow rate.	
Features	Feature	currentFlow
	Reference	FlowSensor
	Feature	desiredFlow
	Reference	FlowSetpoint
BAF	Types	Device
	Domains	Heating; Water
	Contexts	HeatingSystem#1

Table 5: Pump Data Point Type

A pump is modeled referencing a flow sensor for showing the current flow rate, and a flow setpoint for setting the desired flow rate.

Additionally, we specify a BAF type "Device" (since each Pump data point corresponds to a single device), "Heating" and "Water" domains (since the pump is part of a heating system and is pumping water), and finally, the "HeatingSystem#1" context for localization.

Data Point Name	HotWaterCylinder		
Description	A hot water storage cylinder with embedded electric heating system and controller.		
Features	Feature	CurrentTemperature	
	Reference	TemperatureSensor	
	BAF	Domains	Water
	Feature	desiredTemperature	
	Reference	TemperatureSetpoint	
BAF	Types	Device	
	Domains	Heating	
	Contexts	HeatingSystem#1	

Table 6: Hot Water Cylinder Data Point Type

The hot water cylinder contains a temperature controller for controlling the temperature of the contained hot water, which requires both a sensor and an input for setting the desired temperature. The temperature sensor is accessible to allow monitoring of the current temperature of the water.

Predictably, the hot water cylinder is modeled as shown in Table 6, referencing a temperature sensor (augmenting it with the “water” BAF domain) and a temperature setpoint. Additionally, the BAF type “Device” is added to show that the object is modeling a concrete physical device. The “Heating” domain indicates the device is part of a heating system, and the “HeatingSystem#1” context localizes the cylinder in a specific heating system.

Data Point Name	Boiler		
Description	A boiler and heating controller, taking input from three temperature sensors (feed, return, outside). A desired input temperature is settable via a temperature setpoint.		
Features	Feature	DesiredTemperature	
	Reference	TemperatureSetpoint	
	BAF	Domains	Water
Input Relations	Name	OutsideTemperatureInput	
	Query	hasBAFContext = HeatingSystem#1 AND hasBAFType = OutsideTemperatureSensor	
	Data Type Filter	temperatureRootType	
	Name	ReturnTemperatureInput	
	Query	hasBAFContext = HeatingSystem#1 AND hasBAFType = ReturnTemperatureSensor	
	Data Type Filter	temperatureRootType	
	Name	FeedTemperatureInput	
	Query	hasBAFContext = HeatingSystem#1 AND hasBAFType = FeedTemperatureSensor	
	Data Type Filter	temperatureRootType	
BAF	Types	Device	
	Domains	Heating	
	Contexts	HeatingSystem#1	

Table 7: Boiler Data Point Type

The data point type defined for the boiler is shown in Table 7. The boiler is a bit more complex than the data point types defined previously, but it serves well to display a benefit gained by semantic tagging: Using *input relations*, the boiler can locate the correct sensors in the heating system, and take them as input for temperature regulation. This is done by specifying a semantic query for all devices in “HeatingSystem#1” that have correct BAF type (Outside-, Return- and FeedTemperatureSensor, respectively). These sensors then provide input to the boiler. Additionally, the boiler references a “TemperatureSetpoint” feature for setting the desired water temperature.

Semantic tagging is added to specify the boiler is a physical device, participating in a heating system, and localize the boiler in “HeatingSystem#1”.

4.2.3.7.3 Defined Service Types

Finally, each of the defined data types will be referenced in a service. On deployment, these services will be instantiated to correspond to the actual elements of the heating system: One boiler service instance, one hot water cylinder service instance, one of each temperature sensor type and three pump instances.

Since these services simply reference the defined data point types and add no additional semantic tagging, their description will not be shown in this document.

4.2.4 Ontology Modeling

BaaS ontologies describe Information View concepts in a formal and machine interpretable way. They consist of BaaS data point (DP) ontology, location ontology, Quantity Kinds and Units (QU) ontology (e.g. QUDT from NASA) and semantic tags from a chosen building automation (BA) ontology (e. g. from Project Haystack), see Figure 4-12.

The DP ontology covers the concepts introduced by the BaaS information model. The ontology has been implemented in the W3C (OWL) language [20].

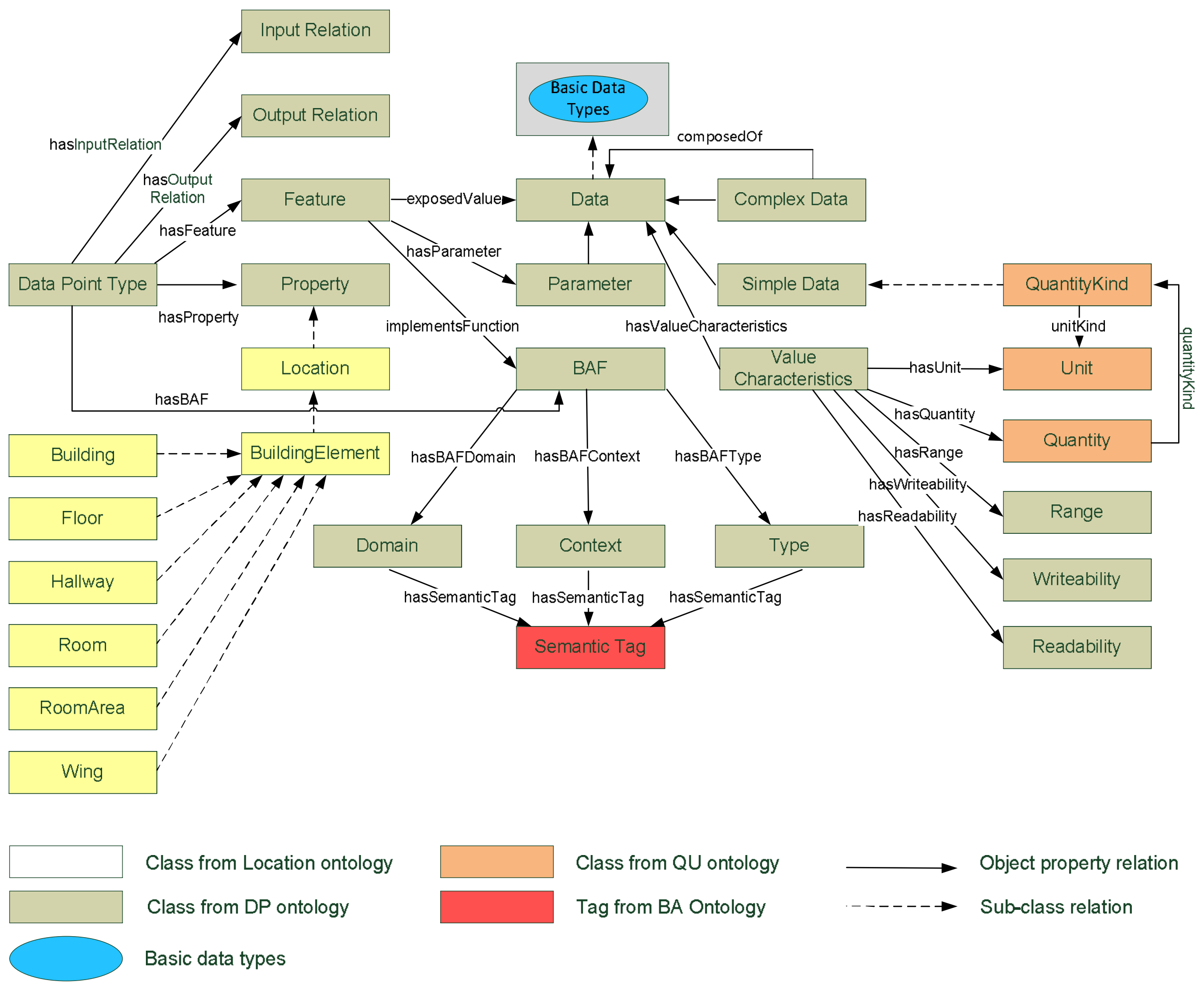


Figure 4-12: BaaS Ontologies, Key Concepts and Relations.

The DP ontology, shown in Figure 4-12, formalizes the specification of a BaaS data point. A data point is composed of BaaS features, which represent functionalities of building automation systems. Furthermore, a BaaS data point uses data of other BaaS data points as input and/or provides data as output. Therefore the DP ontology defines input relations and output relations. Finally, a data point may have properties for example, location information (i.e., information about physical location of a data point). This information is provided in a location ontology (marked in yellow in Figure 4-12 and Figure 4-13). Ontologies for other kinds of properties may be added.

A BaaS feature exposes a particular set of BaaS data. Furthermore, it can be configured via parameters, and contains additional meta-information describing a Building Automation Function (BAF).

There are different types of data. Basic data is data as defined with XSD Schema (e.g., integer, float, boolean etc.), and depicted in Figure 4-12 with one blue circle. Simple data defines BaaS data types in the ontology (e.g., Temperature, Illuminance etc.), and complex data are a combination of data (i. e. simple data and complex data). These types are subclasses of the class data. There exist ontologies that already define various data types. For example, for this purpose in our DP ontology we have reused the ontology for Quantity Kinds and Units (QU) [21]. In this sense DP ontology has been designed to enhance *interoperability* by reusing existing ontologies. For the same reason we employ BA vocabulary in the form of semantic tags because there exist common vocabularies for building automation (BA) systems (e. g. Project Haystack) that can be reused. The interoperability between the BaaS DP ontology and other existing ontologies used in the domain of BA is an important aspect since the BaaS project should offer a BA platform that is extensible from other domains (e.g., energy domain, industry domain etc.). Actually, the general concepts behind BAF are not specific for building automation systems. By defining vocabularies for other domains, the BaaS approach can be applied to those domains as well.

Data is described by value characteristics (e.g., value range or measurement range, defined together with units, data types etc.). Later in this section an example is presented that shows how value characteristics are defined for concrete cases.

The BAF contains three kinds of meta-information: BAF Domain, BAF Type and BAF Context.

Parameters of a feature define how a particular data point is configured (e.g., the sampling rate according to which the data point will deliver its data etc.). The class parameter is used for this purpose (see Figure 4-12).

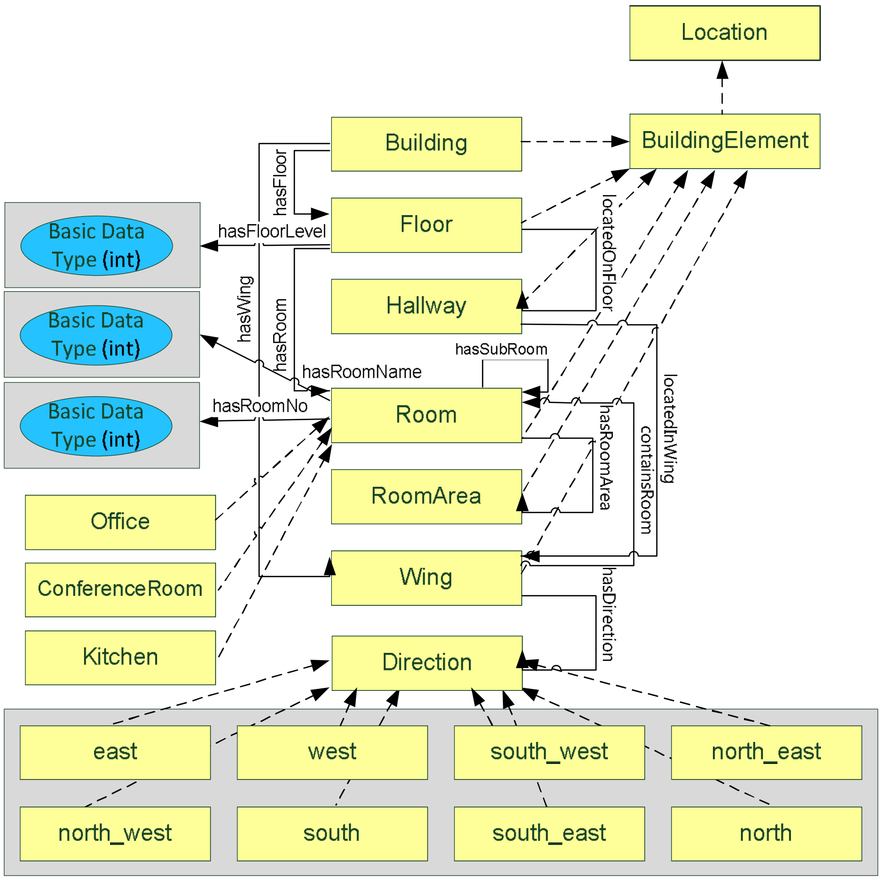


Figure 4-13: BaaS Location Ontology

A more detailed version of location ontology is depicted in Figure 4-13. Apart from the main concept "BuildingElements", it also provides relation between them (e.g., hasFloor, hasRoom, hasWing etc.). Figure 4-13 shows also how each "BuildingElement" can be subclassed (e.g., an Office is a type of Room). The basic concept of direction is specified and used in a relation with the concept of wing. Direction is defined as an enumeration class, which containes a set of directions (e.g., east, west, south, north etc.). We have specified few data type relations to demonstrate how basic data types can be used the cocept of room (e.g., hasRoom, hasRoomNo, and hasFloorLevel).

Finally, let us present an example with the above described BaaS ontologies. Figure 4-14 shows the feed temperature data point. *feed temperature dp* is a data point with a *feed temperature sensing feature*. It is located in *room 4*. The feature provides *temperature* as data. The data point may be used for measuring *water feed* temperature in a *boiler* as described with Haystack tags in the DP Context, Domain, and Type. It is of type *temperature sensor* that samples data every second, used in the Domain *water heating* and in context *feed flow circut*. Minimum and maximum value range for the temperature data has been defined (i.e., -50 and +100, respectively). These values are provided as *float* numbers (f), and the unit of measure is *degree Celsius*. (see Figure 4-14).

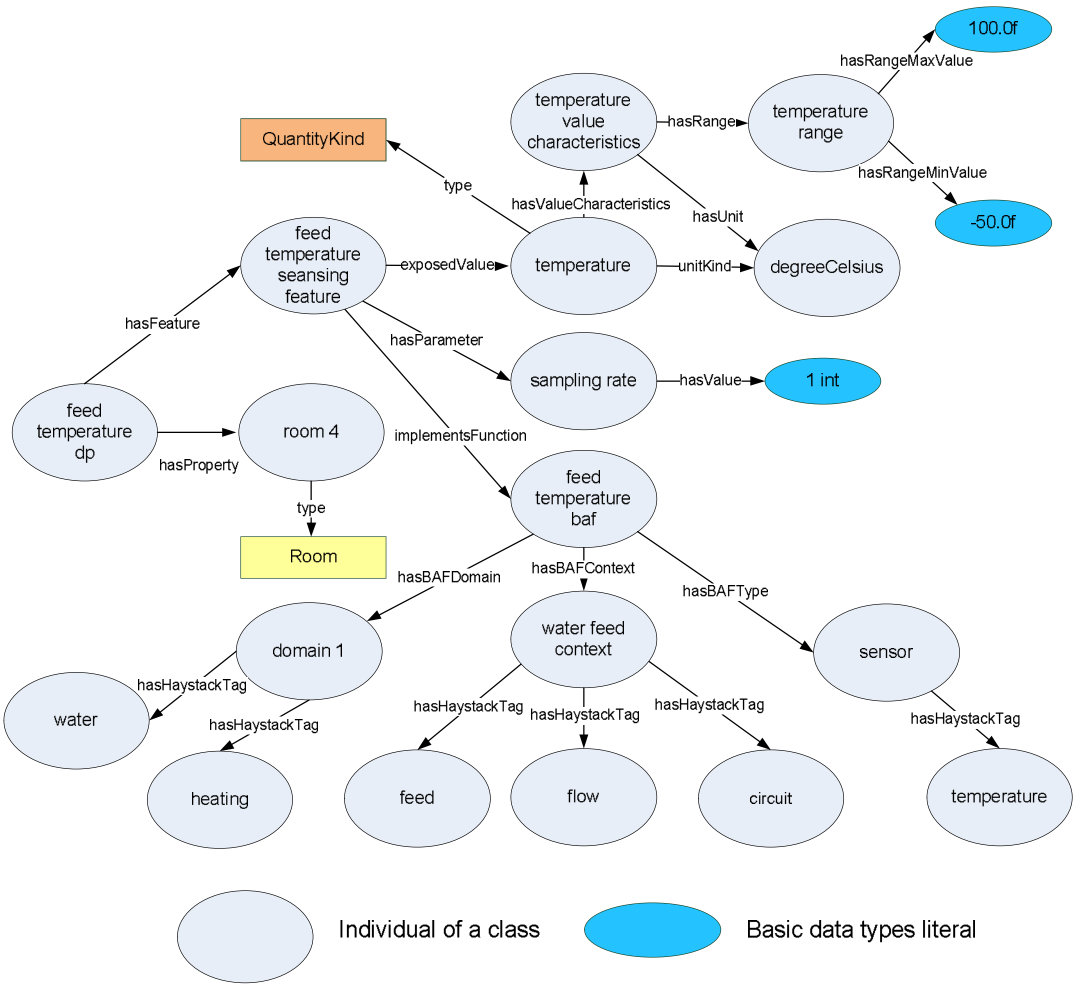


Figure 4-14: Ontology Example: A Feed Temperature Data Point

4.3 Functional View

4.3.1 Motivation

The functional view of the reference architecture aims at the following goals:

- Each functional building block specifies its responsibilities and relations to other functional building blocks.
- The functional view captures the interactions of the functional building blocks. Each lifecycle activity triggered by an actor is be covered by an interaction scenario between functional building blocks.
- Each lifecycle phase of a BaaS system is dealt with separately.
- Finally, an overview of all functional building blocks across all lifecycle phases is presented.

4.3.2 Types of Functional Building Blocks

Section 3.2 introduces the concepts of a BaaS framework and a BaaS runtime environment. This section intends to introduce and give an overview of the different types of functional building blocks being either part of the BaaS framework or the BaaS runtime environment. The terms BaaS framework and BaaS SDK are used in an exchangeable way in this section. The term SDK rather indicates that the BaaS framework also consists of different kind of tools.

4.3.2.1 BaaS Framework/SDK

This section lists the main types of functional building blocks that are used at design time (during the development and engineering phases).

4.3.2.1.1 Editors to Specify BaaS Entities

- are either based on
  - one or more domain-specific languages of the BaaS SDK,
  - existing tools and their respective editors (e.g. using the Protégé tool to express data point descriptions in an ontology format),
- do not have to replace existent IDEs or other development tools (e.g. the manually implemented BaaS service core can be done by any means used by the respective developer),
- allow creating/modifying/deleting certain entities and their relationships according to certain rules and constraints, e.g.
  - BaaS data point types, their properties and their relation to other data points,
  - BaaS services (service types) and their relation to the BaaS data point types they expose,
  - BaaS Constraints, their forms/quality and realization at runtime
- allow choosing from a certain set of pre-defined options (e.g. entities from external ontologies to refer to when creating a BaaS data point type),
- transform the entities to other representation formats (e.g. storage format like rdf for BaaS data point types),
- use BaaS entities from other BaaS information storages (e.g. when specifying a BaaS service the editor uses the BaaS data point information storage),

- generate code according to the selected options (e.g. generate the BaaS service core code skeleton and the glue code to cover the call interaction of the core and the automatically generated code).
- store BaaS entities to one or more BaaS information storages, e.g.
  - the BaaS information storage of data point types,
  - the BaaS information storage of BaaS service types,
  - the BaaS information storage of the BaaS service core implementation,
- specify queries to a BaaS information storage (e.g. to specify the set of BaaS data points a certain BaaS data point has an input/output relation to),
- perform queries to a BaaS information storage (e.g. to retrieve a set of existing BaaS data points according to some criteria)

4.3.2.1.2 Tools to Engineer a BaaS system

- are based on one or more domain-specific languages (DSLs) of the BaaS SDK (rather graphical DSLs),
- use the information of BaaS information storages filled during the development phase, e.g.
  - browse the developed and available BaaS service types
  - browse the defined and available BaaS Constraints on reliability quality options for BaaS services
  - perform queries to retrieve a set of BaaS service instances meeting certain BaaS data point (instance) criteria
- allow creating a functional model of a BaaS system by creating/modifying/deleting certain entities and their relationships according to certain composition rules and constraints, e.g.
  - create BaaS service instances (and indirectly their contained BaaS data point instances),
  - wire BaaS service instances to each other according to its BaaS data point (input/output) relations,
  - specify dependability qualities of BaaS service instances. A system engineer specifies the demand of a service composition being available at least for 99.99%. He should not necessarily know how this requirement can be fulfilled. The BaaS Constraint *availability* is defined at the development phase with the qualities 99.9%, 99.99%, 99.999% and 99.9999%. Besides the definition of different qualities of availability the concrete measures to fulfill these qualities is defined in the development phase as well. E.g. one measure could be: constraint services with a requested availability of 99.999% need to be run in BaaS containers with at least an availability of 99.999%. The services with a demand for availability of "5 nines" can only be redistributed in BaaS containers in compliance with the constraint of the service in case of breakdown.
  - check the consistency of dependability quality constraints across services instances (e.g. a highly-available services instance cannot depend on values from less-available service instances)
- allow to choose from a set of pre-defined options to specify properties of BaaS service instances, e.g.
  - the communication protocol of a BaaS service type (e.g. CoAP, REST, DPWS) to be instantiated,

  - the communication pattern of a BaaS service type with respect to its exposed data points (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)
  - the cryptogrphic protection of values exchanged by the BaaS service type
  - 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)
  - the constraints describing the dependability qualities of service instances. The kind of constraints could have been defined during the development phase.
- generate service adaptor(s) and glue code to integrate adaptors with the service core of a BaaS service type. adaptors bind the BaaS service type to a certain
  - communication protocol (e.g. a CoAP resource tree of a BaaS service of BaaS data point resources)
  - communication pattern, e.g.
    - a BaaS service requiring values from another one gets subscribed if the other BaaS service supports a publish/subscribe communication pattern,
    - a BaaS service just requests the value from a BaaS service instance supporting the request/reply pattern (only)
  - activity type, e.g.
    - an active adaptor including a scheduler/timer to trigger the current value processing of a data point exposed by a BaaS service and the storage of the latest value
  - security mechanism, e.g.
    - to encrypt/decrypt the incoming/outgoing values of a BaaS service instance
    - to use a certain authorization mechanism like OAuth in order control the access of requests to a data point of a BaaS service (after having to identified the originator of a request by some other authentication mechanism)
- generate deployable BaaS service packages to deploy the BaaS service instances during commissioning. Such a package contains
  - the executable program of the BaaS service instance
  - the configuration of a BaaS service instance, e.g. specifying the (relative) address (e.g. URI) of the BaaS service
- generate management rules for the technical management system according to service/system constraints
  - rules follow the ECA paradigm (Event-Condition-Action)
  - detected system changes (events) and related fulfilled conditions (condition) lead to automatic reconfiguration and adaptation of the runtime system (action) to assure the demand system quality in terms of BaaS Constraints
- fill up the BaaS service instance registry with the BaaS service software packages and BaaS service instance descriptions
- allow to bind BaaS data points to BAS legacy systems and their endpoints

4.3.2.1.3 Tools to Commission a BaaS system

- allow creating a deployment/system model of a BaaS system by creating/modifying/deleting certain entities and their relationships according to certain composition rules and constraints, e.g.
  - create BaaS containers with certain properties like resource capacities,
  - deploy BaaS devices/container software packages,
  - create BaaS container/device instances,
  - assign BaaS service instances to BaaS containers/devices according to matching resource requirements and capacities,
  - deploy BaaS service software packages on BaaS container
  - specify computing nodes,
  - assignment of BaaS container/device/gateway instances to computing nodes

4.3.2.2 BaaS Framework/SDK and BaaS Runtime Environment

This section lists the main types of functional building blocks that are used at runtime/operation time. Some of them (e.g. the BaaS information storages) are also already used at design time.

4.3.2.2.1 BaaS Information Storage

- a stored collection of certain BaaS entity types (e.g. the collection of all BaaS data point descriptions)
- the data of an information storage can be used (imported/queried and/or filled) by a BaaS editor/tool during the design phases

4.3.2.2.2 BaaS Registries

- a stored collection of BaaS entity type instances to be used at installation time and runtime
- the data of a BaaS registry can be queried during the runtime phases, e.g.
- the BaaS registry of BaaS service instances (i.e. the deployable bundles) and their descriptions
- the BaaS registry of the addresses (e.g. the URL) of all running BaaS service instances

4.3.2.2.3 BaaS Software Libraries to be used by Code Generation Mechanisms when Developing and Engineering BaaS Services

- communication protocol bindings/adaptors (e.g. CoAP, DPWS)
- communication pattern adaptors
- activity model adaptors (e.g. active by regularly calculating the values of the data point, or passive by calculating values on request)
- security adaptors (e.g. adaptors based on OAuth, DTLS and an access control model of roles, permissions and role assignments)
- management libraries
- BAS legacy system access/query libraries

4.3.2.2.4 BaaS Software Packages to be used when Engineering a BaaS System

- BaaS container packages
- BaaS gateway packages (e.g. a BaaS BACnet gateway library)

4.3.2.2.5 Technical Management of a BaaS System

When talking about technical management we are talking about management agents monitoring and/or controlling resources or managed objects. In the context of BaaS these resources are BaaS nodes, BaaS containers and BaaS services. These resources need to provide a pre-defined interface to be accessible by the technical management agents. The interface defines both functions to be implemented by the resources and management information like variables for actual resource properties.

All resources providing the management interface can then be monitored and configured homogeneously and independently of the resource type. This results in a highly maintainable and extendable feature which can deal with newly introduced resources or resource types without changing the management agents or structure. Only for resource type specific management capabilities the agents need to be updated accordingly.

As BaaS nodes, containers and services may be quite numerous within a building it needs to be possible to divide the resources into several management domains. This means that resources can be put into one or more management domains and within each domain specific management rules are applied (e.g. for lighting control other management behavior is used than for building security). To further improve the scalability the management agents can be organized in an hierarchy which also raises the overall stability of the management system.

Altogether the technical management system can be used to adapt the system to changing use cases. For instance, when different customers share the same (meeting) room, each customer has specific needs on lighting and heating. Even these preferences can change in the course of the meeting. These can be reflected by the technical management and therefore the behavior of the building installation. Another example would be a multi tenancy access control where each customer is represented by a tenant. It must be ensured that management data cannot be accessed across tenant boundaries. The technical management system will ensure that access to the management data is only granted to authorized users.

In terms of the functional view there are some facts that need to be respected:

- Management agents are distributed depending on the infrastructure of the BaaS system. Therefore they are not deployed together with BaaS nodes or containers by default.
- Engineering has not to be considered, because the management is controlled/ parameterized by the management rules, which are automatically derived from the runtime model.

Further detail regarding the technical management in the BaaS reference architecture is described in the dependability perspective in Section 5.2.

For further details on technical management in general have a look at Section 5.3.

4.3.3 Overview of Functional Building Blocks

There are the following functional building blocks used per and across phases.

- Functional View of the Development Phase
    - BaaS Data Point Editor

  - BaaS Data Point Information Storage
  - Query Wizard
  - BaaS Service Editor
  - BaaS Dependability Editor
  - BaaS Service Information Storage
  - IDE for Implementing the BaaS Service (and Legacy Protocols)
  - BaaS Legacy Profile as Software
  - BaaS Legacy Profile Tool
- Functional View of the Engineering Phase
  - BaaS Engineering Tool
  - Query Wizard
  - BaaS Service (Instance) Registry
  - BaaS Software Libraries
  - BaaS Legacy Analysis Tool
- Functional View of the Commissioning Phase
  - BaaS Commissioning Tool
  - Technical Management System
  - BaaS Service Instance Registry
  - BaaS Container Software Packages
- Functional View of the Operation Phase
  - BaaS Service Instance Registry
  - BaaS Container/Device
  - Technical Management System
  - BaaS Legacy Gateway
  - BaaS Service Instance
- Functional View of the Optimization Phase
  - The Analysis/Optimization phase will be covered during the second iteration of specifying the BaaS reference architecture.

4.3.4 Functional View of the Development Phase

This section describes the functional building blocks of the BaaS SDK during the development of BaaS data points and BaaS services. It also covers some of the collaboration scenarios of these building blocks.

4.3.4.1 BaaS Data Point Editor

This editor provides means for the domain engineer to specify data point types according to a certain structural description (Section 4.2), rules and constraints how to specify BaaS-specific data point types.

Responsibilities	Actor	Collaborations
Create/modify/delete a BaaS-specific data point type and specify its properties (e.g. temperature data point).	domain engineer	none
Specify the building automation functionality in a semantic way (possibly using only a set of keywords/tags from the Haystack project).	"	"
Specify the value(s) and whether it(these) are read-only or read-write	"	"
Specify the input relations to needed input values of other data point types. This might be a simple reference to one or more data point types or it might be a complex semantic query that contains several criteria in order to resolve the set of data points compliant to the input relation at engineering time.	"	"
Specify the output relations to data point types the values of which are derived based on the value of this data point. This might be a simple reference to one or more data point types or might be a complex semantic query that contains several criteria in order to resolve the set of data points compliant to the output relation at engineering time.	"	"
Specify the "processing mechanism" of the value computation. This is needed for the software engineer to implement the functionality and can be provided in plain text. There is no need of a computer readable form.	"	"
Specify the configuration properties / parameter of the data point type (e.g. time to keep the stored data in a history database).	"	"
Select defined data types provided/required by the data point		
Specify the selective or extendible options of each data point type property from which to choose or which to extend later on during the lifecycle. For example a temperature data point type is specified in a way that the unit of the value (as part of the value characteristic) can be chosen from Celsius or Fahrenheit later. This is also an example that a property might be linked to a set of entities from other ontologies like the "units.owl" ontology. This ontology captures all kind of units (as Celsius, Kelvin or Fahrenheit).	"	"
Define data types including their characteristics	domain engineer, software engineer	none
Possibly the BaaS platform SDK introduces an own mechanism of high-level data types like (numeric, alphabetic, enumerations) to be used by the BaaS editors of the development phase. Those can be refined (eg. Numeric->float), restricted (minimum, maximum, regular expression), structured and named to be reusable.	"	"

Responsibilities	Actor	Collaborations
Specify the value characteristics (e.g. semantic meaning, unit) of the value(s) of the data point type.	"	"
Define data type bindings for each programming language supported by the BaaS platform. These data type bindings are implemented by the code generation mechanism of the BaaS service editor.	software engineer	"
Use a pre-defined sub-set of entities from external ontologies for certain data point properties.	domain engineer	
The editor provides support of using entities from certain pre-defined external ontologies in order to specify some properties of a data point type. There are two options:	"	none
1. Import whole external ontology could be imported as own entities into the editor.	"	external ontology, external ontology tool
2. Query and select the used entities to be imported from an external ontology into the editor	"	external ontology tool
Store/load/search/query data point types.	domain engineer	
Store/load all (or a subset of) data point types to/from the BaaS information storage of data point types.	"	data point information storage
Search/browse the existing collection of BaaS data point types.	"	data point information storage
Express a query against the existing collection of BaaS data point types and along certain criteria. Such queries are possibly needed to specify the input/output relations of a data point type to other data points.	"	query wizard
Perform the query and present the results.	"	data point information storage
Check consistency and constraints of data point types (e.g. naming conventions, missing data point relations). There are two options: Or the domain engineer triggers global consistency checks across all data point types.	editor and domain engineer	constraint/ consistency checker
1. Local constraints on one or several data point types can be checked on the fly in an automated way by the editor displaying "errors" to the domain engineer.	"	"
2. Or the domain engineer triggers global consistency checks across all data point types.	"	"

The BaaS data point editor could be a tool (and its editor) to specify Ontologies (e.g. Protégé). Such tools allow performing queries over the set of data point types. In addition, using external ontologies might also be supported by an ontology tool. But checking the consistency and constraints of individual data point types or across all data point types is normally not supported by ontology tools.

4.3.4.2 BaaS Data Point Information Storage

Responsibilities	Actor	Collaborations
Store data point types.	BaaS data point editor	none
Perform the search on/browse/query the set of existing data point types.	BaaS data point editor	“

The BaaS data point information storage could be an ontology (concepts, individuals, rules stored as e.g. RDF or turtle format files) combined with a semantic database, or a set of domain-specific models (as files) or a repository. An ontology with a semantic database supports semantic queries (e.g. SPARQL). Both of them could be combined with a reasoning mechanism. Queries and reasoning are not supported by domain-specific models. Domain-specific frameworks allowing building domain-specific language editors often offer support of constraints/consistency checking. A repository also adds versioning support to data point type descriptions.

4.3.4.3 Query Wizard

Responsibilities	Actor	Collaborations
Supports the user in specifying queries on a BaaS information storage	BaaS editor	
Support the user in specifying a query on the BaaS data point information storage	BaaS data point editor	domain engineer, software developer
Support the user in specifying a query on the BaaS service information storage	BaaS service editor	software developer

The query wizard of data points / services could be a separate domain-specific language to construct, trigger and present the result of a data point / service query.

4.3.4.4 BaaS Service Editor

This editor provides means for a software developer to specify (and possibly develop) BaaS service types.

Responsibilities	Actor	Collaborations
Create/modify/delete BaaS service types.	software developer	
Specify the BaaS data point types to be provided by a BaaS service type.	“	BaaS data point information storage.
Decide on the selectable or extendible data point type properties.	“	ontology

Responsibilities	Actor	Collaborations
For example, the service type computes temperature values of its exposed temperature data point type in Celsius only.		
Optionally select dependability qualities for all instances of this service type per default. These could be overridden at engineering time. The dependability qualities are specified and provided by the BaaS dependability editor. This also implies the generation of skeletons implementing certain managed object Interfaces suiting the respective dependability quality.	"	BaaS dependability editor
Specify any need to protect the access to certain provided data points (either by authentication/authorization and/or confidentiality levels). (The actual protection is specified and realized at engineering time.)	"	none
Load/search/query BaaS data point types.	software developer	
Search/browse the existing collection of BaaS data point types.	"	data point information storage
Express a query against the existing collection of BaaS data point types and along certain criteria. Such queries are possibly needed to specify the data point types provided by a BaaS service.	"	query wizard
Perform the query of data point types and present the result.	"	data point information storage
Store/load/search/query BaaS service types.	software developer	
Store/load all (or a subset of) service types to/from the BaaS information storage of service types.	"	data service information storage
Search/browse the existing collection of BaaS service types.	"	data service information storage
Express a query against the existing collection of BaaS service types and along certain criteria in order to identify other implemented services (already exposing certain data points/dependability/security qualities etc).	"	query wizard
Perform the query of service types and present the result.	"	data point information storage
Bind a BaaS service to a BAS legacy system.	software developer	
Use the code generation hooks provided by the BaaS legacy profile tool.	"	BaaS legacy profile tool
Generate the needed BaaS service type skeleton and the hook skeletons to be implemented by the software developer. Different service hooks could also be implemented at different times. Each of the hook implementation could be stored as different packages to be re-used for the implementation of different service type skeletons.	software developer	

Responsibilities	Actor	Collaborations
Generate the BaaS service type skeleton containing the hook skeletons below to be implemented. Generate the BaaS service core hook skeletons for each supported data point of the service Such a core hook gets the input values (of the respective input relations) as input parameters and returns the value of the data point. Its implementation is a realization of the processing mechanism specified by the data point.	“	code generator (including programming language bindings for the value types)
Generate the BaaS service management adaptor hook skeleton (i.e. from a pre-defined set of possible BaaS managed object interfaces). This hook has to be implemented manually. It is called by the management adaptor of the service in order to monitor and manage the service instance. The code generation rules are specified as part of the BaaS dependability editor and used by the BaaS service editor as part of the code generation.	“	BaaS dependability editor

A BaaS service editor actually could be a domain-specific language and a (generated) editor for it. Since it is about software development the domain-specific language should rather be a textual one. Ideally, the domain-specific language can be embedded into a set of programming languages supported by BaaS (e.g. Java, C/C++). The domain-specific language framework should include a model transformation language and engine which could be used for code generation purposes. It could also be used for BaaS specific language bindings of data point value types to programming language types. In addition, it should allow using the API of any other library of supported languages in order to trigger queries to the BaaS data point information storage and present the results.

4.3.4.5 BaaS Dependability Editor

The BaaS dependability editor allows creating dependability qualities to be used by other functional building blocks like the BaaS service editor and the BaaS engineering tool. Each dependability quality and its possible values are linked to a set of transformation rules. These support the creation of management rules for each BaaS service type specification using any of the supported dependability qualities.

Responsibilities	Actor	Collaborations
Specify BaaS dependability qualities/constraints.	software developer	BaaS service editor
Specify the BaaS dependability quality/constraint set which could be used for the specification of BaaS service types and instances.	“	none
Specify the value set of each dependability/quality constraint.	“	“
Provide the dependability quality/constraint set to the BaaS service editor.		BaaS service editor
Specify the managed object interfaces.		BaaS service editor

Specify the managed object interface(s) for each BaaS dependability quality/constraint. This interfaces have to be implemented if a BaaS service type uses such a dependability quality as part of its specification.	“	none
Provide the code generation rules to the BaaS service editor. These rules are used to generate the managed object skeletons of a BaaS service using a respective dependability quality.	“	BaaS service editor.
Specify the transformation rule(s) to generate management rules.	software developer	BaaS technical management system, BaaS engineering tool
Specify the transformation rule(s) for each BaaS dependability quality/constraint to generate management rules for each.	“	BaaS technical management system (Management Rules)
Provide the transformation rules to the BaaS engineering tool.	“	BaaS engineering tool
Specify the BaaS resource types to be used for BaaS service and BaaS container specifications.	software developer	BaaS service editor, BaaS legacy profile tool, BaaS commissioning tool
Specify the kind of resources and their value types to be used.	“	none
Provide the resource types to the BaaS service editor, the BaaS legacy profile tool and the BaaS commissioning tool.		BaaS service editor, BaaS legacy profile tool, BaaS commissioning tool

A BaaS dependability editor actually could be a domain-specific language and a (generated) editor for it. The underlying framework to build domain-specific languages and editors should support the integration of different domain-specific languages across the generated editors. This functionality could be used to integrate the artifacts of the BaaS dependability editor with the BaaS service editor and the BaaS engineering tool.

4.3.4.6 BaaS Service Information Storage

The BaaS service information storage aims at providing some persistent means to store and query BaaS service types.

Responsibilities	Actor	Collaborations
Store BaaS service types.	BaaS service editor	
Store the specification of each BaaS service type (e.g. the set of provided data points).	“	none
Store the manually implemented code (i.e. the BaaS service core hook implementation and its BaaS managed object hooks) of each BaaS service type. Alternatively, store the implementation of the service core hook separately in order to re-use is for several service type implementations.	“	“
Perform the search/browse/query on the set of existing service types. Or perform the search/browse/query on the set of existing	BaaS service editor	

service core hook implementations of a data point.		
Control the access to BaaS service types.	BaaS service editor	
Control the access of a BaaS service Provider (company or person) to its BaaS service type implementations only.	"	none / Authentication instance

The BaaS service information storage could be a repository (including versioning support) like current configuration management systems. The repository would contain the service specification as a domain-specific file, the generated and the manually implemented code files for each BaaS service type (and possibly build files to generate the software package for a BaaS service). In addition, the BaaS service information storage could store hook implementations only (and their build files to generate a software library out of it to be used by different service implementations).

On top the BaaS SDK would need to provide the query support described above. Access control is also a common feature of repository implementations.

4.3.4.7 IDE for Implementing the BaaS Service (and Legacy Protocols)

Responsibilities	Actor	Collaborations
Implement the BaaS service core hook(s) along the processing mechanism description(s) of its exposed data points.	software developer	none
Implement the BaaS management adaptor hook as a BaaS managed object.	"	"
Implement the BAS legacy protocol stack (if not available).	"	"

The selected IDEs (Integrated Development Environment) depends on the programming languages supported by the BaaS platform. But an IDE does not have to be used by the BaaS software developer. It is an optional building block. In addition, the BaaS platform does not prescribe the usage of any specific IDEs.

4.3.4.8 BaaS Legacy Profile as Software Package

The binding between BaaS services and a legacy technology (like an existent building automation system) is very dependent on the legacy technology to be used. In most cases this requires the development of individual software packages to achieve that goal. To support the development of a legacy technology mapping there should be a common abstract interface between BaaS and the legacy systems. For example, in the context of the JMEDS framework the assumption is that each technology can be separated into three logical groups: entities which contain/host services, services and functions provided by services. All technologies are then mapped to this schema. A mapping to the BaaS schema is called a profile in this section. Support for the mapping of a legacy technology to BaaS data points (e.g. BACnet objects to BaaS data points) has to be implemented and provided as software libraries.

Responsibilities	Actor	Collaborations
Map the legacy communication protocol to the BaaS communication protocol. Implement the mapping.	software developer	Used by BaaS legacy profile tool (and BaaS service editor), Uses existent legacy stack library.
Map the legacy communication patterns to BaaS communication patterns. Implement the mapping.	“	“
Map BaaS addressing schema of BaaS service communication endpoints to legacy communication addressing schema.	“	“
Map legacy security mechanisms to BaaS mechanisms (encryption, authentication, authorization). Implement the mapping.	“	“

4.3.4.9 BaaS Legacy Profile Tool

Besides having to implement BaaS legacy profile libraries, a BaaS legacy profile tool is also needed. It allows writing code generation mechanisms to be used by the BaaS service editor for creating BaaS services which map exposed data point(s) to legacy technologies entities.

These code generation rules could be used by several hooks of the BaaS service core code generator to introduce legacy specific implementation details making use of the supportive BaaS mapping software packages.

Another responsibility of the profile tool is the specification of requirements to be used during the engineering phase to find suitable Baas devices / containers for the deployment of the gateway.

Responsibilities	Actor	Collaborations
Specify a BaaS legacy profile	software developer	
Specify resource requirements of the profile implementation on a hosting BaaS legacy gateway (e.g. computing power, RAM/ROM, phys. connectors). The kinds of resources that can be specified are defined by the BaaS dependability editor.	“	BaaS dependability editor
Specify code generation rules that can be hooked into BaaS generators to generate legacy specific core code for BaaS service types.	“	IDE

4.3.4.10 Collaborations at Development Time

This section contains two exemplary collaborations of the functional building blocks at development time.

4.3.4.10.1 Browse/Create/Modify/Delete BaaS Data Points

The following Figure 4-15 shows the collaboration of functional building blocks in order to browse existent data point types and to create a new one.

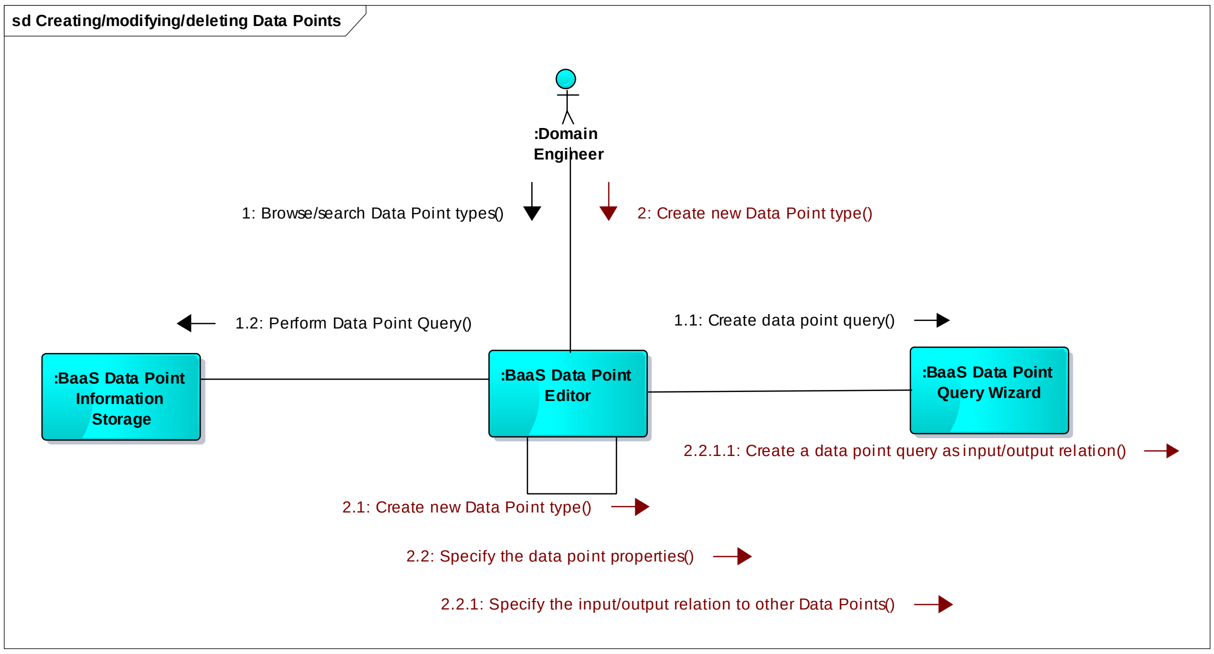

sd Creating/modifying/deleting Data Points
:Domain Engineer
1: Browse/search Data Point types()
2: Create new Data Point type()
1.2: Perform Data Point Query()
1.1: Create data point query()
:BaaS Data Point Information Storage
:BaaS Data Point Editor
:BaaS Data Point Query Wizard
2.2.1.1: Create a data point query as input/output relation()
2.1: Create new Data Point type()
2.2: Specify the data point properties()
2.2.1: Specify the input/output relation to other Data Points()

Figure 4-15: Collaborations on Data Points

The first collaboration is about the domain engineer browsing the existent set of data point types by using the BaaS data point editor. 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 presents the query results to the domain engineer.

The second collaboration captures the creation of a new data point type on a high-level. The domain engineer uses the BaaS data point editor to create a new data point type and to specify all its properties. For instance, she also uses the query wizard to express a data point query as the input or output relation of the new data point type to other data point (instances). This query is not performed on the BaaS data point information storage at development time. It is rather applied at engineering time on the data points of BaaS service instances. Thus, the query might already refer to context information of the data point like information on the location (e.g. a query to retrieve all Light Switch data points on a certain office/floor of the building).

Similar collaborations apply when modifying or deleting an existent data point type.

4.3.4.10.2 Specifying and Implementing a Service type

The following Figure 4-16 shows the collaborations of functional building blocks in order to browse existent BaaS service types and to create a new service type by specifying and implementing it.

The first collaboration is about the software developer browsing the existent set of service types by using the BaaS service editor. The service editor uses the BaaS service query wizard to support the software developer in expressing a service query. The service editor uses the

BaaS service information storage to perform the query and presents the query results to the software developer.

The second iteration sketches the specification of a new BaaS service type. The software developer determines the data point type(s) to be exposed by the new service type by using the 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 all service properties (e.g. the dependability qualities created by the BaaS dependability editor and used by the service editor). As a third step the same or some other software developer triggers the code generation of the service editor to create the service, service core and service managed object skeletons. The software developer uses the generated artifacts to implement the skeletons, creates the implementation artifacts manually by using his favorite IDE and stores adds these to the BaaS service type description of the BaaS service information storage.

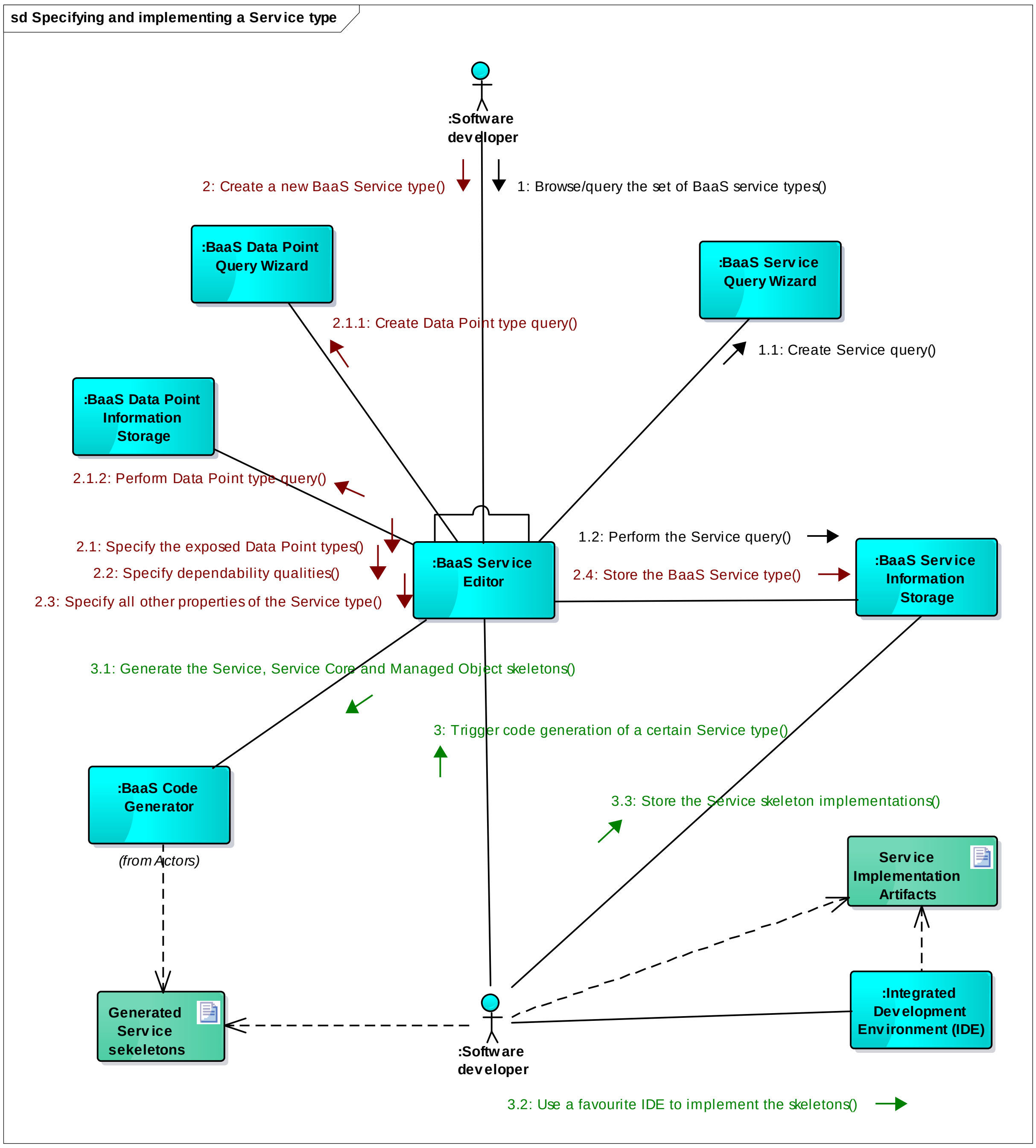

sd Specifying and implementing a Service type
:Software developer
2: Create a new BaaS Service type()
1: Browse/query the set of BaaS service types()
:BaaS Data Point Query Wizard
:BaaS Service Query Wizard
2.1.1: Create Data Point type query()
1.1: Create Service query()
:BaaS Data Point Information Storage
2.1.2: Perform Data Point type query()
2.1: Specify the exposed Data Point types()
2.2: Specify dependability qualities()
2.3: Specify all other properties of the Service type()
:BaaS Service Editor
1.2: Perform the Service query()
2.4: Store the BaaS Service type()
:BaaS Service Information Storage
3.1: Generate the Service, Service Core and Managed Object skeletons()
3: Trigger code generation of a certain Service type()
3.3: Store the Service skeleton implementations()
:BaaS Code Generator
(from Actors)
Service Implementation Artifacts
Generated Service sekeletons
:Software developer
:Integrated Development Environment (IDE)
3.2: Use a favourite IDE to implement the skeletons()

Figure 4-16: Collaborations about BaaS Service Types

4.3.4.10.3 Specify Dependability Qualities and Transformation Rules

The following Figure 4-17 shows the collaborations of functional building blocks in order to specify dependability qualities and resource types. It also shows how these means of specifying a BaaS service and its instances are made available to the functional building blocks of the engineering and commissioning phases.

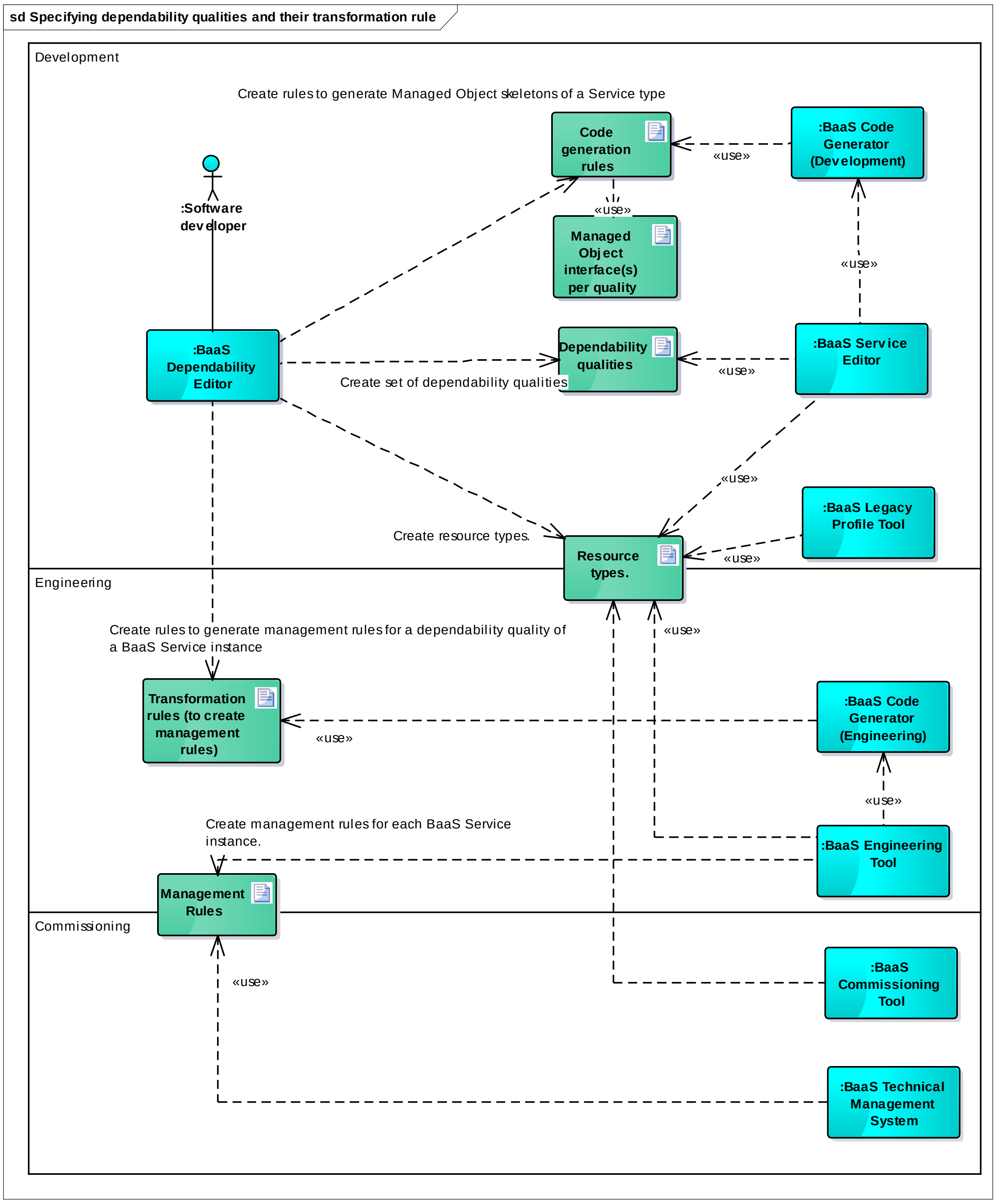

sd Specifying dependability qualities and their transformation rule
Development
Create rules to generate Managed Object skeletons of a Service type
:Software developer
Code generation rules
«use»
:BaaS Code Generator (Development)
«use»
Managed Object interface(s) per quality
«use»
:BaaS Dependability Editor
Dependability qualities
«use»
:BaaS Service Editor
Create set of dependability qualities
«use»
:BaaS Legacy Profile Tool
Create resource types.
Resource types.
«use»
Engineering
Create rules to generate management rules for a dependability quality of a BaaS Service instance
«use»
Transformation rules (to create management rules)
«use»
:BaaS Code Generator (Engineering)
«use»
Create management rules for each BaaS Service instance.
:BaaS Engineering Tool
Management Rules
Commissioning
«use»
:BaaS Commissioning Tool
:BaaS Technical Management System

Figure 4-17: Collaborations of the BaaS Dependability Editor

Figure 4-17 illustrates the impact of the software developer using the BaaS dependability editor on other BaaS functional building blocks and their usage. The dependability editor enables the software developer to create a set of dependability quality attributes to be used when specifying or modifying a new service type at development time by using the BaaS service editor. This implies that the set of supported dependability quality attributes is made available to the BaaS service editor. In addition, this set is also integrated into the BaaS engineering tool, thus allowing the system engineer to override and tune the dependability quality attribute settings on service instance level. (This collaboration between the BaaS dependability editor and the BaaS engineering tool is omitted due to figure layout reasons only).

Each dependability quality attribute implies a set of managed object interfaces for any service type using the respective quality attribute as part of their specification. The set of managed object interfaces of a dependability quality attribute has to be manually implemented for each service type having specified the respective quality attribute. Thus, the BaaS code generator of the BaaS service editor has to include the respective code generation rules to generate the managed object implementation skeletons of a service type. The BaaS code generator uses the managed object interfaces in turn to generate their respective implementation skeletons.

Each dependability quality attribute also implies a set of management rules for each service instance using it. (This occurs either by inheriting the specified quality attribute of the service type or by overriding it at service instance level or by using it on instance level only). Thus, the BaaS dependability editor has to generate transformation rules to be leveraged by the BaaS engineering tool (and its code generator called BaaS code generator (engineering)). It uses the transformation rules in order to create the respective management rules for each service instance which has to be managed by the BaaS technical management system by taking these service instance-specific management rules as input.

The last collaboration of the BaaS dependability editor is about allowing the software developer to create a set of resource types to be used by several other tools across the phases. Resource types are used by the BaaS service editor to optionally specify the resource requirements of a BaaS service type at development time. They are also integrated They are also integrated into the BaaS legacy profile tool. The BaaS commissioning tool also makes use of them to specify the resource capabilities of BaaS containers.

4.3.5 Functional View of the Engineering Phase

This section describes the functional building blocks of the BaaS SDK during the engineering of BaaS service instances and the wiring of them. It also covers some of the collaboration scenarios of these building blocks.

4.3.5.1 BaaS Engineering Tool

The BaaS engineering tool is used to create a functional model of a BaaS system of service instances and their wiring. In addition it generates all artifacts like the BaaS service software packages and their specific management rules and stores them for later usage during the commissioning and operation phases.

Responsibilities	Actor	Collaborations
Use the information of the BaaS service information storage	system engineer	BaaS service information storage, query wizard
Search/browse for (complex) value types and the BaaS services behind.	“	BaaS service information storage
Search/browse/query and load the developed and available BaaS service types.	“	query wizard

Responsibilities	Actor	Collaborations
Search/browse the defined and available BaaS Constraints of BaaS services (e.g. their specified dependability quality properties).	“	BaaS service information storage
Use the available engineering information of the BaaS service instance registry of the current BaaS system installation.	system engineer	Baas service instance registry
Specify queries to retrieve a set of BaaS service instances meeting certain BaaS data point (instance) criteria or certain BaaS Constraints.	“	“
Create and specify BaaS service instances according to the following pre-defined options supported by the engineering tool. Store the BaaS service instances.	system engineer	BaaS service instance registry
Select the communication protocol of a created BaaS service instance (e.g. CoAP, REST, DPWS).	“	none
Select the communication pattern of a BaaS service instance with respect to its exposed data points (e.g. from the options like get/set, publish/subscribe, unsolicited set).	“	“
Select the activity model of the BaaS service type (e.g. active (i.e. the value computation of a data point is triggered on a regular basis independently from any specific request) or passive (i.e. the value computation is triggered on demand on serving the request of another BaaS service instance).	“	“
Select the security options (e.g. encryption of values, authentication, replay protection (e.g. by using challenge-response)).	“	“
Specify the security access model to the BaaS service instance and its exposed data points (e.g. by specifying the required role(s) in order to get access to a data point of the BaaS service instance.	“	“
Select the supported constraints specifying the dependability qualities of service instances.	“	“
Set the configuration properties/parameters of a BaaS service instance.	“	“
Store the BaaS service instances.	“	BaaS service instance registry
Create a functional model of the whole BaaS system according to certain composition rules and constraints. Store the functional model.	system engineer	constraint checker, BaaS service instance registry
Wire BaaS service instances to each other according to their BaaS data point (input/output) relations.	“	none
Check the consistency of communication patterns of wired BaaS service instances.	“	constraint/ consistency checker

Responsibilities	Actor	Collaborations
Check the consistency of dependability constraints across services instances (e.g. a highly-available services instance cannot depend on values from less-available service instances)	“	constraint/ consistency checker
Store the information about the wiring of BaaS service instances.	“	BaaS service instance registry
Add and store additional context information to the data points of BaaS service instances. (e.g. location)	“	BaaS service instance registry
Generate the software package of each BaaS service instance.	system engineer	code generator, software build system
Generate the service adaptor(s) and glue code to integrate adaptors with the service core of a BaaS service type.	“	code generator
Generate the communication protocol adaptor for a BaaS service instance and integrate it with the service core of a BaaS service type (e.g. the CoAP resource tree of the BaaS service of BaaS data point resources).	“	“
Generate the communication pattern adaptor of a BaaS service instance to provide its values.	“	“
Generate the communication pattern adaptor of a BaaS service instance to consume the required values. For example the BaaS service instance requiring values from another one gets subscribed if the other BaaS service supports a publish/subscribe communication pattern. Or it just requests the value from a BaaS service instance supporting the request/reply pattern (only).	“	“
Generate the activity adaptor of a BaaS service instance (e.g. generate an active adaptor including a scheduler/timer to trigger the current value processing of a data point exposed by a BaaS service and the storage of the latest value).	“	“
Generate the security adaptor (covering encryption/decryption (e.g. DTLS) and authentication protocols (e.g. OAuth)).	“	“
Generate the configuration of each BaaS service instance.	“	“
Build the service instance software packages.	“	software build system

Responsibilities	Actor	Collaborations
Generate the management rules for the BaaS system.	system engineer	BaaS dependability editor, technical management tool
Generate management rule(s) for each BaaS service instance and possibly for subsets of service instances. This is done by using the transformation rules of the BaaS dependability editor as transformation hooks.	"	BaaS dependability editor
Store the management rules as part of the technical management tool.	"	technical management tool
Bind to an installed BAS legacy systems	system engineer	legacy profile-analysis tool
Bind BaaS data points of BaaS services to BAS legacy systems.	"	"

A BaaS engineering tool captures a domain-specific language and associated with a constraint/consistency checker and code generation and model transformation mechanisms. Being used by system installers graphical languages seem to be a natural fit. The exchange artifact (and storage format) among domain-specific languages like a BaaS service editor and the BaaS service instance registry could be ECore of the Eclipse Modeling framework.

4.3.5.2 Query Wizard

Responsibilities	Actor	Collaborations
Supports the user in specifying queries on a BaaS service (instance) registry	system engineer	BaaS engineering tool
Supports the user in specifying queries on a BaaS service instance registry according to certain criteria (e.g. data point characteristics, service types and their properties like the value type)	"	"

The query Wizard for BaaS service instance queries could be an own domain-specific language to construct, trigger and present the result of a BaaS service instance query. It is used by the BaaS engineering tool which takes the query specification to perform it on the BaaS service instance registry.

4.3.5.3 BaaS Service (Instance) Registry

Responsibilities	Actor	Collaborations
Store the functional model of a BaaS system.	BaaS engineering tool	
Store the specification of each BaaS service instance and their wiring.	"	none
Store the generated BaaS software packages for each BaaS service instance.	"	"

Responsibilities	Actor	Collaborations
Perform the search/browse/query on the set of existing service instances.	"	"
Control the access to BaaS functional model.	"	"
Control the access of a BaaS service provider (company or person) to the whole BaaS functional model of a BaaS system.	"	"

4.3.5.4 BaaS Software Libraries

This section lists the BaaS software libraries to be used by code generation mechanisms when engineering BaaS services.

Responsibilities of separate software libraries	Actor	Collaborations
BaaS libraries supporting the usage of BaaS Communication protocol bindings (e.g. CoAP, DPWS). BaaS libraries supporting the usage of certain BaaS communication pattern. Libraries supporting the usages of different activity models (e.g. active by regularly calculating the values of the data point, or passive by calculating values on request). Libraries for certain security options (e.g. adaptors based on OAuth, DTLS and an role-based access control model	none	Used by the BaaS engineering tool

4.3.5.5 BaaS Legacy Analysis Tool

The legacy analysis tool is used to collect information about a legacy system to integrate to. This information can then be used to find a suitable gateway services capable of making the legacy system available in the BaaS domain. As this task is highly depending on the legacy technology it needs to be implemented individually for each.

Responsibilities	Actor	Collaborations
Analyze software components of a legacy system to get an automated overview about APIs.	system engineer	none

4.3.5.6 Collaborations at Engineering Time

4.3.5.6.1 Creating a Functional Model

The following Figure 4-18 shows the collaborations of functional building blocks in order to create a functional model at engineering time.

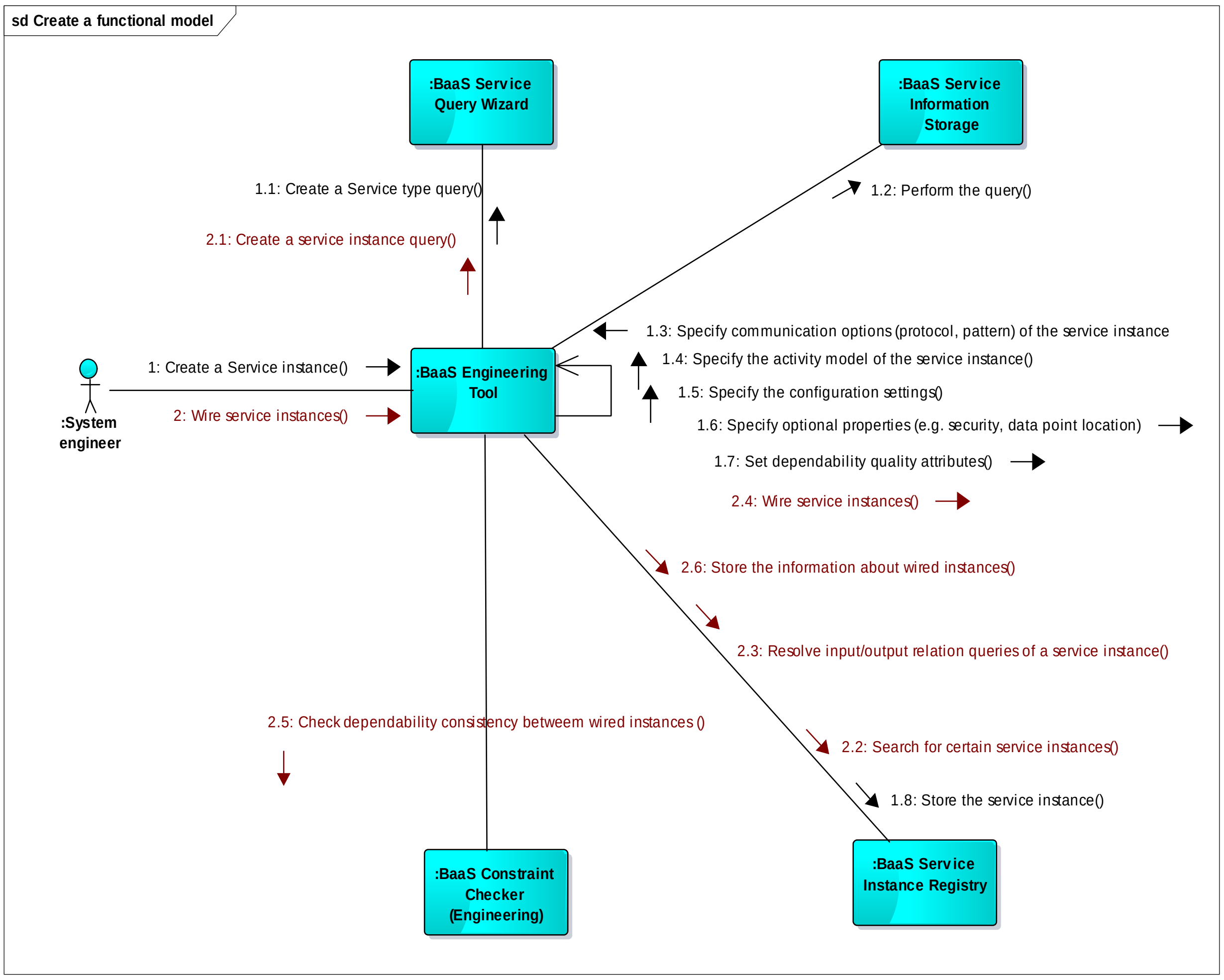

sd Create a functional model
:BaaS Service Query Wizard
:BaaS Service Information Storage
1.1: Create a Service type query()
1.2: Perform the query()
2.1: Create a service instance query()
1.3: Specify communication options (protocol, pattern) of the service instance
1.4: Specify the activity model of the service instance()
1: Create a Service instance()
:BaaS Engineering Tool
1.5: Specify the configuration settings()
2: Wire service instances()
1.6: Specify optional properties (e.g. security, data point location)
:System engineer
1.7: Set dependability quality attributes()
2.4: Wire service instances()
2.6: Store the information about wired instances()
2.3: Resolve input/output relation queries of a service instance()
2.5: Check dependability consistency between wired instances ()
2.2: Search for certain service instances()
1.8: Store the service instance()
:BaaS Constraint Checker (Engineering)
:BaaS Service Instance Registry

Figure 4-18: Collaborations of the BaaS Engineering Tool

The first collaboration captures the creation of a service instance. It is straightforward by letting the system engineer use the BaaS engineering tool to search the BaaS service information storage for certain service type she wants to create instances of by using a query. The system engineer uses the engineering tool to specify the properties of the newly created service instance (like the communication protocol and pattern, its activity model, its configuration properties and dependability quality attributes).

The second collaboration is about wiring matching service instances. Wired service instances have to match according to their input/output relations, along their communication patterns and their dependability quality attributes. Resolving service instances matching the input/output relation of a certain service instance is also performed by queries on the set of existent BaaS service instances stored by the BaaS service instance registry. The BaaS constraint checker of the BaaS engineering tool supports checking constraints and the consistency of the whole functional model, e.g. by checking whether the dependability quality attribute settings of wired service instances are consistent to each other.

4.3.6 Functional View of the Commissioning Phase

The commissioning phase covers the deployment, the installation and the update of a BaaS system. The first step of installing a BaaS system is to create a deployment/system model. It specifies which BaaS containers have to be installed on which computing nodes. In addition, BaaS service instances have to be assigned to BaaS containers. Subsequently, the deployment model has to be realized by deploying BaaS container software packages to the computing nodes and installing them. All software packages (including their configuration) of the BaaS service instances a BaaS container hosts to have to be deployed and installed as well.

Similarly, deploying any additional BaaS container instance or BaaS service instances after the initial BaaS system has been setup has to follow the same steps, i.e. updating the deployment/system model and performing the additional deployments and installations.

4.3.6.1 BaaS Commissioning Tool

The commissioning tool allows creating the deployment/system model according to certain composition rules.

Responsibilities	Actor	Collaborations
Create a deployment/system model.	system installer	BaaS dependability editor, BaaS service instance registry, constraint/consistency checker
Create BaaS container instances and specify their resource capabilities.	“	BaaS dependability editor
Specify existing computing nodes and their IT properties (e.g. the IP addresses) and assign the BaaS container instances to them.	“	none
Load BaaS service instances of the functional model and assign all service instances to BaaS containers. Resource requirements of BaaS service instances and resource capabilities of BaaS containers must be met in a compliant way.	“	BaaS service instance registry, constraint/consistency checker
Set the address of BaaS service instances. Store the information as part of the BaaS service instance registry.	automatically	BaaS service instance registry
Resolve the address of the service instances a certain BaaS service instance is wired to. Get the service instance address from the BaaS service instance registry.	automatically	BaaS service instance registry
Realize the deployment/system model.	system installer	BaaS container, computing node
<authorized> Transfer BaaS container software packages to the computing nodes. Install the software.	“	computing node
<authorized> Transfer the software packages (including the configuration) of BaaS service instances to the BaaS container(s). Install the software	“	BaaS container software package
<authorized> Transfer BaaS software library binaries to the BaaS container(s).	“	BaaS container software package

Responsibilities	Actor	Collaborations
Install the technical management system.	“	technical management system, computing node
Deploy the software of the technical management system to computing node(s).	“	“
Deploy the management rules for BaaS service instances to the technical management system.	“	“

4.3.6.2 Technical Management System

Responsibilities	Actor	Collaborations
Start and activate the management functionality according to deployed configuration and management rules of BaaS services.	system installer	BaaS commissioning tool

4.3.6.3 BaaS Service Instance Registry

The BaaS service instance registry stores the information added to the BaaS service instance descriptions at commissioning time. The communication address of BaaS service instances is such an example. This information is needed at commissioning time for wiring BaaS services. But it is also used and might be updated at operation time. This is also the rationale why it is kept as part of the BaaS service instance registry.

Responsibilities	Actor	Collaborations
Serve location and query requests for BaaS service instance descriptions and communication details (e.g. service address)	BaaS commissioning tool	none

4.3.6.4 BaaS Container Software Packages

Responsibilities	Actor	Collaborations
Lifecycle Management of the hosted BaaS service instances of a running BaaS container.	BaaS commissioning tool	none
Status monitoring of the hosted BaaS service instances of a running BaaS container.	“	“
Enforcement of management rule actions (e.g. the activation of a redundant service instance).	“	“

4.3.7 Functional View of the Operation Phase

At runtime all functional building blocks like BaaS containers, BaaS services, BaaS legacy gateways, the technical management system and the BaaS instance registry have to start, get initialized and run. The following sections contain the responsibility of each of them at operation time.

4.3.7.1 BaaS Service Instance Registry

Responsibilities	Actor	Collaborations
Serve location and query requests for BaaS service instance description and communication details (e.g. communication address)	BaaS service instances	none
Serve updates of BaaS service instance descriptions (e.g. the update of the communication address in case a BaaS service instance is replaced by another instance on another computing node).	BaaS container	technical management system
Allow (de-)registering of running BaaS service instances.	BaaS container	none

4.3.7.2 BaaS Container/Device

Responsibilities	Actor	Collaborations
Lifecycle Management and monitoring (start, stop, update, get status) of BaaS service instances.	automatically	technical management system
Registration of hosted and running BaaS service instances. De-registration of stopped BaaS service instances.	Automatically	BaaS service instance registry
Enforcement of management rule actions (install, uninstall, update, get information)	technical management system	none
<authorized> Access to transfer new versions of software packages of service instances to the container	system installer	BaaS commissioning tool

4.3.7.3 Technical Management System

Responsibilities	Actor	Collaborations
Monitor the quality of service (QoS) and compliance of service level agreements (SLAs) of service instances.	automatically	BaaS container
Dynamically reconfigure BaaS containers and BaaS service instances according to the actions of management rules.	automatically	BaaS container

4.3.7.4 BaaS Legacy Gateway

Responsibilities	Actor	Collaborations
Start and stop running.	triggered by the BaaS container/device	none
Send periodic keep-alive messages to the hosting BaaS container.	automatically	BaaS container
Find and bind required service instances	automatically	BaaS service instance registry
find and bind legacy systems	system installer	legacy analysis tool (e.g. installation plan)

4.3.7.5 BaaS Service Instance

Responsibilities	Actor	Collaborations
Start and stop running.	triggered by the BaaS container/device	none
Send periodic keep-alive messages to the hosting BaaS container.	automatically	BaaS container
Find and bind required services	automatically	BaaS service instance registry

4.3.8 Functional View of the Optimization Phase

All previous phases have to facilitate some monitoring of the BaaS system in order to able to do some analysis and optimization later. Normally, optimization results in adapted configurations of operating BaaS services or the development/engineering/commissioning of new BaaS services (or even a new version of a BaaS service).

This section will be added as part of the second iteration of the reference architecture specification.

4.4 Behavioral View

The Behavioral View focuses on the technical communication with virtual data points and BaaS services. The communication with physical data points depends on and is delegated to the used underlying field bus, such as KNX, BACnet or any bus developed in future. Furthermore, the communication is analyzed and described from inside a virtual communication layer and thereby abstracted from requirements specific to particular applications. This also means that the established results exhibit the characteristics of reusable components. This will become evident in Section 4.4.4, where typically usages from the perspective of an application are composed into more complex application patterns.

The Behavioral View applied the following methodology for the identification of communication patterns: First, general aspects of communication relevant for BaaS were identified and analyzed with respect to their impact. Impacts are mainly consequences with negative side effects. For example, an impact can be increased effort for implementation, need for more powerful hardware and the like. These general aspects and impacts are addressed in Section 4.4.1. Second, the Application Cases from deliverable D02 are taken as a basis and analyzed with respect to the implied communication by the respective project partner. Already in this analysis step the general aspects were considered. The results of this step are Application Case specific tables containing identified need for communication, including the purpose and the required characteristics. These tables are documented in Section 4.4.2. Third, the tables were discussed and updated until a unified understanding was achieved. Afterwards, multiple occurrences of communication with same characteristics were identified and proposed as communication patterns needed in a BaaS reference architecture and any BaaS platform implementation respectively. This major outcome of the Behavioral View is documented in Section 4.4.3. Last but not least, the Behavioral View used these identified communication patterns to compose more or less complex communication patterns from the application point of view. These can be regarded as recommendations, i.e. a developer may lookup whether an application pattern is proposed that matches his needs and then adopt it.

4.4.1 General Aspects of Communication and their Impact

Before the Application Cases as documented in deliverable D01 [4]are analyzed with respect to communication requirements, general aspects of communication and their impact on the operation must be understood and considered. This is of major importance, because such requirements imply design decisions on a conceptual level and these have to be followed by any realization of the BaaS reference architecture. Further, certain design decisions can result in considerable additional implementation efforts, additional expenses for more powerful building automation hardware, reduced performance in the operation phase or even limited scalability.

In the following, general aspects are grouped into areas of interest. Each area introduces its scope, explains the respective impacts of design alternatives in the context of building automation systems and, if possible, indicates a favorable design alternative. Any concrete implementation of the BaaS reference architecture may arrive at a different trade-off and may deviate from these recommendations.

State and Session Management: This area focuses on two closely related aspects. The first aspect is whether the communication requires management of state information, i.e. whether the entities involved in the communication have to create, maintain, synchronize and cleanup information that is only required to enable the communication at all. The second aspect is whether the communication provides a session concept, i.e. whether it is connection-oriented or connectionless. It is well-known that connection-oriented protocols such as TCP actually manage state information. For example, during connection setup the state information is allocated and initialized by the entities, during communication phase it is implicitly synchronized and updated by adding data to exchanged messages and this state is finally cleaned up in the connection teardown phase. In essence this means that communication providing a session concept often relates to connection-oriented and stateful communication. This potentially includes a delay imposed by the connection setup phase before communication of business data can start. Further, the additional teardown messages are necessary to clean up and avoid indefinite allocation of resources. Both of these impacts must be balanced with the total volume of business data to be transmitted and the resources available at the entities, especially if these are building automation components with hardware constraints. However, any secure communication will require management of state information at the endpoints, which may be realized on higher layers, e.g. by BaaS services.

An alternative to connection-oriented is connectionless communication. This is also known as datagram based communication. In this type of communication there is neither a connection setup nor teardown phase; any business data is directly communicated between the entities. In essence, it allows immediate communication between entities in a, at least potentially, stateless fashion. However, lack of reliability on the communication layer may impose higher implementation efforts for all services, which can be mitigated or even balanced by a model-based service development approach which is followed by BaaS. Whether the communication can be truly stateless depends on design decisions in other areas as well as on decisions made during implementation of concrete BaaS services.

In the context of building automation systems where small to medium volumes of business data is expected to dominate the communication between constrained hardware components, connectionless communication in stateless fashion seems favorable. If enforcing statelessness on communication layer implies management of state information on higher layers, e.g. by BaaS value-added services then these should favor a soft-state approach. This means, any information about state is maintained for limited period and must be explicitly kept alive by some “heart-beat”. Otherwise the state is automatically reclaimed and the respective resource can be freed. Such a principle is adopted, for example, by both, BACnet and CoAP.

Temporal Dependency: This aspect mainly questions whether the initiator of a communication will strictly depend on the finally established result. For example, an initiator may not be in position to continue its work without a response to its sent request, e.g. sensing the current feed-flow temperature. In such cases a dependency exists, which mandates for an adapted handling of messages on the communication layer in terms of time. In BaaS, such dependencies are related with the concept of synchronous communication in order to reflect the fact that the initiator is blocked until the communication succeeds. If no such dependency exists, the communication can take place in the background, which also means that it can be asynchronous.

Reliability of Delivery: This aspect typically relates to the State and Session Management area above. This is because messages lost due to transmission failures, distortions or even hardware outages need to be repeated to achieve reliable delivery. This repetition typically implies numbering of messages, caching of sent messages that are yet not confirmed to be successfully delivered, additional messages and network load for confirmation of reception etc. To summarize, it typically implies management of state including the respective impacts. Further, it must be noted that it is impossible to establish absolute reliability over a best-effort communication network.

In the context of building automation systems with its constrained resources it is, therefore, favorable to use reliable communication where necessary and otherwise use unreliable communication; the capability to cope with failures is anyhow required. In BaaS, the principle to opt for reliable or unreliable communication as appropriate for the respective application is named selective reliability. This principle is detailed below.

The following cases of selective reliability are distinguished. The two major characteristics of selective reliability are the reliability style and confirmation style as summarized by the following tables.

Reliability Style	Description
Mandatory unreliability	Any message sent must not be confirmed. This means that there is no possibility foreseen by the communication layer to check and confirm delivery. This style implied no overhead at all. First off, this style is mainly mentioned for sake of completeness. No communication pattern identified by the Behavioral View uses this style.
Mandatory reliability	Any message sent must be unconditionally confirmed by some mechanism. This means that it is not foreseen by the communication layer to disable the confirmation; even not for some messages. This style can imply overhead on the communication network.
Optional reliability	Any message sent may be confirmed or not. This means that the communication layer provides an interface to enable and disable confirmation of delivery. This may be provided on a per message or session basis.

In BaaS, the mandatory and optional reliability styles are favored.

Confirmation Style	Description
Implicit confirmation	This style applies if an involved entity "naturally" must react to a received message by sending a return-message. The reception of the return-message can be interpreted as a confirmation of delivery for the initial message. Thus, there is no need for any additional messages and the confirmation of delivery happens implicitly. This style avoids unnecessary overhead.
Explicit confirmation	This style applies if implicit confirmation cannot be used. This means, if an entity receiving a message needs not to send a return-message as a "natural" reaction then additional efforts have to be taken for confirmation of delivery. In essence, this entity is forced to send a message solely to confirm delivery, which implies overhead.

In BaaS, both confirmation styles are equally favorable. However, the implicit confirmation style should be applied where possible to reduce the load on the communication network.

4.4.2 Analysis of Application Cases

In the following tables we concisely document the communication requirements from the application case point of view as described above.

The column with the header "Multi." describes the multiplicity of the communication, i.e. whether the communication is unicast (UC), multicast (MC), broadcast (BC) or even geocast (GC).

AC1 - Smart Meeting Room					
#	Description	Purpose	Type	Multi.	Requirements
1	Discover required sensors	discovery	Request/response	UC or BC	Discovery mechanism, asynchronous, reliable
2	Query required sensors	get	Request/response	UC	Unique data point addresses, asynchronous, reliable
3	Set required actuators	set	Request/response	UC	Unique data point addresses, asynchronous, reliable
4	Event subscription (e.g. "Notify me when person enters room")	subscribe	request (Subscribe)	UC or BC	Unique data point addresses, asynchronous, reliable
5	Event notification (e.g. "Person entered room")	notify	response (Publish)	UC or BC	Unique data point addresses, asynchronous, reliable
6	Keep-alive (status of virtual data points)	continuous data tracking	request/response	UC or BC or MC	Unique data point addresses, asynchronous, reliable

AC2 - Predictive Automation					
#	Description	Purpose	Type	Multi.	Requirements
1	discover 'meeting room booking service'	discovery	request/response	UC or BC	depends on implementation-detail
2	Discover sensors and actuators (temperature, room heating, electro mechanic window etc.)	discovery	request/response	UC or BC	see above
3	resolve to address	name-address resolution	request/response	UC or BC	depends on implementation-detail
4	read data point (e.g. sensor value / temperature)	on demand data query	request/response	UC	synchronous with timeout, connectionless, implicitly acknowledged = selective reliability
5	write set point (e.g. actuator value / temperature set value	on demand data query	request/response	UC	synchronous with timeout, connectionless, explicitly acknowledged =

AC2 - Predictive Automation					
	of heating)				selective reliability
6	register for COV at data point (e.g. temperature)	registration or subscription, continuous data tracking	request/re sponse (with aim soft-state publish-subscribe)	UC	synchronous with timeout, connectionless, explicitly acknowledged = selective reliability
7	keep-alive for COV (e.g. temperature)	registration or subscription, continuous data tracking	request/re sponse (with aim soft-state publish-subscribe)	UC	asynchronous, connectionless, explicitly acknowledged = selective reliability
8	COV/data point value (e.g. temperature)	continuous data tracking	announce ment (with aim soft-state publish-subscribe)	UC	asynchronous, connectionless, unacknowledged = selective reliability
9	deregister from COV at data point (optional, data-point-friendliness)	deregistration or un-subscription, continuous data tracking	announce ment (with aim soft-state publish-subscribe)	UC	asynchronous, connectionless, unacknowledged = selective reliability
10	service shutdown announcement (e.g. heating goes down for maintenance + dependent-service-friendliness)	deregistration or unsubscriptio n	announce ment	UC or MC	asynchronous, connectionless, unacknowledged = selective reliability

AC3 - Detection of anomalies of a central water heating					
#	Description	Purpose	Type	Multi.	Requirements
1	discover all service instances of the monitoring service that provide needed data points:	discovery	request/re sponse	UC or BC	depends on implementation-detail
2	ambient temperature sensor	subscription	request/re sponse	UC	asynchronous, acknowledged
3	feed flow water temperature sensor	publishing	publish	UC	asynchronous , acknowledged
4	return flow water temperature sensor	on demand data query	request/re sponse	UC	synchronous with timeout, acknowledged

AC3 - Detection of anomalies of a central water heating					
5	register for notifications from discovered data points	on demand data query	request/re sponse	UC or MC	asynchronous, acknowledged
6	publish notification	unsubscribe	announce ment	UC or MC	asynchronous, unacknowledged

AC4 - Detecting how heating or cooling behave in contrast to the scheduled temperature set points of a room					
#	Description	Purpose	Type	Multi.	Requirements
1	discover service instances that offer needed data points i.e. valve positions of the heating/cooling fan-coil, indoor temperature and temperature schedule	discovery	request/re sponse	UC or BC	depends on implementation-detail
2	subscribe for notification from needed data point	subscription	subscribe	UC	asynchronous , acknowledged
3	publish notification	publishing	publish	UC	asynchronous , acknowledged
4	read data point	on demand query of data point	request/re sponse	UC	synchronous, acknowledged
5	keep-alive check for notifications	subscription	request/re sponse	UC or MC	asynchronous, acknowledged
6	unregister from notifications	unsubscriptio n	announce ment	UC or MC	asynchronous, unacknowledged

AC5 - Adjustment of heating schedule taking into account user presence and feedback					
#	Description	Purpose	Type	Multi.	Requirements
1	discover service instances that offer needed data points i.e. room temperature, temperature schedule, room occupancy, booking information, user interactions etc.	discovery	request/re sponse	UC or BC	depends on implementation details
2	subscribe for notification from needed data points	subscription	publish/su bscribe	UC	asynchronous, acknowledged

AC5 - Adjustment of heating schedule taking into account user presence and feedback					
3	publish notification	publishing	publish	UC	asynchronous, acknowledged
4	write set point	set	request/re sponse	UC	asynchronous, acknowledged
5	read data point	on demand data query	request/re sponse	UC	asynchronous, acknowledged
6	keep-alive check for notifications	subscription	request/re sponse	UC or MC	asynchronous, acknowledged
7	unregister from notifications	unsubscriptio n	announce ment	UC or MC	asynchronous, unacknowledged

AC6 - Energy monitoring for tenants and lessors					
#	Description	Purpose	Type	Multi.	Requirements
1	Get devices offering the wanted data point. In the context of this AC it is used to find temperature sensors and metering devices for gas, water and electricity metering. Additionally it will be used to identify devices that authenticate the current user in a shared room.	Discovery (centralized)	request /respo nse	UC	connection-based, synchronous, reliable
2	Get devices offering the wanted data point. In the context of this AC it is used to find temperature sensors and metering devices for gas, water and electricity metering. Additionally it will be used to identify devices that authenticate the current user in a shared room.	Discovery (decentralized)	request /respo nse	BC	connection-less, synchronous, unreliable
3	Get the latest data from already discovered devices. This is used to get the latest data triggered either by a scheduler (time based) or at the beginning of a room rental (user enters room)	request data	request /respo nse	UC	connection-based, synchronous, reliable

AC6 - Energy monitoring for tenants and lessors					
4	(Un-)Subscribe for a change of value event. This is used by a monitoring service to provide a fine granular set of data related to the consumption of gas, water and electricity by a certain tenant or lessor.	(un-)subscribe	request /respo nse	UC	connection-based, synchronous, reliable
5	Send notice about changed/new values. This is used by meters to send changed values to the monitoring service	publish COV/next Value notice	publish	MC	connection-less, synchronous, unreliable

AC7 - Building Evacuation Based on Generated Alarm					
#	Description	Purpose	Type	Multi.	Requirements
1	Discover all service instances of the monitoring service that provide needed data points, e.g. smoke sensor monitoring, occupancy monitoring, alarm monitoring, wearable device monitoring.	discovery	request /respo nse	UC or BC	depends on implementation details
2	(Un)Register for notifications from discovered services instances	(un)subscripti on	request /respo nse	UC	synchronous with timeout, acknowledged
3	Read data point	on demand data query	request /respo nse	UC	synchronous with timeout, acknowledged
4	Keep-alive check for notifications	subscription	request /respo nse	UC	asynchronous, acknowledged
5	Write data point (e.g. fire alarm, location)	report incident	announ cement	GC	asynchronous, unacknowledged, connectionless
6	Publish notification (for evacuation service to notify output devices i.e. speakers, wearable devices etc.)	publishing	publish	UC	asynchronous, acknowledged

4.4.3 Identified Communication Patterns

4.4.3.1 CPAT-01: Request Response to GET Data From a Node

	Entities	Packets	Purpose	Multi.	Properties
CPAT-01	service instance, virtual data point	request: GET information; response: business information	on demand data query, discovery, name-address resolution	UC	· synchronous with timeout · connectionless · implicit ACK
Pattern used to query a data point (e.g. a sensor) by providing a source and destination address. In return to the query the "business information" is delivered, which can be a sensor value or, e.g. a discovered address. Examples: - node (A) wants to get a value from a sensor (B), the response is the value - node (A) queries a centralized registry (B) to receive information					

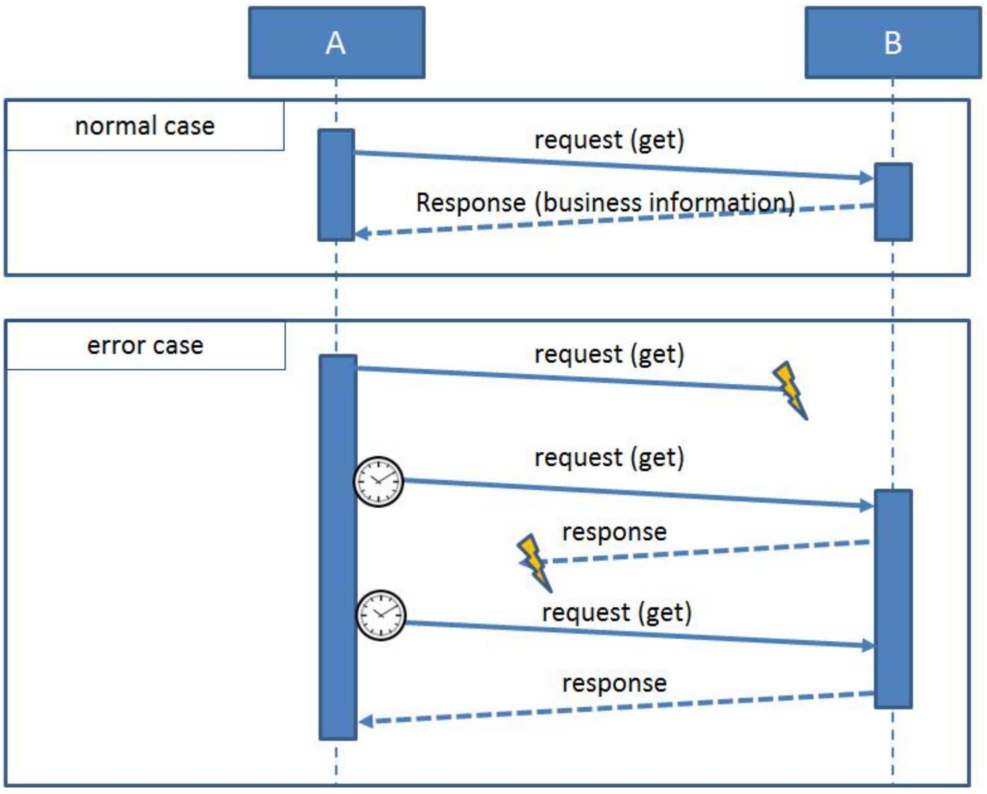
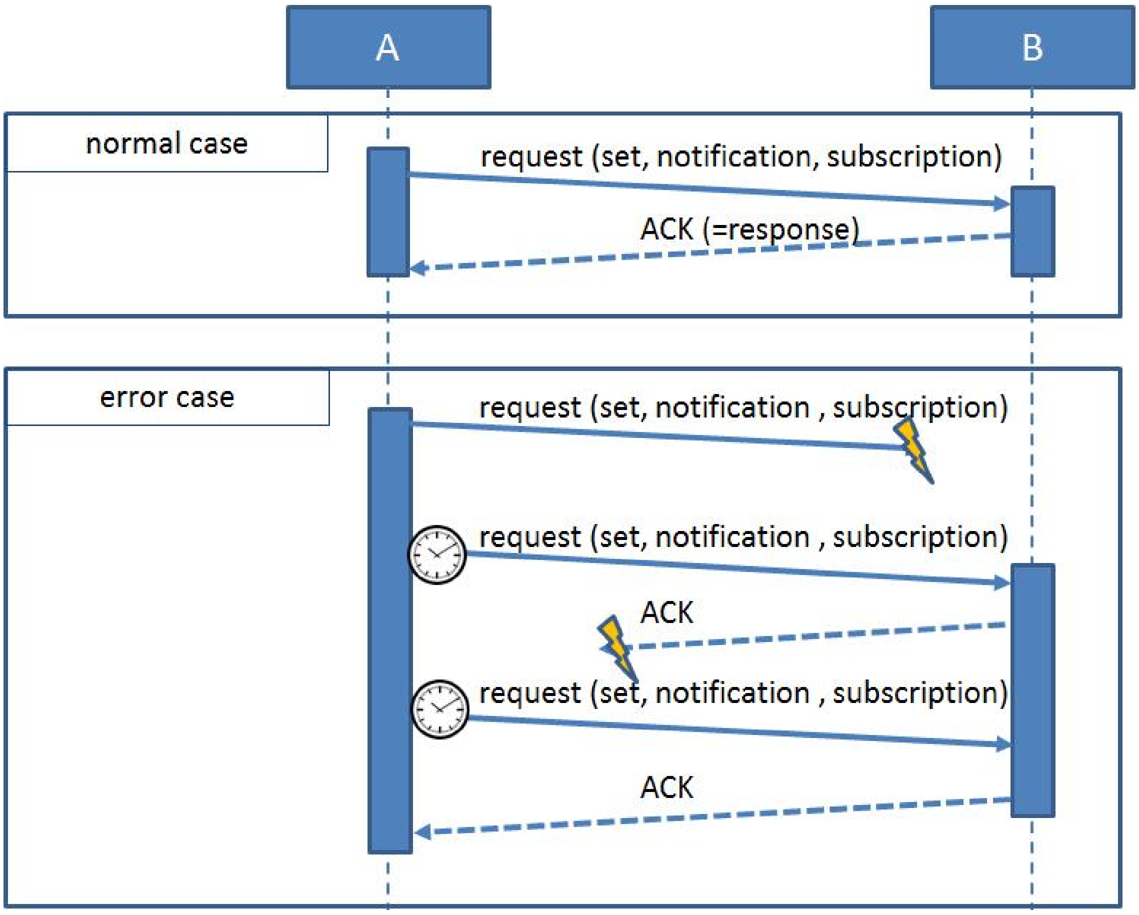


Figure 4-19: CPAT-01: Request Response to GET Data from a Node

4.4.3.2 CPAT-02: Request Response to SET Data to a Node

	Entities	Packets	Purpose	Multi.	Properties
CPAT-02	service instance, virtual data point	request: virtual data point; response: ACK	on demand data query, registration or subscription, continuous data tracking	UC	· synchronous with timeout · connectionless · explicit ACK
Pattern used to SET a value of a data point, e.g. an actuator value, a subscription or a notification. B needs to acknowledge all packets. Reliability needs to be implemented at the initiator. Examples: - node A sets a value at actuator B. An ACK is sent back as a response - node A subscribes to a service B, e.g. for a COV node B ACKs the subscription - A sends a notification to B, e.g. a deregistration and needs it ACKed					


A
B
normal case
request (set, notification, subscription)
ACK (=response)
error case
request (set, notification , subscription)
request (set, notification , subscription)
ACK
request (set, notification , subscription)
ACK
Figure 4-20: CPAT-02: Request Response to SET Data to a Node

4.4.3.3 CPAT-03: Asynchronous Request Response

	Entities	Packets	Purpose	Multi.	Properties
CPAT-03	service instance, virtual data point	request: virtual data point; response: ACK	registration or subscription, continuous data tracking	UC	· asynchronous · connectionless · explicit ACK
Pattern used to send a request to a node in a non-blocking way. Receiver needs to ACK all packets Examples: - A sends a notification to B, e.g. a keep-alive packet					

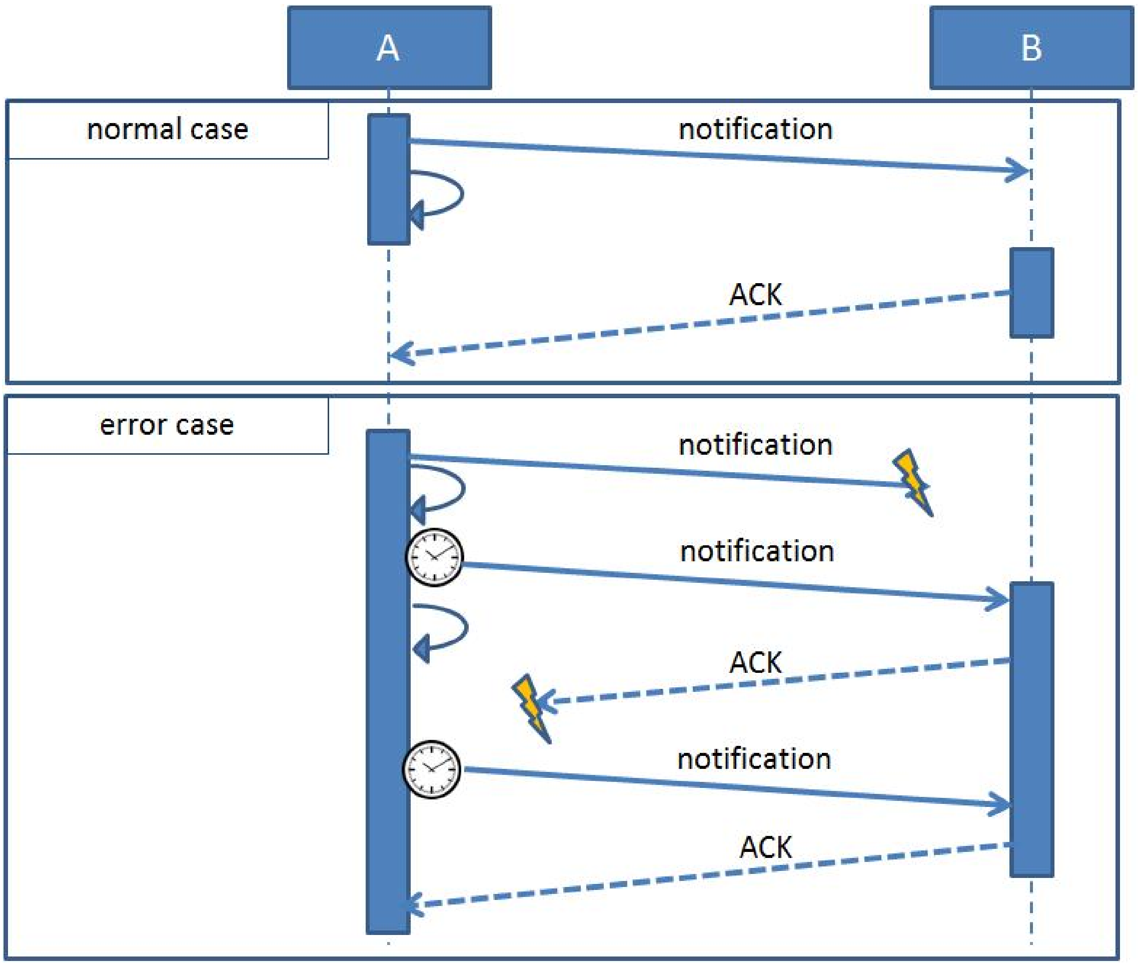


Figure 4-21: CPAT-03: Asynchronous Request Response

4.4.3.4 CPAT-04: Asynchronous Notification

	Entities	Packets	Purpose	Multi.	Properties
CPAT-04	service instance, virtual data point	Announcement: virtual data point Optional response: ACK	deregistration or unsubscription, continuous data tracking	UC	· asynchronous with timeout · connectionless · optional ACK
Non-blocking pattern used for sending a notification to a node. ACK is optional Examples: - Sensor (A) sends changed value to subscribed nodes (B) - Node (A) sends deregister to sensor (B)					

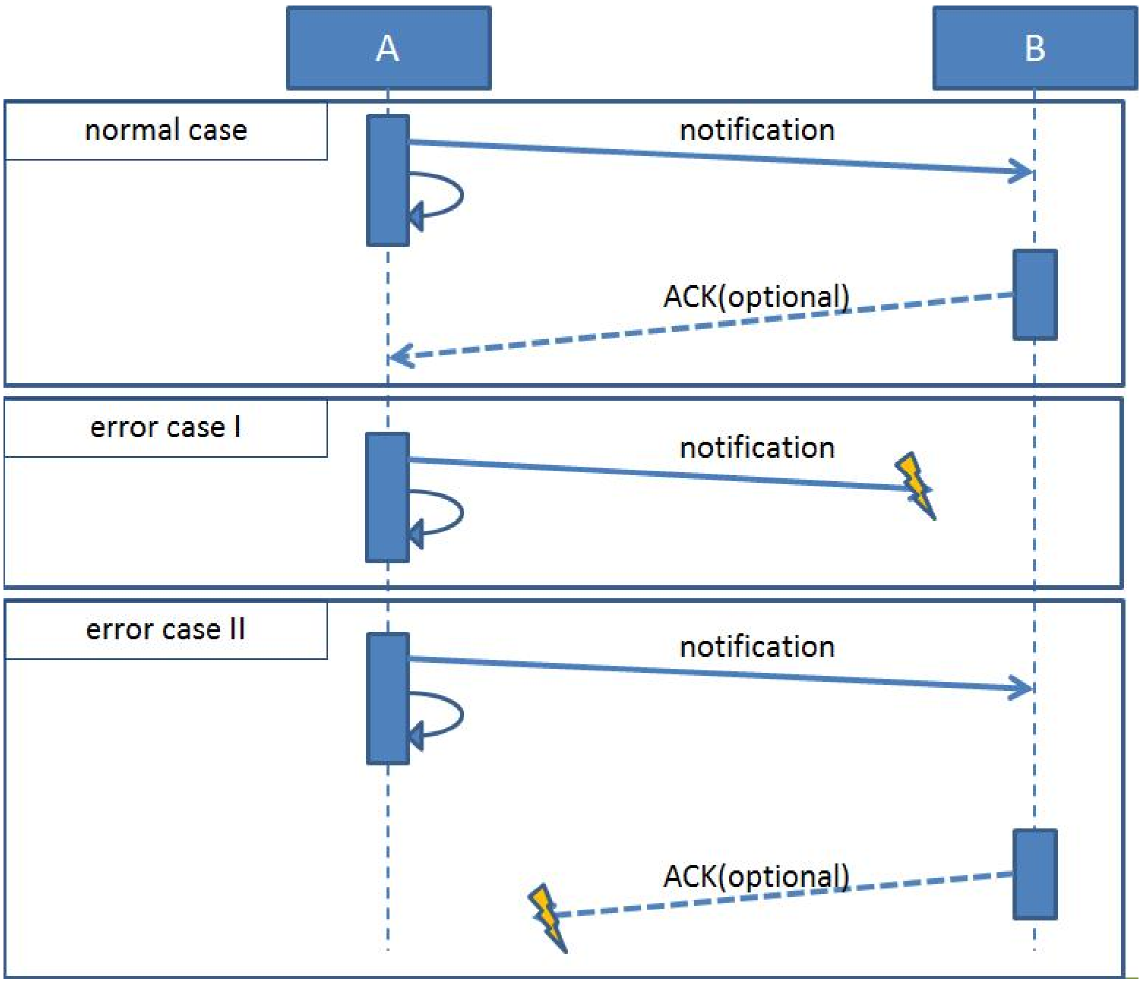


Figure 4-22: CPAT-04: Asynchronous Notification

4.4.3.5 CPAT-05: Asynchronous Message to a Group of Receivers

	Entities	Packets	Purpose	Multi.	Properties
CPAT-05	service instance, all service instances	message to: subscribers multicast group	deregistration or unsubscription, announcement, continuous data tracking	MC	· asynchronous · connectionless · explicit ACK
Pattern used to asynchronously send a messageto a group of receivers. It comes in two flavors (see figures below): 1) IP Layer MC and 2) AL Multicast Examples: - Gracefully deregister via Multicast/broadcast in order to notify a whole group about you leaving.					

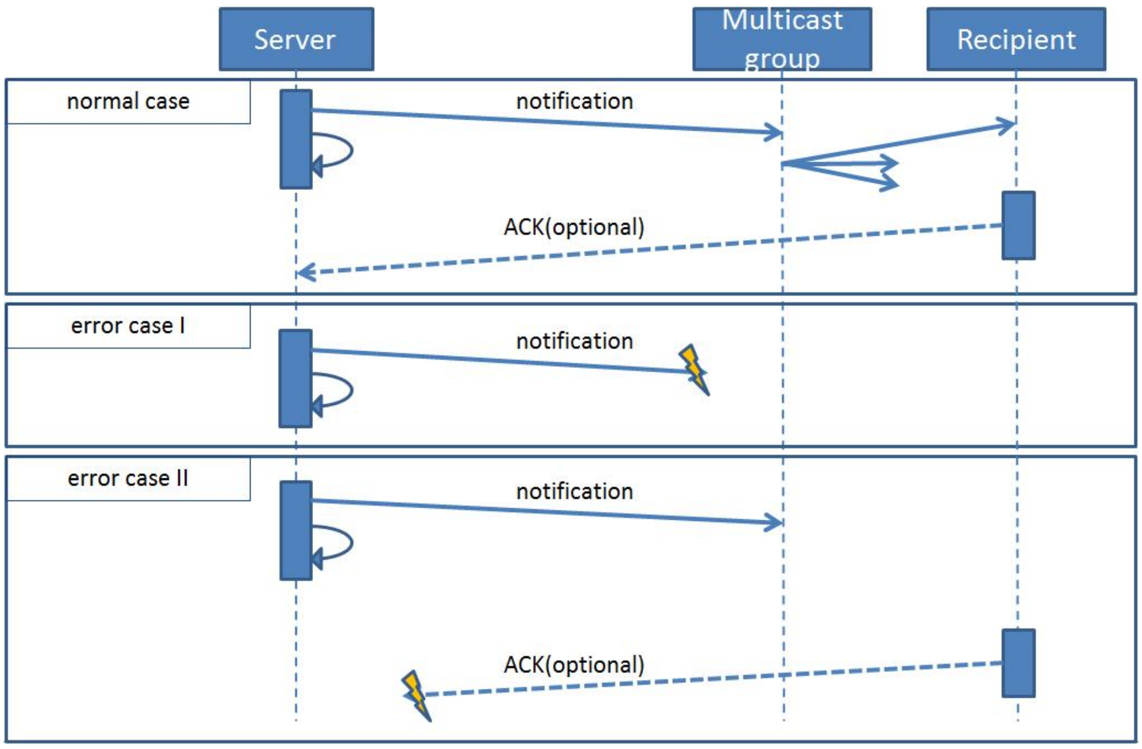


Figure 4-23: CPAT-05: Asynchronous Announcement to a Group of Receivers (1)

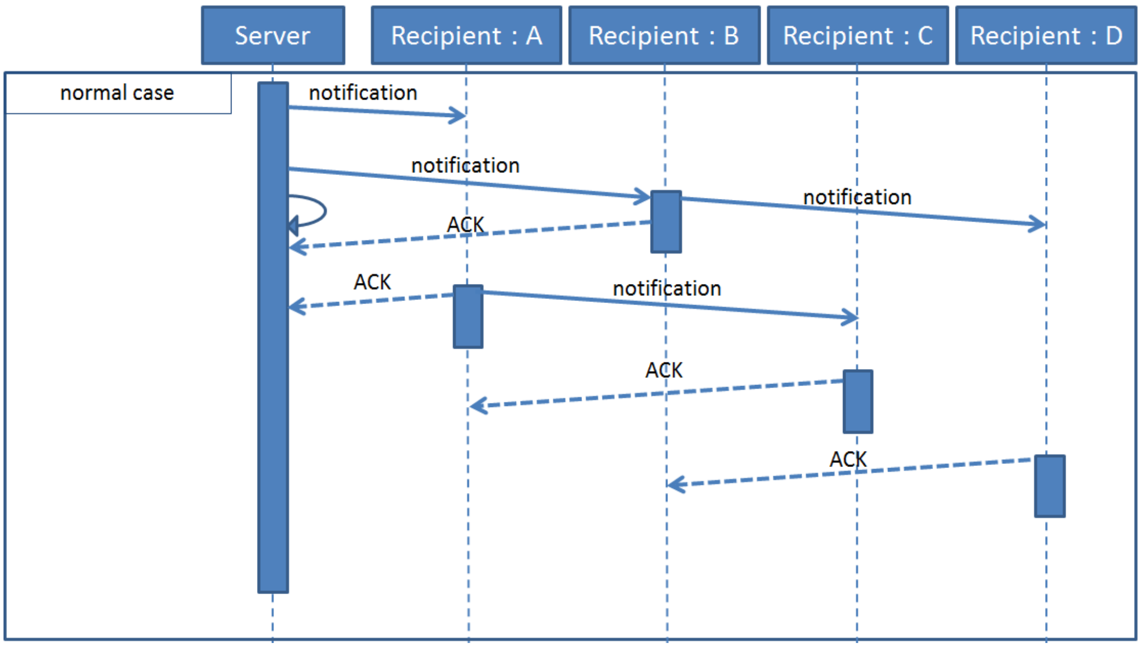


Figure 4-24: CPAT-05: Asynchronous Announcement to a Group of Receivers (2)

4.4.3.6 CPAT-06: Broadcast to all Nodes

	Entities	Packets	Purpose	Multi.	Properties
CPAT-06	service instance, all service instances	request to: virtual data point semantic filter expression; response: virtual data point	discovery	BC	· synchronous with timeout · connectionless · explicit ACK
Pattern used to synchronously send data to all other hosts Examples: - node A needs to discover other services					

4.4.4 Composition of Communication Patterns to Application Patterns

In this section application patterns are composed from the identified communication patterns as described in Section 4.5.3.

4.4.4.1 APAT-01: On Demand Data Point Query / Get Value Request

It is assumed that the service initiating the query depends on the result, i.e. the service waits until the value is successfully queried or the query finally and entirely fails.

This application pattern is realized by one step and covered by the following communication pattern.

Step No.	Used CPAT	Step Description	Comments and Rational
1	CPAT-01	This communication pattern is made for such applications and completely covers the application pattern.	This communication pattern directly matches all requirements of the application pattern. It operates synchronously with timeouts, which covers the aspect "the service waits until". Further, there is an implicit confirmation of success, because the reception of the value in response to the query indicates success. However, if there are serious problems, e.g. on the communication layer and the query times out then it is up to the service to decide how to proceed. For example, the service can decide that the query entirely failed and escalate the problem to a more abstract level, or start to retry the query (which may just delay the final failure of the query).

4.4.4.2 APAT-02: Reliably Set a Value at a Data Point / Set Value Request

It is assumed that the data point written controls some building automation actuator. Thus, it is essential to the service initiating the write that the write is reliably performed. This means, the service expects a confirmation that the write was executed and the service waits until either a confirmation is available or the write finally and entirely fails.

This application pattern is realized by one step and the covered by the following communication pattern.

Step No.	Used CPAT	Step Description	Comments and Rational
1	CPAT-02	This communication pattern is made for such applications and completely covers the application pattern.	This communication pattern directly matches all requirements of the application pattern. It operates synchronously with timeouts, which covers the aspect "the service waits until". Further, there is an explicit confirmation of success by means of an acknowledgement message. This additional message is required, because there is no necessity for the actuator to respond to the write request other than to explicitly indicate success. However, if there are serious (communication) problems (e.g., write request or acknowledgement message gets lost), the write request times out and it is up to the service to decide how to proceed. Possibilities to proceed are already indicated in the context of APAT-01.

4.4.4.3 APAT-03: Publish/Subscribe for Change-of-Value Notifications

A service wants to monitor the temperature of, e.g. a room and timely react to changes. To avoid high network loads, the service subscribes at the respective data point for so called change-of-value (COV) notifications. As the service depends on the successful subscription, the subscription must be performed reliably. Because the subscription is realized in a soft-state manner, i.e. it automatically times out unless it is explicitly kept alive by the service, the service periodically sends *"keep-alives"* as a background task.

This application pattern is composed of the following steps and communication patterns.

Step No.	Used CPAT	Step Description	Comments and Rational
1	CPAT-02	This communication pattern covers the subscription for COV-notifications.	Because the service depends on the subscription and as there is no necessity for the respective data point to respond to the subscription request other than to confirm success, a synchronous communication pattern with timeouts and explicit acknowledgement is required. In essence, CPAT-02 exactly fulfils these core requirements.
2	CPAT-03	This communication pattern covers the keep-alive of COV-subscriptions.	After the subscription is established, it must be kept alive to indicate the data point that the service is still operational. The keep-alive of the subscription can be performed as a background task and does not depend on very timely responses. However, the service should be enabled to identify when the data point is no longer operational, which mandates for explicit confirmation that a keep-alive message was received. Therefore, an asynchronous communication pattern with explicit acknowledgements is appropriate; CPAT-03 perfectly fulfils the stated needs.
3	CPAT-04	This communication pattern covers the COV-notification of the service.	Once the data point detects a changed value, it needs to notify all subscribers about this COV. Dependent on the service requirements, the delivery can be unreliable or reliable. Irrespective of reliability, the notification of the subscribers can be performed concurrently and asynchronously. The CPAT-04 fulfils these requirements as it is an asynchronous communication pattern that selectively supports reliability. This means, acknowledgements to be sent by the subscribed service can be optionally required (turned on). If reliability is activated then the retry-limits for the COV-notification must have been agreed on at subscription point in time.

Step No.	Used CPAT	Step Description	Comments and Rational
4	CPAT-04	This communication pattern covers the friendly unsubscription from COV-notifications.	As introduced above, the subscriptions are soft-state, i.e. that the data point automatically unsubscribes any service from which no keep-alive was recently received. However, the service behaves very nicely and explicitly unsubscribes from COV-notifications, which gives the data point the opportunity to promptly free resources. As the service neither depends on the success of the unsubscription nor on any timely unsubscription, it can be performed asynchronously and unreliably. COV-notifications that are already in transit or transmitted due to failed unsubscription can be identified and silently ignored by the service (if still operational at all). In case the explicit unsubscription fails, a fallback to the implicit soft-state-based unsubscription is sufficient. The CPAT-04 with acknowledgements disabled matches the described requirements.

5 Perspectives

Perspectives serve to bring considerations into the architecture that have implications across different, but not necessarily all, architectural views. The insights gained in this process can be used to refine and modify the views. Perspectives can act as a store of knowledge, guide and memory aid. [3]

5.1 Security Perspective

The security perspective is concerned with the security and privacy implications in the BaaS reference architecture. In the following, we will consider information and communications security. In this context, privacy refers to the protection of information about natural persons only.

We do not consider operational security. Additionally, we will assume the physical security of the BaaS components. Communication over unsecured networks will be examined.

For implementations of this reference architecture, varying preconditions could result in diverging requirements and security ambitions.

5.1.1 Threat Model

There are various reasons for introducing security mechanisms into the reference architecture.

As the BaaS system will operate on data capable of identifying individuals, a possibility to enforce privacy mechanisms is required. Security systems can also help in enforcing safety properties, e.g., by providing appropriate access authorization on sensitive interfaces.

On the business side, BaaS operators may want to protect data access as a business model. The protection of investment, e.g., regulating access to interfaces, may also be considered.

As we allow communication over unsecured networks, active and passive attacks on network traffic need to be mitigated. Also, the possibility of unauthorized access to network facing interfaces needs to be addressed.

5.1.2 Concept

The access to the information and functionality of a building automation system system has to be authorized. This authorization happens at the access to the exposed value of data points – those are the interface of the service offering of data points.

services are the active components in BaaS. Therefore, we need to authenticate services in order to make an authorization decision when a service accesses the data point value of another service.

To achieve this access control, data point values and services need to be augmented with permission information. When a service accesses a data point, the permissions of the service will be checked against the access control rule defined for this data point value. Since data points themselves are offered by services, a service building block, the security adapter, is used perform this permission check. Due to the security adapter, services are not required to implement authorization logic themselves.

The authorization mechanism should support the delegation of authorization. This allows a service B to request a resource on behalf of another service A or a user. The authorization decision will then be taken according to the permissions of service A or the user respectively. The delegation pattern allows cascading the access to data and its authorization. Cascading is used in scenarios where the data access traverses multiple abstraction layers.

The authorization permissions should be designed early in the lifecycle (c.f. Section 4.1.) to facilitate a security-aware design process. In order to achieve reusability and simplification, the permissions should be identifiable and stored in a repository.

5.1.3 Security During the Lifecycle

We will show what steps have to be taken during the different phases in the lifecycle in order to define and achieve security goals.

5.1.3.1 Development Phase

During the development of a data point, default access rights should be set on the data point values in order to establish access control. This allows providing a secure default access policy, while reducing burden in the later phases. This default may be modified during the engineering phase. The access permissions are referenced by data point descriptions and contain the following information:

- Unique ID
  The ID is used to identify and reuse permissions in different data point descriptions.
- Read-or-write
  Specifies if this permission controls read or write access
- Description text for human understanding
  This description explains the intended use of the permission.
- [Reference to data point values]
  The permission may reference the data point values it is used for in order to simplify searching for and reusing existing permissions.
- [Reference to services]
  The permission may reference the services it is used for in order to simplify searching for and reuse existing permissions.

A repository of these permissions allows reusing them and simplifying the permissions structure.

Additionally, services can be grouped by their security and privacy properties. The description of this behavior in machine readable form allows reusing that information to create segregated information domains during the engineering of authorization permissions.

5.1.3.2 Engineering Phase

In the engineering phase the configuration of permissions takes place. The exact access rights have to be determined and configured. The default values may be modified to specify appropriate access control rights.

For example, reusing the service privacy properties of the development phase allows building collections of services, which share data only with other certified services. This way, service groups of privacy preserving services can be created and privacy goals can be achieved.

A security adapter provides functionalities such as authorization for use by services, in order to not to require each service to implement access control correctly. This module will also to be able to handle revocation of access rights. Additionally, the security adaptor includes a security protocol facility providing encryption and message authentication capability for use in the operation phase.

5.1.3.3 Commissioning Phase

During commissioning, the security adapters are provisioned with credentials such as certificates. These credentials may be used during operation for authentication, authorization and encryption.

5.1.3.4 Operation Phase

In the operation phase, several components need to fulfill security related functionality.

The security adapter needs to be available to service instances in order to perform access control of data points. Revoking access rights is possible.

Service instances are authenticated in order to allow secure access control by the security adapter. Using these mechanisms, access control can be enforced at virtual data points.

End-to-end secure communication between service instances, via the interface offered by data points, is available. This allows services to be run on arbitrary BaaS devices without further security considerations. Secure communication is achieved by authenticated encryption with replay protection. Authenticated encryption is the secure combination of message authentication and encryption. It provides integrity protection, authentication and confidentiality.

Any security properties are bootstrapped by the credentials provisioned during commissioning. Access rights can be revoked by revoking the credentials of a service. Communication with legacy devices may be unsecured.

Service instances provide audit and logging interfaces.

5.2 Dependability Perspective

Due to the fact that different devices and components are spread across buildings, it is obvious to think of building automation systems (BAS) as distributed systems. In context of the BaaS project devices and components are assumed to be represented by services.

Due to services controlling the environment where people live respectively work, it is necessary to impose requirements on the behavior and quality of the underlying system – the BaaS platform and the services run in its context. For the reason that applying a service system in the field of building automation leads to high complexity and dependency between various services combined to control respective parts of buildings, dependability is a required aspect which needs to be considered within the BaaS project. Without any constraints and requirements on the involved services, it is not possible to assure certain qualities of the complex service system.

The dependability can be summarized by the well known abbreviation RAMS defined in the norm EN 50126 [16]. RAMS stands for reliability, availability, maintainability and safety. Reliability is generally a property of a system, which describes how reliable a given functionality is provided by a system in a particular interval of time free of failures. [17] The availability of a technical system is in contrast to the reliability a calculable measure describing the proportion of time the system was available to the total observed amount of time [18], [19]. The maintainability describes the required effort to solve a systems functional issue and can be quantified through the time needed to fix a problem and how difficult/complex the troubleshooting is. In the context of BaaS we want to focus on reliability, availability and maintainability. Safety is an important property of distributed systems too, but in the field of building automation safety critical parts in buildings (e.g. fire alarm and evacuation systems) are usually regulated and controlled by legal authorities already.

The dependability in the field of building automation can approximately be considered and supported on the abstraction levels hardware (computing nodes and network), platform services and building automation services. The platform and the building automation services need the physical hardware to be executed and physical networks to communicate in the distributed environment. The building automation services require a platform to be managed and run inside the context of building automation, e.g. to access information about available data points and other services. In the BaaS project we consider dependability in the context of platform services and building automation services called BaaS platform services and BaaS services.

The dependability of BaaS services can only be ensured when the BaaS platform services run dependably. Therefore we consider both levels of abstraction in the investigations of the BaaS project to enable dependable building automation systems.

5.2.1 Concept

As already stated in the previous section we want to focus on the dependability of BaaS platform services and the BaaS services. We describe the different concepts for these parts in the following. The following subsections describe an approach to accomplish dependability in regard to reliability and availability, whereas the aspect of maintainability is described in the sections covering the functional view. The defined interfaces and services help to detect anomalies in the operation of the building automation system and enable the reconfiguration and adaptation of services in order to ensure a high level of dependability.

5.2.1.1 BaaS Services

BaaS services are executed in the context of the BaaS platform providing among others the technical management system. The BaaS services need to provide interfaces to the management system for the following operations in order to be properly managed:

- Monitoring/Alerting
- Configuration/Repair
- Diagnosis
- Self-test

These interfaces have to be implemented and considered along the BaaS lifecycle. Therefore the BaaS platform should for instance provide skeletons for BaaS services including the interface definition. The utilization is described more in detail in the functional and lifecycle view.

5.2.1.2 BaaS Platform Services

The BaaS platform services provide basic utilities and functions in order to support the operation of BaaS services. Generally, as BaaS services, BaaS platform services have to provide interfaces for the management operations monitoring, configuration, repair, diagnosis and self-test. In principle, BaaS platform services should be managed as well (by using the mentioned interfaces) in order to apply another level of fault-tolerance. Otherwise, in case of a failure of a BaaS service and a failure of platform service at the same time, the platform service cannot react/avoid damage as result of the failure of the BaaS service. We are able to provide a much more dependable system by being able to manage platform services too.

5.2.1.3 Management Services

The management services are BaaS platform services and form the management system, which monitors, configures, repairs and distributes the components of a BaaS system. The technical management system manages BaaS services and BaaS platform services amongst other managed objects like BaaS container and BaaS devices.

Thus, important management services are:

- Monitoring services,
- Configuration services,
- Deployment and Distribution services,
- Rule Engine services.

We especially want to use the technical management system (see Subsection 3.2.3) to realize and enable the development, engineering and operation of dependable BaaS services.

System changes are observed and can be detected by monitoring services covering the lifecycle of the managed objects in order to (re-)configure, (re-)deploy, repair, analyze and test these. These system adaptations are enforced by management rules which are executed and interpreted by a rule engine service after evaluation of defined conditions. The rule engine service is provided by the technical management system. The execution of particular rules is triggered by certain events which consist of the observed system changes. Therefore management rules are defined by a set of corresponding (Event, Condition, Action)-triples.

5.3 Technical Management Perspective

Adapting its behavior to changing conditions and requirements as well as reacting on system intrusions and faults are targeted key qualities of a BaaS system. From the dependability and security perspective, it should be equipped with corresponding self-managing capabilities, which include configuring, reconfiguring, tuning, protecting, and recovering itself constantly and at the same time keeping the complexity of these tasks hidden from users and administrators. Thus, a comprehensive automated technical management is required.

Essentially, the management performs an intelligent control loop: automated methods collect the needed details from the system; these details are analyzed in order to determine if something needs to be changed; a plan and/or sequence of actions which specify the required changes is worked out; and finally the plan is executed. A common knowledge in form of management information is a basis for the whole management process.

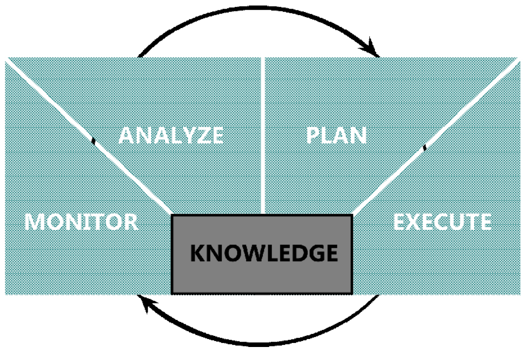


Figure 5-1: Management Control Loop

Thus, in accordance with IBM autonomic computing model (Figure 5-1) [12], the architecture dissects the management loop into four functional parts sharing common knowledge:

The monitor function provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from the monitored resources.

The analyze function provides the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow discovering the IT environment and help predict future situations.

The plan function provides the mechanisms that construct the actions needed to achieve goals and objectives.

The execute function provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

While providing the management functions mentioned above, we stick to the *rule-driven management paradigm* depicted in Figure 5-2. According to it, the management logic, carrying out the management process in the operation phase, is established by a set of exchangeable *management rules*. These rules contain the management knowledge and are derived from the system model for a particular BaaS system in accordance with its constraints and requirement specifications during the engineering phase. A *rule storage* is used for storing the derived rules, which are deployed during the commissioning phase.

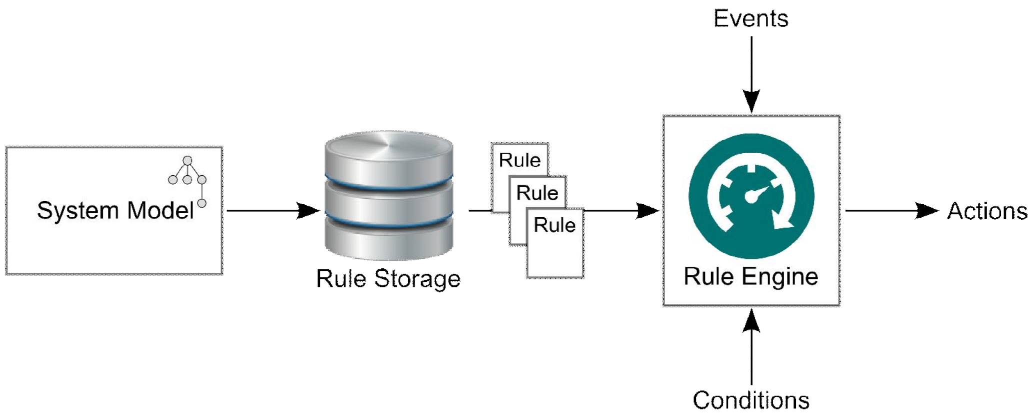


Figure 5-2: ECA Rules Paradigm

A management rule is structured as an *event-condition-action* (ECA) construct, which states the desired management behavior. On a certain *event*, in case a stated *condition* is fulfilled, a corresponding *action* should be executed. *Rule engine* is responsible for the evaluation and

execution of the rules. It receives events generated due to condition changes, evaluates related rules and triggers respective actions.
Since maintaining dependability of the management system is essential, self-managing capability is one of the key features of the BaaS system. For this purpose, in order to make the management process monitorable and controllable, a set of specific management rules is introduced. These rules are supposed to fire when management actions fail.
The following subsections describe the design and implementation details of these rule-driven management functions within the BaaS architecture.

5.3.1 Management Infrastructure

The BaaS management infrastructure is depicted in Figure 5-3. It is to distinguish between the *managing* and the *managed system*. The managing system hosts managing processes, *managers*, while the managed system is resided by *management agents*. A manager is the part of the management process that takes decisions based on collected management information. The manager monitors and configures the managed system by communicating with the management agent residing there. Managers can be arranged in a hierarchical structure when required by the system structure.

A management agent is an entity that collects management information and makes this information available to the manager. It executes received management commands and sends notifications. Managers and agents communicate via protocols at the application layer. The manager-agent relationship adheres to the well-known client-server paradigm.

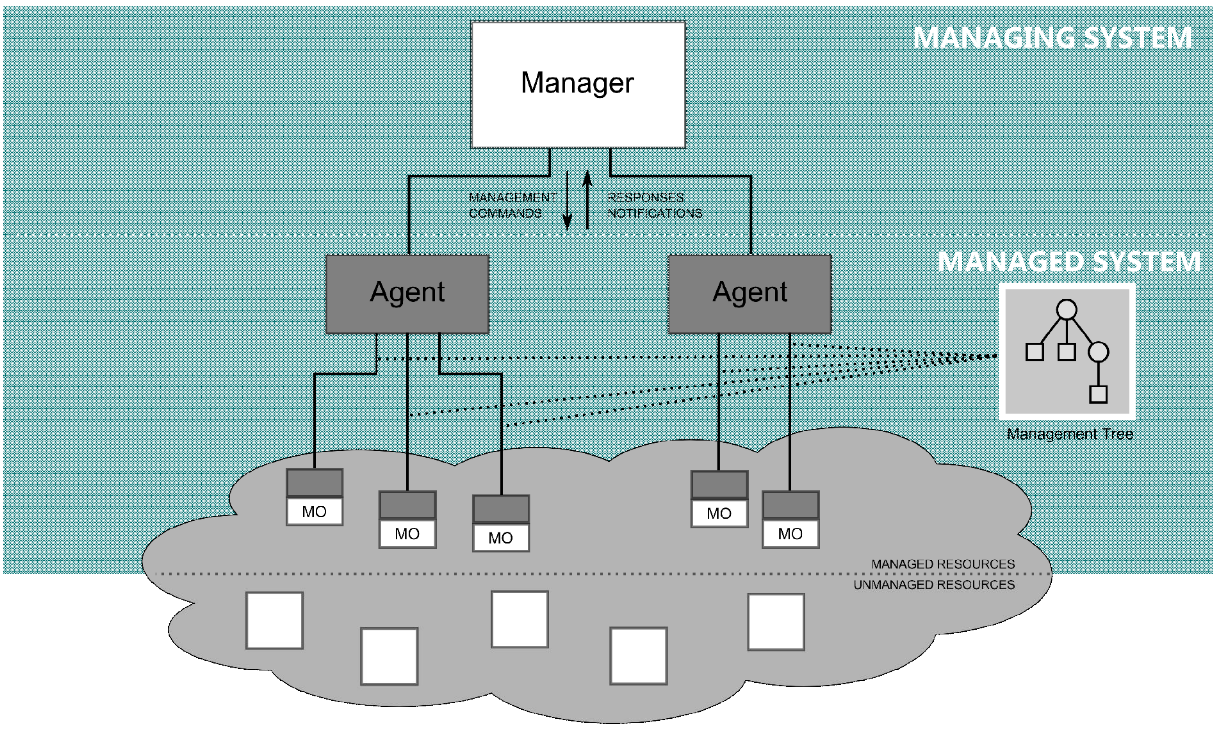


Figure 5-3: Management Infrastructure

From the management point of view the BaaS system components can be classified into *managed* and *unmanaged resources*.

Unmanaged resources do not directly participate in the BaaS management process, but can provide valuable information to other managed resources.

Managed resources are the main subject of interest. In practice, they are BaaS nodes, BaaS containers and BaaS services. Introducing agents means in technical terms that all managed resources are equipped with a dedicated software application that implements the management agent functions. A managed resource is classified by a set of predefined types or classes. This ensures that the management is scalable and allows the seamlessly integration of new managed resources in the management process.

The actual resources to be managed are represented by *managed objects* (MOs). Each of them is identified by an Object Identifier (OID) that is unique and constant throughout its lifetime. A managed object comprises a set of *management variables* that can be further subdivided into *status variables* and *configuration variables*. Status variables represent the management relevant state of a managed resource and can only be read from the management system. Configuration variables offer the capability of (re-)configuration and are solely set by the management system. In fact, querying and modifying the management objects properties is only available in form of reading and writing corresponding status and configuration variables. What operations on managed management objects will be executed is predefined by management rules, but still depends on the current system state.

The BaaS system may contain hundreds of managed objects. In order to make management practically feasible, the concept of a *management domain* is introduced. Management domains provide the means of partitioning management responsibility by grouping objects accordingly to common characteristics. A managed object can belong to several domains whereas all members of a management domain are managed with respect to the domain-specific management rules.

All BaaS managed objects are arranged in a distributed hierarchical tree structure, a *management tree* that offers a virtual data access structure of management data.

5.3.2 Management Tree

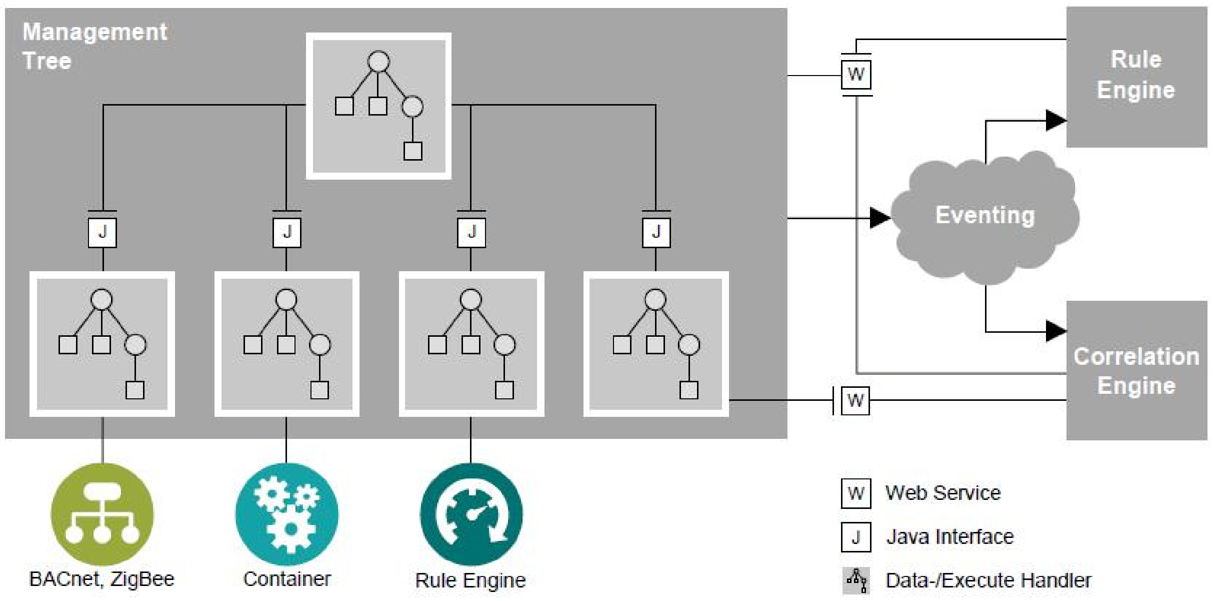


Figure 5-4: Management Tree

A distributed management tree [13] covers the protocol-specific parameters for data acquisition through specific handler implementations which are realized in the form of self-contained software components (Figure 5-4). They are dynamically provided at runtime respecting the requirements imposed on the monitoring and configuration system. It means, only those parameters are accessible through the management system that actually have data consumers. When management data need no longer to be accessible, the corresponding data handlers are disposed in order to improve system performance.

The management tree forms a virtual data access structure that does not store any kind of management data, but rather offers a hierarchical view on the data provided by different management agents. The purpose of a *handler implementation* is to map management data to an adequate hierarchical object model abstracting from access operations and event notification. Data that should be visible within the management space must be provided by management variables, in the form of status and configuration variables. The configuration variables are written by the manager in order to trigger the component to adapt its behavioral state. Changes in state are vice versa reflected by modifying the appropriate status variables by the component.

The separation between status and configuration accounts for the temporal offset between status change triggering and the actual execution. Even when the management system triggers the state change, it is possible that the transition fails or is not executed within the required time constraint leading to an inconsistency between actual and desired state. The variable separation allows the system to control and reconfigure itself by monitoring those actions it has initiated.

Just like any tree data structure, the management tree is a hierarchical structure consisting of *nodes* and *vertices*. We distinguish between *interior nodes* and *leaf nodes*. Interior nodes can have children nodes (i.e. interior and leaf nodes), whereas leaf nodes hold primitive values. Base value types are: Boolean, Short, Integer, Long, Float, Double, String, Binary and Object. Nodes can have an Access Control List (ACL), associating operations allowed on those nodes with a particular principal represented as a String value. Furthermore, nodes can hold meta data describing the actual nodes and their siblings. Leaf nodes may have default values specified in their meta data. Allowed access operations (Get, Add, Replace, Delete and Execute) on nodes are defined by means of meta nodes. Each node of the tree is uniquely identified by an absolute URI starting at the *root node* of the tree.

Furthermore, a node has a number of properties. They can always be read; some of them may also be set at runtime. The properties of interior nodes and leaf nodes are:

Name The node's name, which must be unique among its siblings. A node can be renamed at runtime depending on the capabilities of the underlying handler implementation.

Title A human readable title of the node, which is distinct from the node's name. This property is optional depending on the implementation that handles the node.

ACL The Access Control List for this node and its descendant nodes.

Version The version number starting at 0, incremented after every modification (for both a leaf and an interior node). Changes to the value or any of the properties (including ACLs), or adding/deleting nodes, are considered as changes. The value is read-only. In certain cases, the underlying data structure does not support change notifications or makes it difficult to support versions, therefore this property is optional depending on the node's implementation.

Timestamp The time of the last change in version. The value is read-only. This property is optional depending on the node's implementation.

Data Type The data type of the data that can be held by the corresponding node. Only a leaf node can be typed.

Mime Type The mime type of the data contained in the leaf node. This property is only supported by leaf nodes.

Value The value contained in the leaf node. Only leaf nodes can hold values.

Schema The name of the schema defining the subtree structure with this node as root node.

Thus, the whole management process can be reflected in operations on the management tree, which are limited to node creations, node removals and property assignments. A management node in the form of an interior or leaf node is an abstraction of the actual management information. The management tree incorporates brokerage capabilities and offers a consistent and uniform view of the underlying management data. Each node must be provided by a data handler implementation. Thereby two different handler implementations can be distinguished. The first handler implementation type is the basic one; the management tree itself is equipped with. It restricts the tree structure according to a predefined schema. Schemas are declarative and can be added or removed at runtime. A schema specifies and determines the subsequent tree structure. A node may be typed with a schema at creation time, whereas this particular node represents the root node of this subtree. If a node is untyped, it is automatically a part of the already declared schema valid at this part of the tree. A node can only be added, removed or changed if the underlying schema allows the corresponding modification action. A node can only be added to the tree if it can be associated with a matching schema node otherwise the node cannot be part of the tree and node creation fails. This point in the management tree represents the transition between two schemas, so that the new node must be typed with a new schema valid at this position. The tree data is only stored in memory and is not persistent.

The other handler type is a protocol-specific handler implementation of an existing management service or agent. It adapts the management data and offers a hierarchical view of these data to the consumer. This handler can be added dynamically to the management tree's access mechanism and homogenizes the monitoring and configuration data in order to be in line with the handler's pre-defined object model.

Interacting with the management tree requires a valid session, which allows atomic or transactional data access as requested. The client specifies at the very beginning what kind of session is needed and chooses one of the following three session types:

Shared Session Any number of read-only sessions can run concurrently, but ongoing read-only sessions must block the creation of an exclusive session on an overlapping subtree.

Exclusive Session Two or more exclusive sessions cannot access the same part of the tree simultaneously. An exclusive session must acquire a write lock on the subtree, which blocks the creation of other sessions that want to operate on an overlapping subtree.

Transactional Session A transactional session is the same as an exclusive session, except that the session can be rolled back at any moment, undoing all changes made so far since the last transaction point defined by a previous commit call. All participants must accept the result: rollback or commit.

5.3.3 Overall View

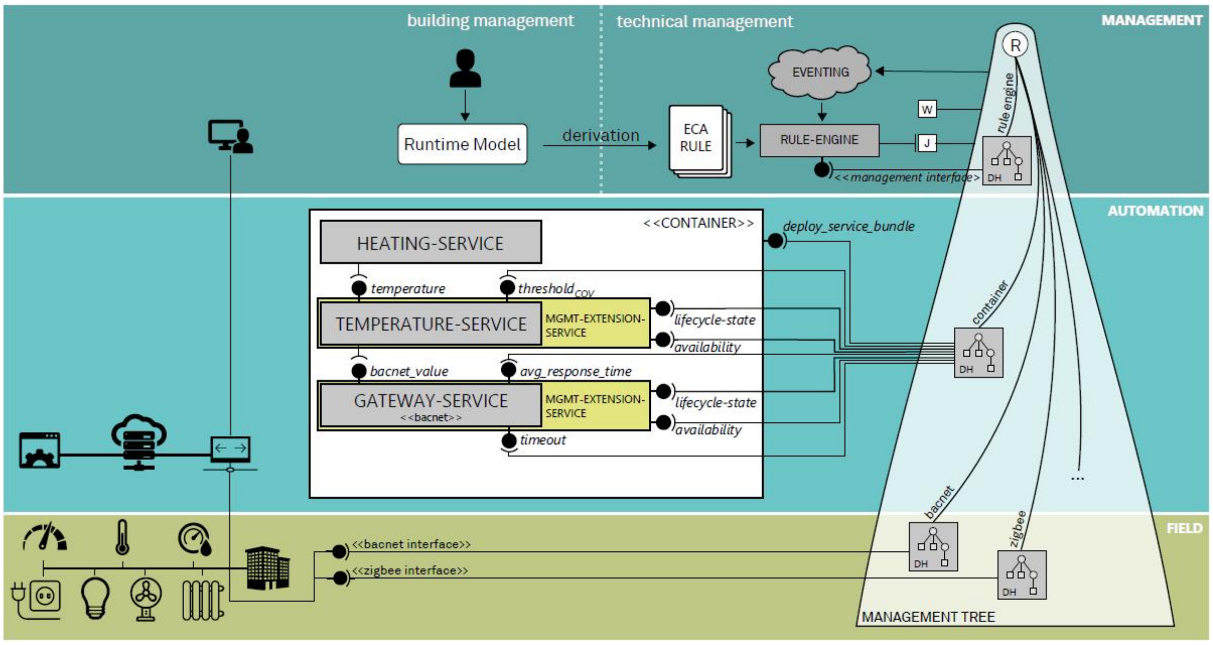


Figure 5-5: Technical Management Overall View

Figure 5-5 depicts how the technical management system is embedded into the overall BaaS architecture [14]. The complete building automation system basically consists of the physical infrastructure (sketched on the left hand side), a building automation service hierarchy deployed on containers and the technical management system. The various parts are spread over the different layers of the automation hierarchy. The access facade forms a homogenous view on the resource landscape composed of entities that provide dedicated functions for the BAS spanning from low-level sensors and actuators to middleware and finally to the technical management system. Thus, the technical management is extensive and global, covering all the BaaS layers: management, automation and field.

The core of the system is given by the management tree (sketched on the right hand side) and its data handlers (e.g. for the rule engine, containers, BACnet and Zigbee) enabling the homogeneous configuration and monitoring of all relevant managed objects through corresponding management variables (e.g. lifecycle-state, avg_response_time, timeout). The bacnet- and zigbee-data handlers are responsible for the protocol dependent management on the field level. It automatically detects outtakes, defects and other anomalies arising on the lowest level of the BAS hierarchy. The container-data handler observes and injects all changes related to the building automation services. Here comes the service lifecycle management into play caring for the proper operation of all BAS services including for instance the dynamic binding of services, service deployment, configuration, and monitoring. It does not only observe the BAS services themselves, but also the environment like mgmt-extensionservices or the underlying containers, too. The last data handler contained in the example is the rule engine-data handler as a prominent representative for data handlers covering the technical management system itself. The rule engine-data handler monitors and configures the execution of management rules in order to provide a reliable management process. Specific branches within the management tree are created for this purpose, which branches contain status variables describing the actions and parameters executed on the related elements. Start time and end time allow monitoring that rule execution succeeds within specific time constraints. This self-managing capability is a key feature in our implementation providing for self-managing ability.

In order to be compliant with the lifecycle of building automation systems, we utilize the runtime model of the BAS to automatically derive the management rules and artifacts for the configuration of the technical management system. Furthermore, the patterns for the derivation of the rules and artifacts are developed at the development phase, which in turn are than applied on the transformation of the runtime model in the engineering phase. The rules and artifacts configure the management system during the commissioning phase when all BAS services are set up too.

Taking our example into account, the automated management works as follows: the management tree observes the lifecycle state of the temperature service by means of the mgmt-exentsionservice. In case any change of the lifecycle state is recognized by the management tree, corresponding events are generated in order to notify all components that are subscribed to lifecycle state changes of the temperature service. Let us suppose, the rule engine is provided with a management rule to react on the changes of this service. As soon as the state changes, the rule engine receives the corresponding event and evaluates the respective conditions defined by the management rule. For example, if the state has changed (event) to uninstalled (condition), the rule engine triggers the redeployment of the temperature service by calling (action) the deploy service bundle-interface through the management tree. Another example is the recognition of a new heating BACnet component on the field level, which in turn results in the deployment of a corresponding heating service.

6 Summary

The document at hand describes the second and final iteration of the BaaS reference architecture. This reference architecture defines an abstract set of mechanisms and patterns to provide guidance for the development of concrete BaaS architectures, taking into account the architectural requirements of the BaaS system.

The BaaS project [1] is targeted to establish a generic service platform for commercial buildings that integrates traditional building automation and management systems with ICT infrastructures. This platform supports the development and deployment of novel valued added services and applications that take advantage of an integrated model of novel and legacy building systems and the data provided and consumed by them.

The BaaS reference architecture also addresses BaaS technical objectives:

- A structured approach for the generation and deployment of value-added building services is presented in the Lifecycle View. The view clarifies the artifacts (models, software,...) created in each phase of the lifecycle of a BaaS system and how those artifacts depend on each other to generate value added services in a model based fashion. This reduces the effort needed to create and maintain building automation systems.
- The Information View defines a BaaS data model which includes additional semantic information to simplify the engineering of value added services and applications for the BaaS system and the integration of legacy systems.
- The Functional View defines the building blocks needed in each phase of the lifecycle to define and generate a BaaS system based on model-based mechanisms. The functional view captures the interactions of the functional building blocks as triggered by the actors during the lifecycle. The resulting system allows the analysis, aggregation and transformation of data in a service-oriented way, where the engineering is supported by the added semantic information. This view also provides building blocks for the integration of existing and novel sources of information to create a "building information sphere".
- The Behavioral View focuses on the interaction and communication patterns of data exchanged between BaaS services. Those patterns derived from application cases are needed to model the distributed communication in a typical building automation system.
- The Dependability, and Security Perspectives deals with the reliability, availability as well as with the security of the BaaS system. These qualities are needed as base in building automation systems and have to be supported by BaaS.
- The Technical Management Management Perspective describes mechanisms to manage and maintain the availability and reliability of the BaaS infrastructure.

6.1 Modifications in the Second Iteration

The second iteration of the BaaS reference architecture uses feedback from the implementation of the BaaS platform for a major overhaul of the Information View. We have introduced the concept of BaaS features to increase the reusability of the models and have refined the Building Automation Function as semantic description of datapoints as well as the modeling of BaaS services. Further, we have consolidated the information provided on the Technical Management as perspective in a single place and introduced several clarifications on the Lifecycle View and Security Perspective.

6.2 Requirements Mapping

In the following we present a mapping of the requirements [2] to the architectural views and perspectives, as well as to the phases of the lifecycle. The preliminary analysis shows, that we archive a good level of matching. Only few requirements cannot be mapped, which might be reasoned that the requirement is either to the architecture itself or might be on technology level.

Req. ID	Requirement	View/ Perspective	Phase
Req. F-01	The BaaS reference architecture shall specify mechanisms to support network independent identifiers.	Information View	Development, Engineering, Operation
Req. F-02	The BaaS reference architecture shall specify mechanisms to search for specific entities using semantic queries.	Functional View	Operation, Optimization
Req. F-03	The BaaS reference architecture shall provide mechanisms for authorized access to data and services.	Security Perspective	Development, Engineering, Operation
Req. F-04	The BaaS reference architecture shall support exchangeable authentication and authorization mechanisms.	Functional View / Information View / Lifecycle View	Development, Engineering, Operation
Req. F-05	The BaaS reference architecture shall specify a mechanism to support exchangeable encryption mechanisms.	Functional View / Information View / Lifecycle View	Development, Engineering, Operation
Req. F-06	The BaaS reference architecture shall specify mechanisms to describe and model BaaS entities.	Functional View	Development, Engineering, Operation
Req. F-07	The BaaS reference architecture shall specify mechanisms to extend a basic set of Baas ontologies and BaaS models.	Functional View	Development
Req. F-08	The BaaS reference architecture shall provide mechanisms for system monitoring.	Functional View	Development
Req. F-09	The BaaS reference architecture shall specify mechanisms for creation of instances of BaaS devices and BaaS services and configuration of the relations between these instances.	Functional View	Engineering
Req.	The BaaS reference architecture shall define mechanisms to specify the basic features of a	Functional View /	Engineering

Req. ID	Requirement	View/ Perspective	Phase
F-10	service for the operational phase, like e.g. communication protocol or authorization mechanism.	Information / Lifecycle	
Req. F-11	The system built using the BaaS reference architecture shall provide software components that implement basic features of a service for the operational phase, like e.g. communication protocol or authorization mechanism.	Functional View / Information / Lifecycle	Engineering
Req. F-12	The BaaS reference architecture shall specify mechanisms to support the communication patterns “request-response”, “publish/ subscribe” and "eventing (observer)".	Behavioral View	Engineering, Operation
Req. F-13	The BaaS reference architecture shall specify mechanisms for group communication.	Information View	Engineering, Operation
Req. F-14	The BaaS reference architecture shall specify mechanism to access building automation data.	Information View, Functional View	Operation
Req. F-15	The BaaS reference architecture shall specify mechanisms for implementing user alert services.	Functional View	Engineering
Req. F-16	The BaaS reference architecture shall specify mechanisms for dynamic deployment and removal of BaaS services and BaaS devices.	Functional View	Operation, Optimization
Req. F-17	The BaaS reference architecture shall specify a mechanism for conflict resolution when accessing building automation data.		Operation
Req. F-18	The BaaS reference architecture shall specify mechanisms to access data and meta data independently.	Information View	Operation
Req. F-19	The BaaS reference architecture shall support development of services that use building data models (building geometry, location and data of rooms, floors, sensors).	Functional View	Operation
Req. F-20	The BaaS reference architecture shall specify mechanisms for service lifecycle management.	Functional View	Commissioning, Operation
Req. N-01	A BaaS platform and a BaaS system must be implemented in a way compliant with the BaaS reference architecture.		Development
Req. N-02	The BaaS reference architecture shall specify mechanisms for the integration of external services.	Functional View	Development, Engineering, Operation
Req.	The BaaS reference architecture must define		Development

Req. ID	Requirement	View/ Perspective	Phase
N-03	mechanisms for transferring data/information which should cover common solutions in literature (xml, json, etc.) and also suitable for custom solutions.		
Req. N-04	The BaaS reference architecture shall provide mechanisms for transparent integration of legacy devices.	Functional View	Development, Engineering
Req. N-05	The BaaS reference architecture shall follow a service-oriented paradigm.	Lifecycle View, domain model	
Req. N-06	The BaaS reference architecture shall provide mechanisms to secure the data that are collected and provided.	Security Perspective	Operation
Req. N-07	The BaaS reference architecture should support the developers in designing a system under awareness of the dependability.	Dependability Perspective	Development, Engineering, Operation

7 List of Figures

8 References

[1] C. Niedermeier, Ed., “BaaS - Building as a Service.” ITEA Full Project Proposal, 28-Sep-2012.

[2] Ö. Aydemir, Ed., “D04 BaaS Architecture Requirements.” ITEA BaaS Project, Dec-2014.

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

[4] A. Müller, Ed., “D01 Use Case Scenarios and Requirements.” Jun-2014.

[5] Kieback & Peter, “Gebäudeautomation.” [Online]. Available: http://www.kieback-peter.de/de-de/support-software/woerterbuch-der-gebaeudeautomation.

[6] Kruchten, Philippe, “4+1 architectural view model,” *Wikipedia*. [Online]. Available: http://en.wikipedia.org/wiki/4%2B1_architectural_view_model.

[7] Siemens, “Siemens Glossary.” [Online]. Available: http://www.fullyengineered.com/journals/Siemens/0-91900-en_Glossary_and_Abbreviations.pdf.

[8] “Setpoint (control system),” *Wikipedia*. [Online]. Available: http://en.wikipedia.org/wiki/Setpoint_(control_system).

[9] E. J. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, 1. A. Addison Wesley, 2003.

[10] “ISO 16484-2:2004 Building automation and control systems (BACS) -- Part 2: Hardware.” ISO/TC 205.

[11] 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.

[12] IBM, “An Architectural Blueprint for Autonomic Computing,” IBM, Jun. 2005.

[13] A. Brinkmann, C. Fiehe, A. Litvina, I. Lück, L. Nagel, K. Narayanan, F. Ostermair, and W. Thronicke, “Scalable Monitoring System for Clouds,” presented at the 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013), 3nd International Workshop on Intelligent Techniques and Architectures for Autonomic Clouds (ITAAC 2013), Dresden, Germany, 2013.

[14] Malte Burkert, Heiko Krumm, and Christoph Fiehe, “Technical Management System for Dependable Building Automation Systems,” presented at the 9th International Workshop on Service-Oriented Cyber-Physical Systems in Converging Networked Environments (SOCNE) in conjunction with the 20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2015), Luxembourg, 2015.

[15] A. Pras and J. Schoenwaelder, “On the Difference between Information Models and Data Models,” Internet Engineering Task Force, 2003.

[16] “EN 50126 - Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS),” European Commitee for Electrotechnical Standardization, 1999.

[17] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[18] K. Echtle, *Fehlertoleranzverfahren*. Berlin; Heidelberg; New York; London; Paris; Tokyo; Hong Kong; Barcelona: Springer, 1990.

[19] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Softw. Syst. Model.*, vol. 7, no. 1, pp. 49–65, Nov. 2007.

[20] 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.

[21] L. Lefort, "Ontology for Quantity Kinds and Units," World Wide Web Consortium, 2010.

[22] "SSN-XG," *W3C*, 2011. [Online]. Available: http://www.w3.org/2005/Incubator/ssn/.

[23] "Project Haystack." [Online]. Available: http://project-haystack.org/.

Annex

A Heating System OWL representation

A.1 Basic Data Type definitions

Basic types are used to describe the format in which BaaS exchanges data. Here, we only need two basic types, consisting of temperature and throughput, both represented by floats.

A.1.1 Water Throughput

```
<xs:simpleType name="throughputType">
  <xs:restriction base="xs:float"/>
</xs:simpleType>
<xs:complexType name="throughputRootType">
  <xs:sequence>
    <xs:element type="troughputType" name="throughput"/>
  </xs:sequence>
</xs:complexType>
```

A.1.2 Temperature

```
  <xs:simpleType name="temperatureValueType">
    <xs:restriction base="xs:float">
      <xs:minInclusive value="-60.0"/>
      <xs:maxInclusive value="100.0"/>
    </xs:restriction>
  </xs:simpleType>
<xs:complexType name="temperatureRootType">
  <xs:sequence>
    <xs:element name="temperature" type="temperatureValueType"/>
  </xs:sequence>
</xs:complexType>
```

A.2 Specified Features

A.2.1 Temperature Sensor

```
 <rdf:RDF xmlns="http://www.baas-itea2.eu/dp/ucs#"
xml:base="http://www.baas-itea2.eu/dp/ucs"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sp="http://spinrdf.org/sp#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:spin="http://spinrdf.org/spin#" xmlns:loc="http://www.baas-
itea2.eu/loc#" xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:Information-Model-Objects="http://www.baas-
itea2.eu/EA_Model/Information-Model-Objects/">
```

```
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensor">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature" />

    <rdfs:label
rdf:datatype="&xsd;string">TemperatureSensor</rdfs:label>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasExposedValue" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorTemperature" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorFeatureBAF2" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <description rdf:datatype="&xsd;string">Measures a
temperature.</description>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorTemperature">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Exposed_Value" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataType" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensortemperatureRootType" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#composedOf" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensortemperature" />

      </owl:Restriction>

    </rdfs:subClassOf>
    <dataTypeSource rdf:datatype="&xsd;string">D:/baas-
int/BaaSEditors/XSDFiles/chris.xsd</dataTypeSource>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensortemperature">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Basic_Data" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasExtendedValueCharacteristic" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensortemperatureValueCharacteristic2" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensortemperatureValueType" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <dataTypePath
rdf:datatype="&xsd;string">temperatureRootType.temperatureValueType
temperature</dataTypePath>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorFeatureBAF2">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorSensing" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorSensing">
    <rdfs:label rdf:datatype="&xsd;string">Sensing</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Type" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensortemperatureValueCharacteristic2">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Extended_Value_Characteristic" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasQuantity" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorTemperatureUnit" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasUnit" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorDegreeCelsius" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasRange" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensortemperatureValueCharacteristic2Range"
/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <writeability rdf:datatype="&xsd;boolean">false</writeability>
    <readability rdf:datatype="&xsd;boolean">true</readability>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorDegreeCelsius">
    <rdfs:label rdf:datatype="&xsd;string">DegreeCelsius</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Unit" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensorTemperatureUnit">
    <rdfs:label
rdf:datatype="&xsd;string">TemperatureUnit</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Quantity" />
  </owl:Class>
```

```
</rdf:RDF>
```

A.2.2 Flow Setpoint

```
 <rdf:RDF xmlns="http://www.baas-itea2.eu/dp/ucs#"
xml:base="http://www.baas-itea2.eu/dp/ucs"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sp="http://spinrdf.org/sp#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:spin="http://spinrdf.org/spin#" xmlns:loc="http://www.baas-
itea2.eu/loc#" xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:Information-Model-Objects="http://www.baas-
itea2.eu/EA_Model/Information-Model-Objects/">

  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#FlowSetpoint">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature" />

    <rdfs:label rdf:datatype="&xsd;string">FlowSetpoint</rdfs:label>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasExposedValue" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointdesiredThroughput" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointFeatureBAF4" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <description rdf:datatype="&xsd;string">Describes the desired
throughput of some water-processing device, such as a valve or a
pump.</description>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointdesiredThroughput">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Exposed_Value" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataType" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointthroughputRootType" />
```

```
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#composedOf" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointthroughput" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <dataTypeSource rdf:datatype="&xsd;string">D:/baas-
int/BaaSEditors/XSDFiles/chris.xsd</dataTypeSource>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointthroughput">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Basic_Data" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasExtendedValueCharacteristic" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointthroughputValueCharacteristic4" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointtroughputType" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <dataTypePath
rdf:datatype="&xsd;string">throughputRootType.troughputType
throughput</dataTypePath>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointFeatureBAF4">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFType" />
```

```
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointSetPoint" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointSetPoint">
    <rdfs:label rdf:datatype="&xsd;string">SetPoint</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Type" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointthroughputValueCharacteristic4">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Extended_Value_Characteristic" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasQuantity" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointVolumePerTimeUnit" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasUnit" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointCubicMeterPerHour" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasRange" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointthroughputValueCharacteristic4Range" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <writeability rdf:datatype="&xsd;boolean">true</writeability>
    <readability rdf:datatype="&xsd;boolean">true</readability>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointCubicMeterPerHour">
```

```
    <rdfs:label
rdf:datatype="&xsd;string">CubicMeterPerHour</rdfs:label>

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Unit" />

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FlowSetpointVolumePerTimeUnit">

    <rdfs:label
rdf:datatype="&xsd;string">VolumePerTimeUnit</rdfs:label>

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Quantity" />

  </owl:Class>

</rdf:RDF>
```

A.3 Specified Data Point Types

A.3.1 Temperature Sensors

The temperature sensors measure temperatures in different parts of the heating system. Since they are only differentiated by different semantic tags, only one of the sensor's OWL representations (that of the feed temperature sensor) will be shown here.

```
 <rdf:RDF xmlns="http://www.baas-itea2.eu/dp/ucs#"
xml:base="http://www.baas-itea2.eu/dp/ucs"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sp="http://spinrdf.org/sp#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:spin="http://spinrdf.org/spin#" xmlns:loc="http://www.baas-
itea2.eu/loc#" xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:Information-Model-Objects="http://www.baas-
itea2.eu/EA_Model/Information-Model-Objects/">


  <!--http://www.baas-itea2.eu/dp/ucs#Data_Point_Type-->

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensor">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Data_Point_Type" />

    <rdfs:label
rdf:datatype="&xsd;string">FeedTemperatureSensor</rdfs:label>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeatureBinding" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorFeedTemperature" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>
```

```
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorBAF2" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <description rdf:datatype="&xsd;string">A temperature sensor
connected to the heating system feed pipe.</description>
    <extendedDPT rdf:datatype="&xsd;string"></extendedDPT>
  </owl:Class>


  <!--http://www.baas-itea2.eu/dp/ucs#Feature_Binding-->
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorFeedTemperature">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature_Binding" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeature" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensor" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorFeedTemperatureBAF1" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>


  <!--http://www.baas-itea2.eu/dp/ucs#Building_Automation_Function-->
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorFeedTemperatureBAF1">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFDomain" />
```

```
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorHeating" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFDomain" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorWater" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>



  <!--http://www.baas-itea2.eu/dp/ucs#BAF_Domain-->
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorHeating">
    <rdfs:label rdf:datatype="&xsd;string">Heating</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Domain" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorWater">
    <rdfs:label rdf:datatype="&xsd;string">Water</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Domain" />
  </owl:Class>









  <!--http://www.baas-itea2.eu/dp/ucs#Building_Automation_Function-->
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorBAF2">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFContext" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorHeatingSystem#1" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorFeedTemperatureSensor" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorSensor" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>


  <!--http://www.baas-itea2.eu/dp/ucs#BAF_Type-->
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorFeedTemperatureSensor">
    <rdfs:label
rdf:datatype="&xsd;string">FeedTemperatureSensor</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Custom_BAF_Type" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorSensor">
    <rdfs:label rdf:datatype="&xsd;string">Sensor</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Type" />
  </owl:Class>




  <!--http://www.baas-itea2.eu/dp/ucs#BAF_Context-->
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#FeedTemperatureSensorHeatingSystem#1">
    <rdfs:label
rdf:datatype="&xsd;string">HeatingSystem#1</rdfs:label>
```

```
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Context" />

    <subSystem rdf:datatype="&xsd;string"></subSystem>

    <subSubSystem rdf:datatype="&xsd;string"></subSubSystem>

  </owl:Class>

</rdf:RDF>
```

A.3.2 Boiler

```
 <rdf:RDF xmlns="http://www.baas-itea2.eu/dp/ucs#"
xml:base="http://www.baas-itea2.eu/dp/ucs"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sp="http://spinrdf.org/sp#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:spin="http://spinrdf.org/spin#" xmlns:loc="http://www.baas-
itea2.eu/loc#" xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:Information-Model-Objects="http://www.baas-
itea2.eu/EA_Model/Information-Model-Objects/">

  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#Boiler">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Data_Point_Type" />

    <rdfs:label rdf:datatype="&xsd;string">Boiler</rdfs:label>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeatureBinding" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerDesiredTemperature" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasInputRelation" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerOutsideTemperatureInput" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasInputRelation" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerReturnTemperatureInput" />

      </owl:Restriction>

    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasInputRelation" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerFeedTemperatureInput" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerBAF4" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <description rdf:datatype="&xsd;string">A boiler and heating
controller, taking input from three temperature sensors (feed, return,
outside). A desired input temperature is settable via a temperature
setpoint.</description>
    <extendedDPT rdf:datatype="&xsd;string"></extendedDPT>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerDesiredTemperature">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature_Binding" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeature" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSetpoint" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerDesiredTemperatureBAF0" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerDesiredTemperatureBAF0">
```

```
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFDomain" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerHeating" />

      </owl:Restriction>

    </rdfs:subClassOf>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerHeating">

    <rdfs:label rdf:datatype="&xsd;string">Heating</rdfs:label>

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Domain" />

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerOutsideTemperatureInput">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Input_Relation" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataTypeFilter" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerOutsideTemperatureInputDataTypeFilter" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <dataPointFilter rdf:datatype="&xsd;string">SELECT DISTINCT *
WHERE { {{ SELECT DISTINCT ?dp WHERE { ?dp rdfs:subClassOf
:Data_Point_Type . ?dp rdfs:subClassOf ?bafRes . ?bafRes
owl:allValuesFrom ?baf . ?baf rdfs:subClassOf ?hasRes . ?hasRes
owl:allValuesFrom ?value . ?dp3 rdfs:subClassOf :Data_Point_Type .
?dp3 rdfs:subClassOf ?bafRes3 . ?bafRes3 owl:allValuesFrom ?baf3 .
?baf3 rdfs:subClassOf ?hasRes3 . ?hasRes3 owl:allValuesFrom ?value3 .
FILTER ( ?value = :HeatingSystem#1 && ?dp = ?dp3 && ?value3 =
:OutsideTemperatureSensor ) . } } .}  UNION {{ SELECT DISTINCT ?dp
WHERE { ?hasBaf rdfs:subClassOf :Feature_Binding . ?dp rdfs:subClassOf
?ftb . ?ftb owl:allValuesFrom ?hasBaf . ?hasBaf rdfs:subClassOf
?bafRes . ?bafRes owl:allValuesFrom ?baf . ?baf rdfs:subClassOf
?hasRes . ?hasRes owl:allValuesFrom ?value . ?hasBaf3 rdfs:subClassOf
:Feature_Binding . ?dp3 rdfs:subClassOf ?ftb3 . ?ftb3
owl:allValuesFrom ?hasBaf3 . ?hasBaf3 rdfs:subClassOf ?bafRes3 .
?bafRes3 owl:allValuesFrom ?baf3 . ?baf3 rdfs:subClassOf ?hasRes3 .
?hasRes3 owl:allValuesFrom ?value3 .  FILTER ( ?value =
:HeatingSystem#1 && ?dp = ?dp3 && ?value3 = :OutsideTemperatureSensor
) . } } . }  UNION {{ SELECT DISTINCT ?dp WHERE { ?hasBaf
rdfs:subClassOf :Feature . ?dp rdfs:subClassOf ?dpRes . ?dpRes
owl:allValuesFrom ?ftb . ?ftb rdfs:subClassOf ?ft . ?ft
owl:allValuesFrom ?hasBaf . ?hasBaf rdfs:subClassOf ?bafRes . ?bafRes
owl:allValuesFrom ?baf . ?baf rdfs:subClassOf ?hasRes . ?hasRes
owl:allValuesFrom ?value . ?hasBaf3 rdfs:subClassOf :Feature . ?dp3
rdfs:subClassOf ?dpRes3 . ?dpRes3 owl:allValuesFrom ?ftb3 . ?ftb3
rdfs:subClassOf ?ft3 . ?ft3 owl:allValuesFrom ?hasBaf3 . ?hasBaf3
rdfs:subClassOf ?bafRes3 . ?bafRes3 owl:allValuesFrom ?baf3 . ?baf3
rdfs:subClassOf ?hasRes3 . ?hasRes3 owl:allValuesFrom ?value3 .
FILTER ( ?value = :HeatingSystem#1 && ?dp = ?dp3 && ?value3 =
:OutsideTemperatureSensor ) . } } .} . }</dataPointFilter>

    <simpleQuery rdf:datatype="&xsd;string">hasBAFContext =
HeatingSystem#1

 AND hasBAFType = OutsideTemperatureSensor

</simpleQuery>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerReturnTemperatureInput">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Input_Relation" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataTypeFilter" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerReturnTemperatureInputDataTypeFilter" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <dataPointFilter rdf:datatype="&xsd;string">SELECT DISTINCT *
WHERE { {{ SELECT DISTINCT ?dp WHERE { ?dp rdfs:subClassOf
:Data_Point_Type . ?dp rdfs:subClassOf ?bafRes . ?bafRes
owl:allValuesFrom ?baf . ?baf rdfs:subClassOf ?hasRes . ?hasRes
owl:allValuesFrom ?value . ?dp3 rdfs:subClassOf :Data_Point_Type .
?dp3 rdfs:subClassOf ?bafRes3 . ?bafRes3 owl:allValuesFrom ?baf3 .
?baf3 rdfs:subClassOf ?hasRes3 . ?hasRes3 owl:allValuesFrom ?value3 .
FILTER ( ?value = :HeatingSystem#1 && ?dp = ?dp3 && ?value3 =
:ReturnTemperatureSensor ) . } } .}  UNION {{ SELECT DISTINCT ?dp
WHERE { ?hasBaf rdfs:subClassOf :Feature_Binding . ?dp rdfs:subClassOf
?ftb . ?ftb owl:allValuesFrom ?hasBaf . ?hasBaf rdfs:subClassOf
?bafRes . ?bafRes owl:allValuesFrom ?baf . ?baf rdfs:subClassOf
?hasRes . ?hasRes owl:allValuesFrom ?value . ?hasBaf3 rdfs:subClassOf
:Feature_Binding . ?dp3 rdfs:subClassOf ?ftb3 . ?ftb3
owl:allValuesFrom ?hasBaf3 . ?hasBaf3 rdfs:subClassOf ?bafRes3 .
?bafRes3 owl:allValuesFrom ?baf3 . ?baf3 rdfs:subClassOf ?hasRes3 .
?hasRes3 owl:allValuesFrom ?value3 .  FILTER ( ?value =
:HeatingSystem#1 && ?dp = ?dp3 && ?value3 = :ReturnTemperatureSensor )
. } } . }  UNION {{ SELECT DISTINCT ?dp WHERE { ?hasBaf
rdfs:subClassOf :Feature . ?dp rdfs:subClassOf ?dpRes . ?dpRes
owl:allValuesFrom ?ftb . ?ftb rdfs:subClassOf ?ft . ?ft
owl:allValuesFrom ?hasBaf . ?hasBaf rdfs:subClassOf ?bafRes . ?bafRes
owl:allValuesFrom ?baf . ?baf rdfs:subClassOf ?hasRes . ?hasRes
owl:allValuesFrom ?value . ?hasBaf3 rdfs:subClassOf :Feature . ?dp3
rdfs:subClassOf ?dpRes3 . ?dpRes3 owl:allValuesFrom ?ftb3 . ?ftb3
rdfs:subClassOf ?ft3 . ?ft3 owl:allValuesFrom ?hasBaf3 . ?hasBaf3
rdfs:subClassOf ?bafRes3 . ?bafRes3 owl:allValuesFrom ?baf3 . ?baf3
rdfs:subClassOf ?hasRes3 . ?hasRes3 owl:allValuesFrom ?value3 .
FILTER ( ?value = :HeatingSystem#1 && ?dp = ?dp3 && ?value3 =
:ReturnTemperatureSensor ) . } } .} . }</dataPointFilter>
```

```
    <simpleQuery rdf:datatype="&xsd;string">hasBAFContext = 
HeatingSystem#1

 AND hasBAFType = ReturnTemperatureSensor

</simpleQuery>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerFeedTemperatureInput">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Input_Relation" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataTypeFilter" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerFeedTemperatureInputDataTypeFilter" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <dataPointFilter rdf:datatype="&xsd;string">SELECT DISTINCT * 
WHERE { {{ SELECT DISTINCT ?dp WHERE { ?dp rdfs:subClassOf 
:Data_Point_Type . ?dp rdfs:subClassOf ?bafRes . ?bafRes 
owl:allValuesFrom ?baf . ?baf rdfs:subClassOf ?hasRes . ?hasRes 
owl:allValuesFrom ?value . ?dp3 rdfs:subClassOf :Data_Point_Type . 
?dp3 rdfs:subClassOf ?bafRes3 . ?bafRes3 owl:allValuesFrom ?baf3 . 
?baf3 rdfs:subClassOf ?hasRes3 . ?hasRes3 owl:allValuesFrom ?value3 . 
FILTER ( ?value = :HeatingSystem#1 && ?dp = ?dp3 && ?value3 = 
:FeedTemperatureSensor ) . } } .}  UNION {{ SELECT DISTINCT ?dp WHERE 
{ ?hasBaf rdfs:subClassOf :Feature_Binding . ?dp rdfs:subClassOf ?ftb 
. ?ftb owl:allValuesFrom ?hasBaf . ?hasBaf rdfs:subClassOf ?bafRes . 
?bafRes owl:allValuesFrom ?baf . ?baf rdfs:subClassOf ?hasRes . 
?hasRes owl:allValuesFrom ?value . ?hasBaf3 rdfs:subClassOf 
:Feature_Binding . ?dp3 rdfs:subClassOf ?ftb3 . ?ftb3 
owl:allValuesFrom ?hasBaf3 . ?hasBaf3 rdfs:subClassOf ?bafRes3 . 
?bafRes3 owl:allValuesFrom ?baf3 . ?baf3 rdfs:subClassOf ?hasRes3 . 
?hasRes3 owl:allValuesFrom ?value3 .  FILTER ( ?value = 
:HeatingSystem#1 && ?dp = ?dp3 && ?value3 = :FeedTemperatureSensor ) . 
} } . }  UNION {{ SELECT DISTINCT ?dp WHERE { ?hasBaf rdfs:subClassOf 
:Feature . ?dp rdfs:subClassOf ?dpRes . ?dpRes owl:allValuesFrom ?ftb 
. ?ftb rdfs:subClassOf ?ft . ?ft owl:allValuesFrom ?hasBaf . ?hasBaf 
rdfs:subClassOf ?bafRes . ?bafRes owl:allValuesFrom ?baf . ?baf 
rdfs:subClassOf ?hasRes . ?hasRes owl:allValuesFrom ?value . ?hasBaf3 
rdfs:subClassOf :Feature . ?dp3 rdfs:subClassOf ?dpRes3 . ?dpRes3 
owl:allValuesFrom ?ftb3 . ?ftb3 rdfs:subClassOf ?ft3 . ?ft3 
owl:allValuesFrom ?hasBaf3 . ?hasBaf3 rdfs:subClassOf ?bafRes3 . 
?bafRes3 owl:allValuesFrom ?baf3 . ?baf3 rdfs:subClassOf ?hasRes3 . 
?hasRes3 owl:allValuesFrom ?value3 .  FILTER ( ?value = 
:HeatingSystem#1 && ?dp = ?dp3 && ?value3 = :FeedTemperatureSensor ) . 
} } .} . }</dataPointFilter>

    <simpleQuery rdf:datatype="&xsd;string">hasBAFContext = 
HeatingSystem#1

 AND hasBAFType = FeedTemperatureSensor

</simpleQuery>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#BoilerBAF4">
```

```
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFContext" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerHeatingSystem#1" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFDomain" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerHeating" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilerDevice" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#BoilerDevice">
    <rdfs:label rdf:datatype="&xsd;string">Device</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Type" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerHeating">
    <rdfs:label rdf:datatype="&xsd;string">Heating</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Domain" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerHeatingSystem#1">
    <rdfs:label
rdf:datatype="&xsd;string">HeatingSystem#1</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Context" />
    <subSystem rdf:datatype="&xsd;string"></subSystem>
    <subSubSystem rdf:datatype="&xsd;string"></subSubSystem>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerOutsideTemperatureInputDataTypeFilter">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Complex_Data" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilertemperatureRootType" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <dataTypeSource rdf:datatype="&xsd;string">D:/baas-
siemens/workpackages/wp4/Datapoints/XSDFiles/Control.xsd</dataTypeSour
ce>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerReturnTemperatureInputDataTypeFilter">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Complex_Data" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilertemperatureRootType" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <dataTypeSource rdf:datatype="&xsd;string">D:/baas-
siemens/workpackages/wp4/Datapoints/XSDFiles/Control.xsd</dataTypeSour
ce>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#BoilerFeedTemperatureInputDataTypeFilter">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Complex_Data" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasDataType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BoilertemperatureRootType" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <dataTypeSource rdf:datatype="&xsd;string">D:/baas-
siemens/workpackages/wp4/Datapoints/XSDFiles/Control.xsd</dataTypeSour
ce>

  </owl:Class>

</rdf:RDF>
```

A.3.3 Hot Water Cylinder

```
<rdf:RDF xmlns="http://www.baas-itea2.eu/dp/ucs#"
xml:base="http://www.baas-itea2.eu/dp/ucs"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sp="http://spinrdf.org/sp#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:spin="http://spinrdf.org/spin#" xmlns:loc="http://www.baas-
itea2.eu/loc#" xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:Information-Model-Objects="http://www.baas-
itea2.eu/EA_Model/Information-Model-Objects/">

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinder">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Data_Point_Type" />

    <rdfs:label
rdf:datatype="&xsd;string">HotWaterCylinder</rdfs:label>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeatureBinding" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderCurrentTemperature" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeatureBinding" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderdesiredTemperature" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderBAF7" />

      </owl:Restriction>

    </rdfs:subClassOf>
    <description rdf:datatype="&xsd;string">A hot water storage
cylinder with embedded electric heating system and
controller.</description>

    <extendedDPT rdf:datatype="&xsd;string"></extendedDPT>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderCurrentTemperature">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature_Binding" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeature" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSensor" />

      </owl:Restriction>

    </rdfs:subClassOf>

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderCurrentTemperatureBAF10" />

      </owl:Restriction>

    </rdfs:subClassOf>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderdesiredTemperature">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature_Binding" />

    <rdfs:subClassOf>

      <owl:Restriction>

        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeature" />

        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#TemperatureSetpoint" />

      </owl:Restriction>

    </rdfs:subClassOf>

  </owl:Class>

  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderCurrentTemperatureBAF10">

    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />

    <rdfs:subClassOf>

      <owl:Restriction>
```

```
        <owl:onProperty rdf:resource="http://www.baas-itea2.eu/dp/ucs#hasBAFDomain" />
        <owl:allValuesFrom rdf:resource="http://www.baas-itea2.eu/dp/ucs#HotWaterCylinderWater" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#HotWaterCylinderWater">
    <rdfs:label rdf:datatype="&xsd;string">Water</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-itea2.eu/dp/ucs#BAF_Domain" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#HotWaterCylinderBAF7">
    <rdfs:subClassOf rdf:resource="http://www.baas-itea2.eu/dp/ucs#Building_Automation_Function" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-itea2.eu/dp/ucs#hasBAFContext" />
        <owl:allValuesFrom rdf:resource="http://www.baas-itea2.eu/dp/ucs#HotWaterCylinderHeatingSystem#1" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-itea2.eu/dp/ucs#hasBAFDomain" />
        <owl:allValuesFrom rdf:resource="http://www.baas-itea2.eu/dp/ucs#HotWaterCylinderHeating" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-itea2.eu/dp/ucs#hasBAFType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-itea2.eu/dp/ucs#HotWaterCylinderDevice" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#HotWaterCylinderDevice">
    <rdfs:label rdf:datatype="&xsd;string">Device</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Type" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderHeating">
    <rdfs:label rdf:datatype="&xsd;string">Heating</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Domain" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#HotWaterCylinderHeatingSystem#1">
    <rdfs:label
rdf:datatype="&xsd;string">HeatingSystem#1</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Context" />
    <subSystem rdf:datatype="&xsd;string"></subSystem>
    <subSubSystem rdf:datatype="&xsd;string"></subSubSystem>
  </owl:Class>
</rdf:RDF>
```

A.3.4 Pumps

```
 <rdf:RDF xmlns="http://www.baas-itea2.eu/dp/ucs#"
xml:base="http://www.baas-itea2.eu/dp/ucs"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sp="http://spinrdf.org/sp#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:spin="http://spinrdf.org/spin#" xmlns:loc="http://www.baas-
itea2.eu/loc#" xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:Information-Model-Objects="http://www.baas-
itea2.eu/EA_Model/Information-Model-Objects/">
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#Pump">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Data_Point_Type" />
    <rdfs:label rdf:datatype="&xsd;string">Pump</rdfs:label>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeatureBinding" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#PumpcurrentFlow" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeatureBinding" />
```

```
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#PumpdesiredFlow" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#implementsFunction" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#PumpBAF8" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <description rdf:datatype="&xsd;string">A water pump including a
controller and a flow sensor, allowing setting a desired flow
rate.</description>
    <extendedDPT rdf:datatype="&xsd;string"></extendedDPT>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#PumpcurrentFlow">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature_Binding" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeature" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSensor" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#PumpdesiredFlow">
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Feature_Binding" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasFeature" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#FlowSetpoint" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#PumpBAF8">
```

```
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#Building_Automation_Function" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFContext" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#PumpHeatingSystem#1" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFDomain" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#PumpHeating" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFDomain" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#PumpWater" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="http://www.baas-
itea2.eu/dp/ucs#hasBAFType" />
        <owl:allValuesFrom rdf:resource="http://www.baas-
itea2.eu/dp/ucs#PumpDevice" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#PumpDevice">
    <rdfs:label rdf:datatype="&xsd;string">Device</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Type" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#PumpHeating">
    <rdfs:label rdf:datatype="&xsd;string">Heating</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Domain" />
```

```
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-itea2.eu/dp/ucs#PumpWater">
    <rdfs:label rdf:datatype="&xsd;string">Water</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Domain" />
  </owl:Class>
  <owl:Class rdf:about="http://www.baas-
itea2.eu/dp/ucs#PumpHeatingSystem#1">
    <rdfs:label
rdf:datatype="&xsd;string">HeatingSystem#1</rdfs:label>
    <rdfs:subClassOf rdf:resource="http://www.baas-
itea2.eu/dp/ucs#BAF_Context" />
    <subSystem rdf:datatype="&xsd;string"></subSystem>
    <subSubSystem rdf:datatype="&xsd;string"></subSubSystem>
  </owl:Class>
</rdf:RDF>
```