

An Engineering Tool Suite for Enhanced Smart Environments: GAMMA

Geliştirilmiş Akıllı Mekanlar için Mühendislik Araç Süiti: GAMMA

Metin Tekkalmaz¹ Emre Tapcı² and Özer Aydemir¹

¹ IOTIQ GmbH, Leipzig, Germany
{metin, ozer}@iotiq.de

² ERSTE Yazılım Ltd., Ankara, Turkey
emre@ersteyazilim.com

Abstract. Recent advancements in sensing, networking and computational power have made widespread development and deployment of smart environments possible. On the other hand, rapid design, development and deployment of such systems are still difficult and complex. In this paper, we present design and implementation of an engineering suite which eases smart environment creation process based on a recently proposed lifecycle.

Öz. Algılayıcılar, bilgisayar ağları ve işlemci gücü konularındaki son gelişmeler akıllı mekanların yaygın olarak geliştirilmesinin ve kurulumunun önünü açmıştır. Diğer taraftan bu tarz sistemlerin hızlı şekilde tasarımı, geliştirilmesi ve kurulumu halen zor ve karmaşıktır. Bu makalede, akıllı mekanların oluşturulması sürecini kolaylaştıran ve yakın dönemde sunulan bir yaşam döngüsünü temel alan mühendislik araç süitinin tasarım ve gerçekleştirilmesi verilmektedir.

Keywords: IoT, internet of things, smart environments, software product lines.

Anahtar Kelimeler: nesnelerin interneti, akıllı mekanlar, yazılım ürün hatları.

1 Introduction

Today, people live in buildings not only consist of concrete and bricks but also contain mechanisms to ease their life and increase their comfort. While those mechanisms were only pneumatic transmission at the beginning, it evolved to electric controls, direct digital controls, wireless interfaces, alarms and many steps between them. After having adequate bandwidth, network infrastructure and computational power, we reached the last step of this evolution: IT standardized information presentation models and

intelligent environments strengthened by software paradigms such as artificial intelligence, machine learning and data mining. Even though this last step promises real smart environments with decision making capabilities, it brings complexity on smart environment creation especially in the following areas: (1) Vendor dependency, (2) Device Discovery, (3) Late binding.

Existing needs and problems require having a fully compliant smart environment creation platform. Vicari et al. [1] proposed a lifecycle which defines each steps of smart environment creation and proved the efficiency of lifecycle after having a proof of concept platform as a result of European Research Project BaaS - Building as a Service [2].

We, as IOTIQ GmbH and ERSTE Yazılım Ltd., focus on extending this lifecycle according to the needs of not only smart buildings but also Industry 4.0, smart cities, smart healthcare, smart agriculture, etc. At the end of the project, we are going to have a fully compliant, commercial toolset, namely Engineering Tools for Enhanced Smart Environments (*Turkish* Geliştirilmiş Akıllı Mekanlar için Mühendislik Araçları - GAMMA), which helps “smart environment creators” to design, develop, engineer, commission, operate and optimize smart environment solutions.

In this paper we introduce our interpretation and implementation of smart environment creation lifecycle. In the next section we briefly explain the steps of the lifecycle and their interrelations. In Section 3, we explain our domain model and software architecture, and then, in Section 4, we briefly present our implementation approach. In Section 5, we mention some related work and evaluate our approach in comparison with other similar studies. Finally, in Section 6, we conclude with some future works.

2 Smart Environment Development Lifecycle

Smart environment creation lifecycle describes different steps related to design time and run time of a smart environment system. It defines the activities, artifacts in each phase and relations of each phase with the other phases. The lifecycle is depicted in **Fig. 1** and then each step is explained briefly.

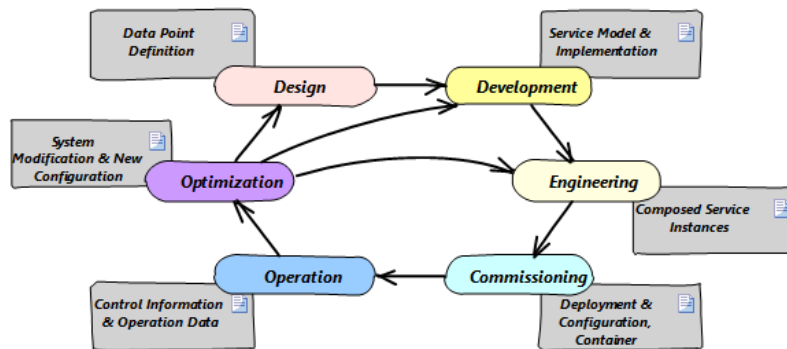


Fig. 1. Smart environment creation lifecycle as described in [1] and [2].

Design. In the Design phase, the models for the hardware (i.e. physical devices) and software functionalities are defined as a central structure. The models are the entities used in the smart environments and they constitute building blocks of the resulting application. The lifecycle starts with definition of Data Point Types.

Development. After having the descriptions of physical devices and other functionalities through Design phase, we need to access primitive services of those models through Development phase. In order to fulfil this need, we define Service Types based on requirements and constraints. These Service Types are strongly based on Data Point Type defined in Design phase. Service Types from development phase are the main ingredients for Engineering phase.

Engineering. At the beginning of Engineering phase, we need to have models through Design and access to primitive services through Development. This phase is the main phase where complex value added services (VAS) are formed through models and primitive services. In order to provide those VAS running with defined rules, the smart environment engineer needs to specify the relations here.

Commissioning. In the Commissioning phase, we need to deploy the developed VAS provided by the Engineering phase. The Commissioning phase needs to have a list of physical devices and list of data point types used by corresponding VAS. In this phase, physical devices are mapped into data models defined in Design phase.

Operation. Operation phase differs according to the domain or use case the system is designed for, and the main responsibility of this phase is to provide the end user applications to operate the deployed system. For instance, letting user to tell the desired temperature and/or humidity to a climatization VAS is the under responsibility of operation tools.

Optimization. Each developed value added service needs to come up with their quantified objectives. Either they should decrease the consumption, or increase the comfort or both. In order to check whether they fulfill the Key Performance Indicators (KPIs) they promised, we need to collect the data on runtime and check the consistency of data according to defined KPIs. Also collecting errors and warnings in run time is another responsibility of optimization phase. Optimization phase helps to improve the development of subsequent versions of the smart environment applications.

3 Domain Model of GAMMA Platform

To be able to realize the tools supporting the smart environment development cycle described in Section 2, a domain analysis [3][4] activity is carried. In the GAMMA context, the main purpose of the domain analysis was to have a good understanding of the smart environment domain, therefore

- domain terms with their clear definitions are extracted, hence a common ground for the stakeholders are established,
- domain boundaries can be drawn, e.g. decide what is in the domain and what is not,
- models, containing the major elements along with their interrelations, are specified.

As a result, a domain model, a data model and a dictionary describing the main components in the model are developed. In this section, we first provide the domain dictionary, which contains the terms used in the provided models, in **Table 1**. We then explain the simplified top-level domain model which is depicted in **Fig. 2** as well as the data model using a sample diagram from the data model which is depicted in **Fig. 3**.

Table 1. Main terms from the GAMMA domain dictionary

Term	Description
Actuator	<i>Physical Entity</i> which is capable of changing state of the environment. Example: blind control, heater
Automation Function	Signifies the role in an automation system (e.g. building automation system). Defined by function, field (e.g. heating), type (e.g. sensing) and context (e.g. heating system #1).
Basic Data Type	The basic data types provided by the underlying platform. All other higher-level types are defined in terms of <i>Basic Data Types</i> . Example: integer, float, string, boolean, date/time, duration etc.
Container	Software platform which can run and manage one or more <i>Service</i> .
Data Point Type	Formally describes an entity in terms of features it provides or requires. Contains dependencies to other <i>Data Point Types</i> , its context within building automation domain and other required information.
Data Store	Repository that is used to store <i>Data Types</i> , <i>Feature Types</i> , <i>Service Types</i> , <i>System Models</i> and the relations among them.
Data Type	Describes a type in terms of <i>Basic Data Types</i> and/or other <i>Data Types</i> with additional semantic information and possible restrictions. Example: Wind velocity data type may contain two basic data types of float, units of knot, with certain min/max restrictions to express north and east components of the velocity vector.
Feature Type	Describes a functionality of an entity. Major components of a <i>Feature Type</i> include an exposed value of type <i>Data Type</i> , parameters of type <i>Data Types</i> , and context of the functionality in building automation domain. Example: wind speed measuring feature
Interface	Defines the data exchange method of a service with another service. A <i>Service</i> may provide an <i>Interface</i> for letting other services to access the data it generates or may require an <i>Interface</i> to access data generated by the other <i>Services</i> .
Parameter	Specifies fixed or configurable features for a type. A <i>Parameter</i> can be assigned a value at the stage that the it is defined (e.g. <i>Feature Type</i> , <i>Data Point Type</i> , <i>Service Type</i> declaration) or at a later stage (e.g. <i>Data Point Type</i> , <i>Service Type</i> , <i>Service</i> declaration).
Parameter Value	Signifies a value assigned to a parameter, e.g. 3, "meeting room", 192.168.1.1.
Physical Entity	Physical devices used by the GAMMA platform.
Registry	Software and related data to search and query <i>Services</i> and <i>Containers</i> in the system.
Sensor	<i>Physical Entity</i> which senses environment and obtains state information about it. Example: temperature sensor, door open/close sensor
Server	Hardware/Software hosting <i>Containers</i> in GAMMA platform. A Server may also contain communication interfaces.

Term	Description
Service	<i>Service Type</i> instances that either communicate with a <i>Physical Entities</i> to make its functionality accessible to GAMMA platform or implement <i>Virtual Entities</i> .
Service Type	An item based on <i>Data Point Type</i> and composed of implementation and parameter values. It can be compiled to communicate with a <i>Physical Entity</i> or realize a <i>Virtual Entity</i> .
System Model	Model which contains the necessary information (e.g. <i>Entities</i> , <i>Service Types</i> that can communicate or implement those <i>Entities</i> , the relations among these, etc.) in order to generate the system satisfying the user requirements.
Virtual Entity	An <i>Entity</i> which provides additional functionality (most probably) based on the functionality provided by other <i>Physical</i> or <i>Virtual Entities</i> .

In classical software product line engineering (SPLE), the main goal is to have a platform with reusable components, in which variabilities and commonalities are implemented. Then, ideally, these components are composed to produce the final product according to a given set of requirements. Apart from the classical SPLE, one of the main purposes of GAMMA is to provide a platform so that the reusable components of the product line engineering are also easily developed in harmony. Later, these components are composed to produce the final products and deployed, again with the help of GAMMA platform.

Considering the business target of the GAMMA project, the main variability point of the final products is the different IoT devices and different value-added functionalities in smart building context. In the domain model, these are depicted as the Entities, which can be either Physical or Virtual Entity (see **Fig. 2**). In the GAMMA platform, these Entities are represented as Data Point Types (DPTs) and domain engineers are expected to introduce necessary DPTs to the platform. As an example, for a temperature sensor, which is a Sensor, there should be a DPT representing it. Note that, there might be multiple types of temperature sensors, from different manufacturers with different measurement units, with different communication protocols, etc. But as long as they all provide the same functionality (i.e. sensing the temperature and providing the corresponding value) they can be represented with the same DPT. DPTs mainly use Feature Types to represent Entities. Each DPT may provide and/or require one or more features. Back to the temperature sensor example, the DPT representing the temperature sensor *provides* the “temperature sensing” feature, while another DPT (e.g. a heater which tries to keep the environment above a certain temperature) may *require* the same “temperature sensing” feature. Each Feature Type has an exposed Data Type. In our example “temperature sensing” feature may have “temperature” type as the exposed type and “temperature” Data Type may be a Primitive Data Type of type “float” Basic Data Type with degrees Celsius as the unit (see **Fig. 3** and later discussion on Data Types).

Until now, parts of the domain model mostly related with the representation of entities are described. Creating DPTs (and subtypes which the DPTs are based on) is basically the responsibility of “domain engineer”. “Software engineer” is responsible for creating the Service Types. Each Service Type is the reflection a DPT in coding realm. GAMMA platform creates a skeleton code for the software engineer to fill the missing parts for that service type to fulfill its duties according to the provided and required

features of the DPT it is based on. For example, for the “Temperature Sensor” DPT, there might be a “ZigBee Temperature Sensor” Service Type in which “software engineer” has provided the necessary coding to communicate with the sensor using ZigBee protocol, get the temperature value and serve it. There might be another Service Type called “ZWave Temperature Sensor” based on the same DPT but having another internal implementation. Hence in a final product depending on the actual temperature sensors in the system, instances of either of these Service Types can be used without affecting the rest of the system since these are based on the same DPT, therefore have the same external interface.

It is the responsibility of “system engineer” to translate the requirements of a product into a set of Service Types and relations among these. In order to do this, “system engineer” decides the types and number of sensors/actuators, the value added functionalities, user interfaces, etc. as well as the data flow among these. Then he/she uses GAMMA platform to describe the system as a System Model. This System Model can be used to generate the deployable package composed of Service Type code and generated glue code.

GAMMA platform also provides tools to deploy the generated deployable packages (mainly to guide/automate the installation and to declare parameters which cannot be determined at design time, such as network addresses of devices), operate the deployed system and to monitor it to collect feedback information for the earlier phases of smart environment development.

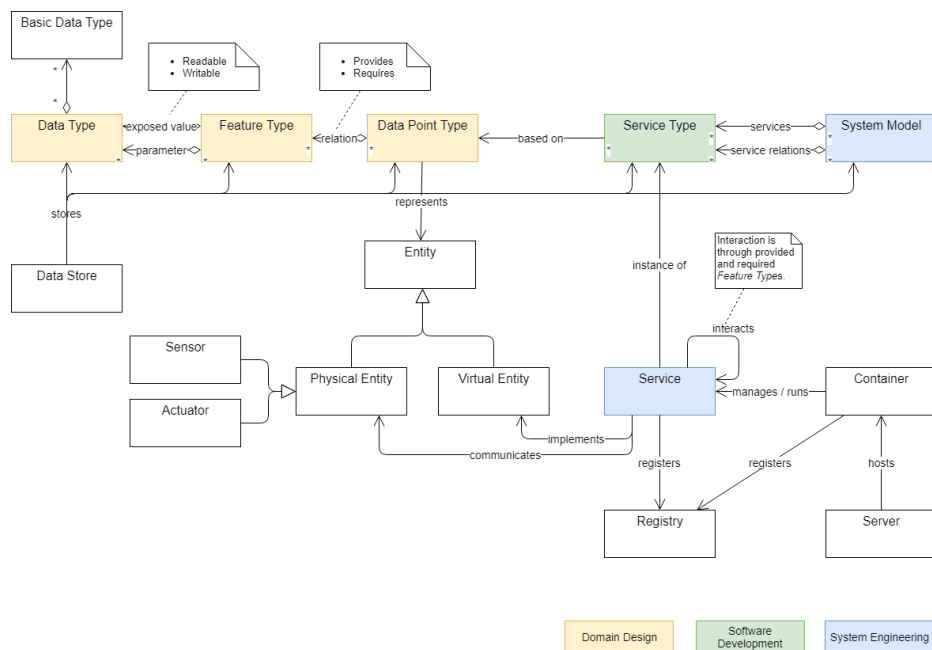


Fig. 2. Simplified GAMMA domain model

In this paper, we do not provide the data model in detail but describe part of it to give an idea. **Fig. 3** depicts the diagram formally describing the Data Types in the GAMMA platform. A Data Type can either be a Primitive Data Type or a Complex Data Type. In the former case it should be based on a Basic Data Type (e.g. integer, string, etc.). In the latter case, the Data Type may be composed of one or more Primitive Data Types or other Complex Data Types. Each Data Type is expected to have a unit and may have restrictions defined as minimum/maximum values, regular expressions, list of alternative values (i.e. enumerations).

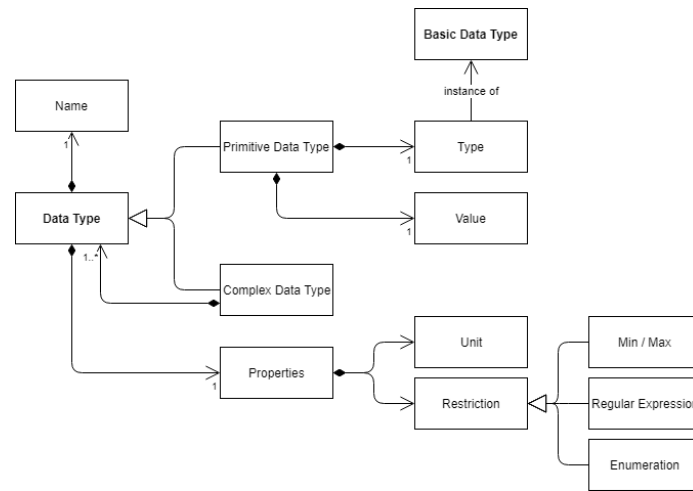


Fig. 3. Data model for “Data Type” entity

4 Implementation of GAMMA Platform

We implemented the GAMMA platform as a web application backed by a server application presenting a RESTful API, written in JavaScript and running on the NodeJS runtime. The API is basically a CRUD (Create, Read, Update, Delete) type of application, keeping the state information in a MongoDB database in the Cloud. The frontend web application runs on any modern web browser. It is implemented as a SPA (Single Page Application) using ReactJS.

The GAMMA front end web application features CRUD operations for the entities defined in Section 3. These entities are Data Types, Feature Types, Data Point Types, Service Types, and Systems. The backend API is divided into 3 types of routes. These routes correspond to the GAMMA system creation phases, which are Design, Development, Engineering, Commissioning, Operation, and Optimization.

In the design phase, Data Types, Feature Types and Data Point Types (DPT) are created and maintained. These models are created and updated by means of the frontend web application and stored in the database. DPT is the conceptual model of a software service relating to a physical entity or a virtual entity depicted in Fig. 2. It is an abstraction of the real world service, which may either serve as a proxy for a physical device

or an orchestrator algorithm between other services. It also has parameters defined to customize its behavior. A DPT corresponding to a physical device present all the features of the device via defined Feature Types and it is vendor, model and communication protocol agnostic.

The listing page and creation and edit pages for Data Types is presented in **Fig. 4** and **Fig. 5**. The pages for Feature Types and DPTs are similar.

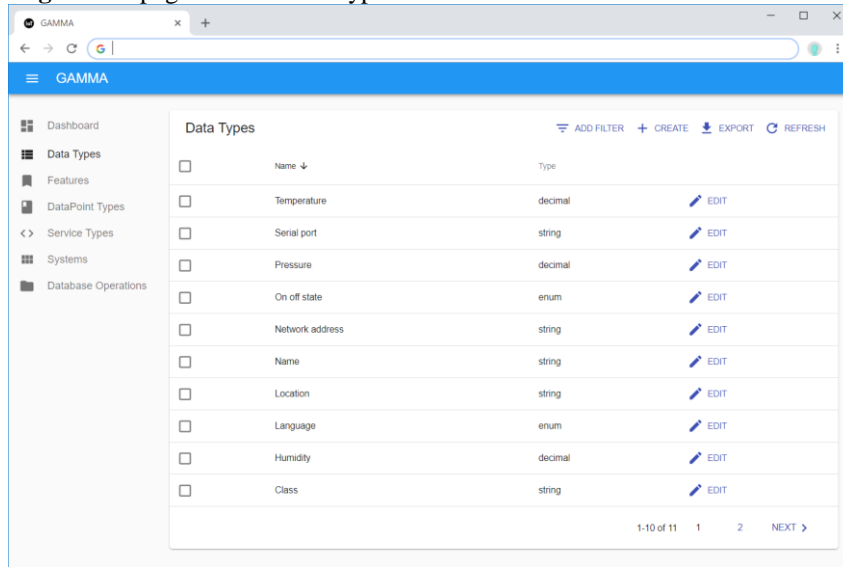


Fig. 4. Data Type listing page

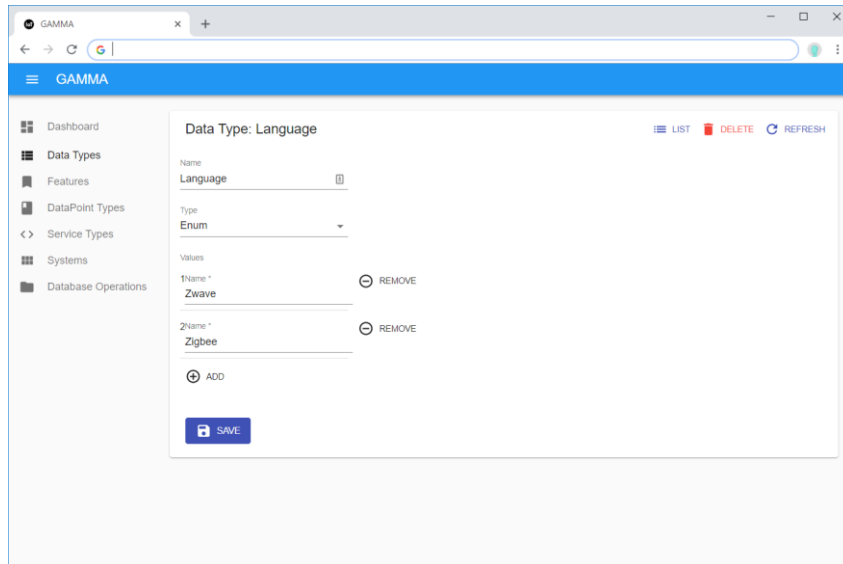


Fig. 5. Data Type creation and edit page

After the design phase, the created DPT is realized as a vendor, model and communication protocol specific Service Type. The parameters defined in the DPT may be assigned to some values in this phase. The Service Type have both a model component and a software implementation. The skeleton code for the software implementation is generated by the GAMMA platform and customized according to the device or algorithm specifications. The generated skeleton code and code customization page is presented in **Fig. 6**.

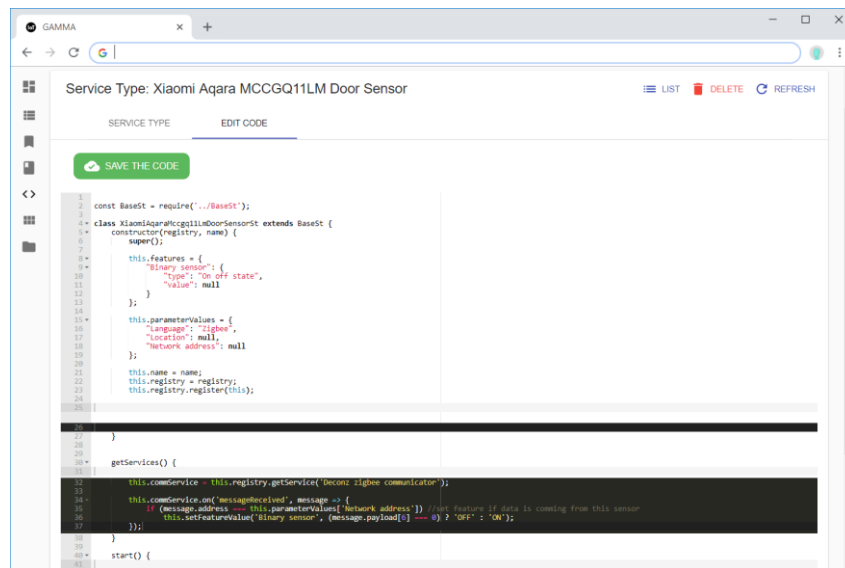


Fig. 6. Service Type generated code and code customization page

After customizing the Service Type code, the code is saved in the database and ready to be used in a system.

There are also communication libraries, that are hardwired into the GAMMA platform, that is, their code is not generated by the user, but included in the GAMMA platform by default. These communication libraries are the gateway services to communication devices (e.g. Dresden Elektronik's ConBee ZigBee dongle) that abstract out the communication details from the services. On top of the communication libraries, there are profile library modules that translate the raw data stream coming from the communication layer into meaningful information to be consumed by device services. One such profile library module is the ZigBee Cluster Library module, that takes data from ZigBee communication service, parses it and presents it to ZigBee device services as meaningful JavaScript objects.

After designing the required entities and developing the Service Types, a system can be created. A system is a model of a real-world working IoT system, that has devices, relations between devices, rules and constraints. Once a system is created, its running code bundle can be downloaded by the web application. The code bundle is an npm (Node Package Manager) project, that has a package.json file. Before running the

system code bundle, all the software dependencies of the npm project must be downloaded by running `npm install` command. Then, the bundle is ready to be run by the command `npm start`.

The basic software design of the system code bundle is presented in **Fig. 7**. In the graph, relations among some of the static classes of the platform (i.e. `BaseSt`, `BaseDeviceSt`, `BaseVirtualSt`, `BaseComm`, `DeconzZigbeeComm`, `OpenZwaveComm`) and some of the developed (by domain and software engineer roles) classes (i.e. `ErsteOrchestratorSt`, `FibaroFgwpeWallPlugSt`, `XiaomiAqaraMccgq11LmDoorSensorSt`).

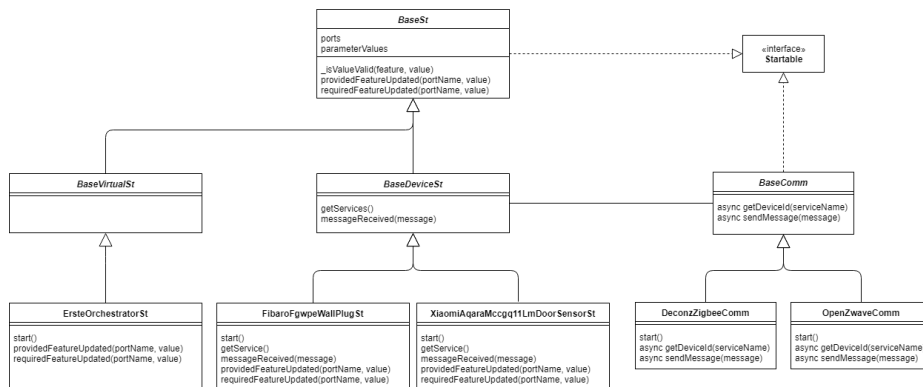


Fig. 7. Class diagram of the system code bundle

5 Related Work and Discussion

Basically, GAMMA is proposed and developed to produce robust IoT systems with reduced effort. In a recent work Giray et al. [5] surveys prominent works on IoT system development methods (SDM) [6-11]. The main difference between the studies presented in [5] and GAMMA is that GAMMA mostly focuses on the development, deployment and operation of the IoT systems. For this purpose, it relies on an engineering toolset and design decisions to guide the development. Although this approach might look like a reduced flexibility in development, it facilitates faster development cycle with robust products developed in majority of the cases. On the other hand, GAMMA does not impose any methodology for requirement analysis and project management. Platform users can choose the best approach suited for their needs.

Furthermore, in [5], an approach to evaluate IoT SDMs is proposed and the studies are compared based on this approach. The IoT SDMs are evaluated based on 14 different criteria. In the following subsections GAMMA, along with the process it is based on, is evaluated with the same criteria.

5.1 Method Artifacts

GAMMA facilitates the creation of the following development artefacts: Domain model (as data types, feature types and data point types), service types (as building blocks of potential IoT systems), IoT system definition, and deployable packages.

5.2 Process Steps

GAMMA is based on the process steps described in Section 2. These steps are design, development, engineering, commissioning, operation, and optimization. GAMMA provides tools for realization of every step of this lifecycle.

5.3 Support for Lifecycle Activities

The main focus of GAMMA includes software development, deployment and operation of IoT systems. Test is currently not part of the platform but it is included in mid term plans. Project management, feasibility analysis and requirements engineering are considered out of scope of the platform.

5.4 Coverage of IoT System Elements

The main purpose of GAMMA is to facilitate modelling of IoT system elements and development of interoperable software components representing IoT elements in the software realm.

5.5 Design Viewpoints

GAMMA partly supports model driven development. Domain engineer models the domain using GAMMA platform. Based on this model, software developer implements the model elements to create executable artefacts. According to requirements and using available artefacts, system engineer designs IoT system and generates the deployable package. All these steps are carried on GAMMA platform, which provides different design views for different stakeholders.

5.6 Stakeholder Concern Coverage

In parallel to the supported lifecycle activities mentioned in Section 5.3, concerns of stakeholders responsible for software development, IoT system design, deployment and operation are supported by GAMMA.

5.7 Metrics

Currently there is no metrics support in GAMMA platform, but in near future, usage information of design artefacts of different stages (e.g. Feature Type, Data Point Type, etc.) in IoT systems will be supported. Hence it will be easier to make impact analysis.

5.8 Addressed Discipline

GAMMA primarily addresses domain engineering, software engineering and system engineering disciplines.

5.9 Scope

Although GAMMA is initially designed and developed with smart buildings in mind, there is no reason not to use it in development of any IoT system.

5.10 Process Paradigm

GAMMA does not impose any process paradigm and therefore open to be used with plan-driven as well as agile paradigms.

5.11 Rigidity of the Method

Since GAMMA and the process it is based on do not address all the steps of the development lifecycle, they are flexible to be extended as necessary.

5.12 Maturity of the Method

The process explained in Section 2 is already validated with research projects. GAMMA itself is still at its relatively early stages. But the initial IoT systems designed using the platform are promising.

5.13 Documentation of the Method

Since GAMMA is a commercial product, publicly available documentation is limited.

5.14 Tool Support

GAMMA platform is an engineering toolset. Therefore, it strongly supports the process described in Section 2.

6 Conclusions and Future Work

This paper briefly introduces the design and implementation of an engineering suite (i.e. GAMMA platform) for smart environment creation based on a development lifecycle targeting smart environments. Core components of the platform is already developed and initial smart environment creation activities show the benefits of the platform. Those benefits include, but not limited to (1) effective cooperation among people with different backgrounds such as domain experts, software developers, system engineers, technicians, operators, etc. (2) streamlined development activities which reduces end-to-end development time (3) systematic reuse of software components with PLE (4) automatic generation of non-creative code which normally consumes time to develop but also error prone.

Development of the GAMMA platform itself continues to increase its efficiency with new or improved features such as better traceability between types, integrated configuration control, built-in support for additional communication protocols, etc. But the next main goal is to focus on virtual entities for smart environment and therefore big data, data mining, and deep learning will be the upcoming key research and development areas.

References

1. Vicari, Norbert, et al. "Engineering and operation made easy-a semantics and service oriented approach to building automation." *Emerging Technologies & Factory Automation (ETFA)*, 2015 IEEE 20th Conference on. IEEE, 2015.
2. "ITEA 3 - Project - 12011 BaaS," ITEA, 2013, itea3.org/project/baas.html.
3. Prieto-Díaz, Rubén. "Domain analysis: An introduction." *ACM SIGSOFT Software Engineering Notes* 15.2 (1990): 47-54.
4. Kang, Kyo C., et al. *Feature-oriented domain analysis (FODA) feasibility study*. No. CMU/SEI-90-TR-21. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.
5. Giray, G., B. Tekinerdogan, and E. Tüzün. "IoT System Development Methods." *Internet of Things*. CRC Press/Taylor & Francis, 2018. 141-159.
6. Slama, Dirk, et al. *Enterprise IoT: Strategies and Best practices for connected products and services*. "O'Reilly Media, Inc.", 2015.
7. Collins, Tom. "A methodology for building the Internet of Things." <http://www.iotmethodology.com> (2017).
8. Patel, Pankesh, and Damien Cassou. "Enabling high-level application development for the Internet of Things." *Journal of Systems and Software* 103 (2015): 62-84.
9. Fortino, Giancarlo, et al. "Towards a development methodology for smart object-oriented IoT systems: a metamodel approach." *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. IEEE, 2015.
10. Ayala, Inmaculada, et al. "A software product line process to develop agents for the iot." *Sensors* 15.7 (2015): 15640-15660.
11. Zambonelli, Franco. "Towards a Discipline of IoT-Oriented Software Engineering." *WOA*. 2016.