GAIA - Green Awareness In Action

D2.1 Initial Infrastructure Software

Document Ref.	
Document Type	Deliverable
Work package	WP2
Author(s)	Federica Paganelli, Giovanni Cuffaro (CNIT), Nelly Leligou, Katerina Papadopoulou (Synelixis), Mariano Leva, Matteo Zaccanignino (OVER), Orestis Akrivopoulos, Nikos Kanakis (SPARK), Georgios Mylonas (CTI), Dimitrios Amaxilatis (CTI)
Contributing Partners	CNIT, SYN, OVER, SPARK, CTI
Dissemination Level	Public
Status	Final version
Version	V1.0
Contractual Due Date	M18 (July 31, 2017)
Actual Delivery Date	August 9, 2017

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 696029.

Disclaimer

Document Revision History

Date	Issue	Author/editor/contributor	Summary
30/05/2017	0.1	Federica Paganelli, Giovanni Cuffaro	Document structure
03/07/2017	0.2	Federica Paganelli, Giovanni Cuffaro, Nelly Leligou, Mariano Leva, Matteo Zaccagnino, Orestis Akrivopoulos, Nikos Kanakis Georgios Mylonas, Dimitrios Amaxilatis	First Draft
20/07/2017	0.3	Federica Paganelli, Giovanni Cuffaro, Nelly Leligou, Mariano Leva, Matteo Zaccagnino, Orestis Akrivopoulos, Nikos Kanakis Georgios Mylonas, Dimitrios Amaxilatis	Second Draft
30/07/2017	0.4	Federica Paganelli, Giovanni Cuffaro	Third Draft
31/07/2017	0.6	Georgios Mylonas	Fourth Draft
01/08/2017	0.7	Georgios Mylonas, Federica Paganelli	Fifth Draft
04/08/2017	0.8	Matteo Zaccagnino, Nikos Kanakis, Federica Paganelli, Giovanni Cuffaro	Sixth Draft
09/08/2017	1.0	Georgios Mylonas, Dimitrios Amaxilatis	Final version

Abbreviations

Abbreviation	Expression
AA	Authentication & Authorization
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AS	Authorization Server
CRUD	Create Read Update Delete
DB	Database
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
URI	Uniform Resource Identifier
WP	Work Package

Executive summary

This deliverable describes the design and implementation of the *GAIA Service Platform*, which represents essentially the back-end of the GAIA ecosystem. The main goal of the GAIA Service Platform consists in acquiring data from the heterogeneous sensor infrastructures deployed in the schools' premises in Greece, Italy and Sweden, store and process these data to derive meaningful information for energy consumption awareness and reduction (i.e., analytics and energy-saving recommendations), and, finally, provide GAIA end-user applications with a unified access mechanism to such data.

This document builds upon and extends considerably the interim report produced by the GAIA consortium regarding WP2 at Month 12 (February 2017) of the project. In this document, we first discuss the main requirements that have driven the design and implementation activities in this work package. We then introduce the logical architecture of the GAIA Service Platform, identifying its main functional blocks. Each functional block is then described in more detail, by first introducing technical requirements, and then providing design specifications and implementation details. This discussion is included in Sections 2 to 8 of this document.

A "drop-in" sensor platform, for easing the gathering of environmental sensor parameters, has also been developed in order to support the GAIA activities. This solution complements the initial approach used in the majority of GAIA's school buildings and is described in Section 9 of this document.

Finally, in Section 10, the interaction among WP2 and WP3 components is described in some relevant GAIA processes. These 2 work packages are tightly interrelated: WP3, as also described in Deliverable D3.2 "Applications Initial Release", consumes the data produced by the GAIA IoT infrastructure and offered as a service by the software described in this document. In this sense, we have designed and implemented a service platform that caters to the needs and requirements tied to the GAIA WP3 applications. However, this service platform has been implemented following current software design and implementation practices, and can be used to produce additional applications on top of it, and is not strictly tied to GAIA's applications. In this sense, this document comes with the documentation of the APIs exposed by the various WP2 software components to make services accessible to internal WP2 components, as well as to third-party software, including, for instance, WP3 applications. The API documentation is available as a PDF document (Attachment A), as well as online services.

We conclude the document with a discussion on future work towards the final version of the software, which will be described in Deliverable D2.2 "Final Infrastructure Software", made available in Month 24 of the project.

Table of Contents

1 Introduction

1.1 Scope of deliverable

This deliverable describes the work conducted by the GAIA Consortium in WP2, by reporting the progress of Tasks 2.1, 2.2 and 2.3.

The goal of WP2 is to support the GAIA mission by providing an intermediate layer (middleware) between sensors' infrastructures and GAIA end-users applications. This middleware is called the *GAIA Service Platform*. The main goals of the GAIA Service Platform consist in: i) acquiring data from the heterogeneous sensor infrastructures deployed in the schools' premises in Greece, Italy and Sweden, ii) storing and processing these data to derive meaningful information for energy consumption awareness and reduction (i.e. analytics and energy-saving recommendations), and iii) providing GAIA end-user applications with a unified access mechanism to such data.

This document describes:

- The main requirements and design principles that have driven the design and implementation activities.
- The logical architecture of the GAIA Service Platform, identifying main functional blocks.
- The design and implementation of each main functional block.
- The architecture of drop-in sensor devices elaborated for GAIA purposes
- Interaction among WP2 and WP3 components in some relevant GAIA processes.
- Main objectives on future work towards the final version of the software.

Services provided by WP2 software are mostly available through REST APIs. The respective documentation of these APIs is made available as a PDF document (Attachment A), as well as online services (see Section 2.3).

1.2 Relation to other deliverables

The design of WP2 components is based on the concepts elaborated in D1.1 - *GAIA Design* [GAIA1.1.], which discuss GAIA philosophy, reference scenarios and main system requirements. D2.1 takes also into account the description of WP3 Applications design and implementation provided in D3.1 - Application Prototypes and D3.2 – Applications Initial Release [GAIA3.1, GAIA3.2]. With respect to WP2 Report delivered in M12, D2.1 provides the specification of the GAIA Service Platform, in a version which has been consolidated after the first trials in some pilot schools (see Deliverable D4.1 - "Initial Trial Documentation").

1.3 Role of the Service Platform in GAIA

As stated in deliverable D1.1, "*GAIA aims at improving energy efficiency by increasing awareness of specific target groups, all related to the educational process and community. In order to achieve this goal, we will utilize the infrastructure available at schools and enrich it towards gathering information. This information is used by a set of applications to guide energy efficient behavior which is assessed through the continuous monitoring of building energy consumption*."

Such a goal will be achieved by leveraging a set of technologies, depicted in Figure 1 as the "GAIA Ecosystem", organized in the following three main layers:

- *IoT Infrastructure*, which includes the sensing and power metering infrastructure deployed at each school building to gather monitoring data (as described in [GAIA1.1]).
- The *GAIA Service Platform*, which is a backend system that provides capabilities for storing and analysing data regarding the energy consumption of buildings and users' behaviour and for disseminating this information to end-user applications (in the scope of this WP).
- The *Applications layer*, which is composed by end-user applications targeting human behaviour change towards energy efficiency (in the scope of WP3 [GAIA3.1, GAIA3.2]).

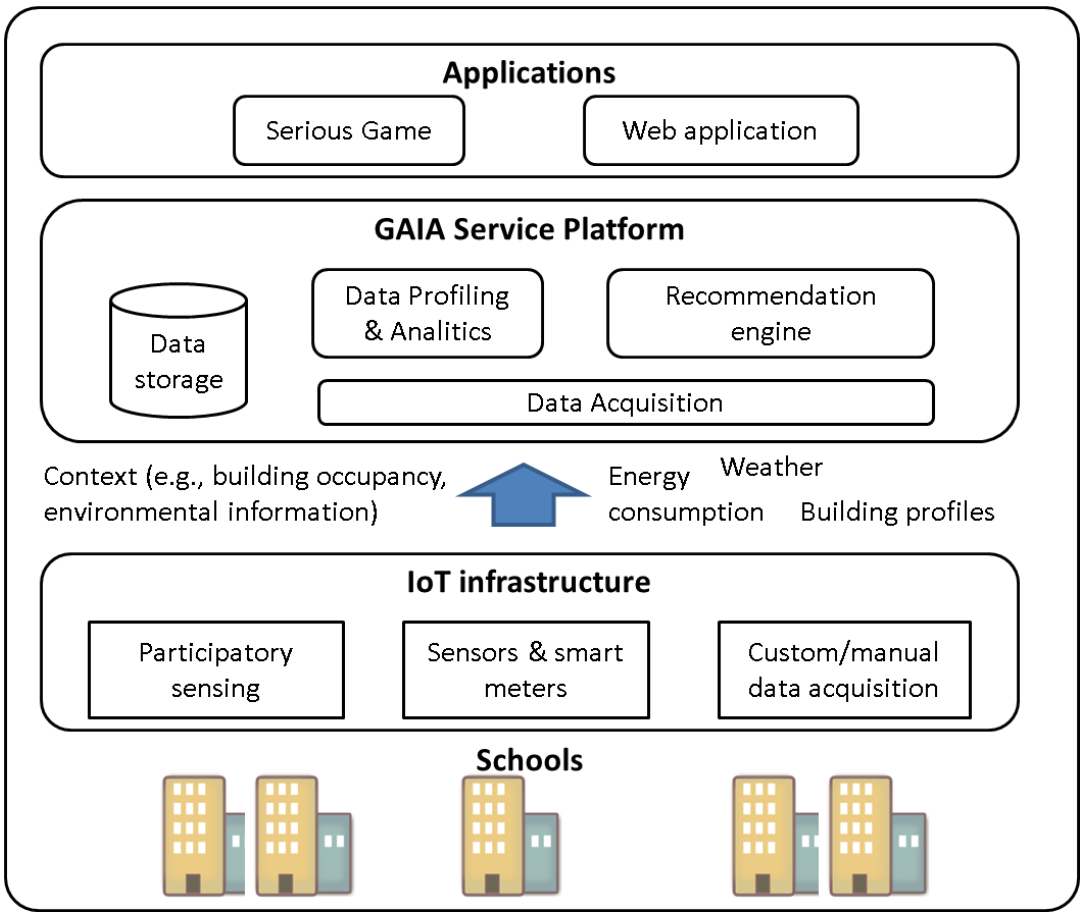

Figure 1 The GAIA Service Platform and its interface with the GAIA ecosystem

A key principle of GAIA is fostering education and experimentation activities leveraging the *availability of actual measurements of environmental parameters, such as energy consumption, indoor and outdoor luminosity, temperature, noise, etc.* The GAIA Service Platform allows to feed end-user

applications with such monitoring data in order to enhance educational processes with data from real-life scenarios.

On the one hand, this is a key characteristic of the GAIA-way of educating students to energy-efficiency, on the other side this approach poses some technical challenges since it will be experimented in a set of schools, distributed in different geographical locations and characterized by different profiles (education level, curricula, management structure, technical building characteristics, etc.). Challenges are mainly due to the fact that the sensing infrastructure available or purposely deployed to gather GAIA-related measurements may vary from school to school (e.g., number and type of sensors, sensor devices, communication protocols in the sensor network, gateway technology and protocols for delivering sensor data to external systems, etc.), since schools may be characterized by different energy consumption needs, management choices, control and operation technologies, regulation.

In order to cope with this issue, the GAIA Service Platform hides the heterogeneity of sensors deployment to end-user applications by managing the communication with the sensor infrastructures through dedicated software modules, handling and storing data according to a uniform model and offering such data to applications through uniform interfaces. This means that if a new school is added to the system or GAIA has to be applied in a different group of schools, GAIA applications can easily access sensors' data with only some minor reconfiguration.

Moreover, measurements provided by sensors may need some processing to become meaningful and usable by end-user applications. Therefore, *filtering, reduction, and formatting operations* are performed on raw data flows acquired by networks of sensors to transform them in information that is suitable for storage and higher level processing. More in detail, formatting operations are applied to convert heterogeneous data obtained from different sensors into a uniform naming convention. In this way, applications and higher-level processing services can seamlessly access data gathered from different sources.

The GAIA Service Platform will provide a set of *data storage and processing services* for extracting, persisting and disseminating information related to energy-efficiency and resource optimization that are derived from measurements collected from sensors or by end-users through participatory sensing. The storage service provided by GAIA is the key to transform *data in motion* (i.e. data flows generated by sensors) to *data at rest*, i.e. data from readily accessible storage facilities [Cisco, 2014]. Indeed, it is widely recognized that in the IoT domain, most end-user applications need to access data on a non-real time basis [Stallings, 2015]

The data storage service provided by GAIA also offers APIs to allow end-users (using applications that leverage such APIs) to access, provide and manage information regarding the profile, the topology and other relevant information characterizing the school buildings and their usage (e.g., geographical location, organization of spaces, number of students, opening and closing times).

These data can then be accessed by modules that perform high-level processing tasks (i.e. *analytics* and *recommendation engine*) for extracting further meaningful and useful information that could help

users (especially building managers) in:

- visualizing and understanding the energy-consumption profile of a building;
- being aware of possible anomalies in energy consumption;
- receiving suggestions for actions and behavior-based changes possibly leading to energy reductions while maintaining the comfort of the learning environment;
- managing and customizing suggestions notified at the occurrence of events of interests related to energy consumption (e.g., list of equipment to be checked and eventually turned off before holiday periods).

In addition, the GAIA Service Platform also provides support to participatory sensing approaches [Estrin, 2010] by offering APIs allowing end-users applications to manually provide consumption data (e.g., via csv and MS excel files) or web forms.

In addition, the GAIA Service Platform also provide end-users applications with authentication and authorization services, leveraging existing standards and offering basic profile storage services. Indeed, GAIA end-user applications need to differentiate permissions according to user's roles (e.g., student, teacher and building manager).

1.1 Overall structure of this document

The rest of this document is organized in the following chapters:

Section 2 describes the logical architecture of the platform, distinguishing main functional blocks, discussing main design requirements and positioning the GAIA Service Platform within the IoT World Forum Reference Model. A conceptual model of the entities handled by the platform is also provide. Subsequent sections describe in detail the GAIA service platform main functional blocks.

Section 3 describes the Authentication and Authorization Infrastructure. **Sections 4 and 5** introduce the Data Acquisition and Data Storage system. **Section 6** describes the Building Knowledge Base and **Section 7** provides details on the Analytics module. **Section 8** describes the Recommendation Engine.

Section 9 described the drop-in sensor device platforms that have been designed and implemented in GAIA for supporting trial activities.

Section 10 provides some examples, which show how WP2 and WP3 components cooperate in supporting a number of main GAIA processes.

2 Logical Architecture

This section presents a logical view of the GAIA service platform. First we introduce the main general requirements and the main design choices. Then, we describe the modular architecture of the platform, by identifying the main functional blocks and positioning such architecture with respect to a reference model for IoT service architecture, i.e., the IoT world Forum Reference Model [Cisco, 2014].

2.1 General Requirements

We present here the main challenging non-functional requirements taken into account for the design and implementation of the GAIA Service Platform.

Interoperability with heterogeneous sensing platforms. The GAIA Service Platform should acquire data from multiple geographical sites and different technological platforms. The technologies deployed in the trials sites have been described in D1.1 [GAIA1.1]. Different sensing infrastructures and communication protocols have been used across the trial sites in Greece, Italy and Sweden. Therefore, acquired raw data are accessible through heterogeneous APIs and naming conventions.

Uniform interface for accessing building's measurement data and information. Measurements data acquired from the trial sites should be stored and then made accessible to end-users' applications through a uniform interface, hiding differences of the underlying technological infrastructures. Applications should thus have a unified global view of available data, in terms of a unified convention for naming resources and a unified mechanism for accessing them.

Modularity and loose coupling. The GAIA Service Platform Components should be independent of one another and communicate with each other in a loosely coupled fashion. This means that components can be modified/replaced without changing the other components, given that the exposed interface remains unchanged.

Scalable and iterative development. The platform design and development should be carried out by multiple teams of different consortium partners in different locations. An iterative and incremental development methodology is needed in order to ease the integration with end-user applications.

Support of existing standards. It is desirable to support existing technological standards, in order to promote interoperability with third-party systems and promote the reuse of other standard-compliant tools and systems.

Open source approach. The use of open standards and systems is a common GAIA design principle, due to its benefits in terms of costs, reliability, community support and interoperability.

Security and Privacy protection. The GAIA Service platform will acquire, store and disseminate data related to school sites and user accounts. State of the art security and privacy protection technologies

will be put in place to fully address security and privacy requirements and regulations.

2.2 Main Design choices

In order to cope with above-mentioned non-functional requirements, the design and implementation of the GAIA Service Platform is inspired to the REST architectural style and its novel declination into the Microservice architecture. First, we provide a brief description of the REST architectural style and then we describe the *Microservice architecture* and how the GAIA Service Platform relates with the FIWARE Service Architecture.

2.2.1 REST architectural style

The REST architectural style was proposed by Fielding [Fielding, 2000] in his doctoral dissertation as an architectural style for building large-scale distributed hypermedia systems. On the REST vision, data sets and objects handled by client-server application logic are modelled as resources.

The key principles of REST are five-fold [Fielding, 2000]:

1. *URIs as resource identifiers*. Resources are exposed by servers through Uniform Resource Identifiers (URIs). Since URIs belong to a global addressing space, resources identified with URIs have a global scope;
2. *Uniform interface*. The interaction with the resource is fully expressed with four primitives, i.e., create, read, update and delete;
3. *Self-descriptive messages*. Each message contains the information required for its management (metadata are used for content negotiation and errors notification);
4. *Stateless interactions*. Each request from client to server must contain the information required to fully understand the request, independently of any request that may have preceded it;
5. *Hypermedia As The Engine Of Application State (HATEOAS)*. A hypermedia system is characterized by participants transferring resource representations that contain links; the client can progress to the next step in the interaction by choosing one of these links [Richardson & Ruby, 2007].

REST is currently considered a best practice for the design of web service interfaces, thanks to its advantages in terms of scalability, interoperability and simplicity [Fielding & Taylor, 2002; Zuzac et al., 2011]. The REST architectural style most diffused implementation is the HTTP protocol and most REST APIs are indeed HTTP-based. However, this causes some limitations in the use of REST in Internet of Things application scenario, since HTTP does not provide efficient mechanisms for communication persistence and asynchronous notification.

To cope with these limitations, the GAIA service platform APIs will be based on the following set of technologies: REST HTTP APIs, AMQP and WebSocket as well [Collina et al., 2012; Fernandes et al., 2013].

2.2.2 Microservice architecture

The Microservice architecture is an approach to develop a single application as a set of small services each running on its own process and communicating each other with lightweight protocols, e.g., HTTP Rest APIs [Fowler, 2016]. In a Microservice architecture, the primary way of componentizing software is by breaking it down into services, i.e., components who communicate with a mechanism such as a web service request, or remote procedure call. This approach shows many benefits in terms of software lifecycle management and scalable development and deployment, since if an application is decomposed into multiple services, most single service changes would only require that service to be redeployed. Of course, changes that affect service interfaces would require some coordination, but the aim of a good microservice architecture is to minimize these through cohesive service boundaries and evolution mechanisms in the service contracts.

Another consequence of using services as components is a more explicit component interface thanks to the explicit use of remote call mechanisms. Key benefits of microservice architecture are listed hereafter [Newman, 2015].

Technology Heterogeneity. With a system composed of multiple, collaborating services, it is possible to use different technologies. This allows to pick the most appropriate tool for each job, rather than having to select a more standardized, one-size-fits-all approach that often ends up being the lowest common denominator.

Organizational Alignment and scalable development. Microservices allow to better align the system architecture to organization principles, by minimizing the number of people working on any codebase. This eases the collaboration on a distributed team of developers across both geographical and administrative boundaries (which is the case of the GAIA consortium development team).

Ease and scalability of deployment Microservice architecture allows to deploy (and re-deploy) a single service and deploy it independently of the rest of the system. Moreover, this also facilitates the implementation of customized and cost-effective scalability policies, since it is possible to just scale those services that need scaling, while other parts of the system can be run on less powerful and economic hardware.

Composability and reuse. Microservices implement and expose well-defined and independent features, thus promoting reusability and composability of implemented features to cope with different goals.

Of course, the Microservice architecture also has some drawbacks. Indeed, the complexity is not eliminated, rather it is shifted around to the interconnections between services. This can lead to increased operational complexity, such as difficulties in debugging behaviour that spans services and the need of some orchestration capabilities. Wise choices of appropriate service boundaries might help in delimiting these drawbacks.

The GAIA Service Platform will be thus composed of microservices that can be developed

independently one from another, where each module provides a solid boundary allowing to be implemented and managed by different teams even using different programming languages. Thanks to this approach, each GAIA partner involved in software implementation will develop one or more components that are almost independently replaceable and upgradable, supporting iterative and scalable development and decentralizing decisions about conceptual models and persistence decisions. Of course, service contracts will be the result of strict collaboration across teams. As mentioned above, this approach will allow us also to deploy different services in different locations, to replicate the most critical components and to scale up only the compute-intensive services saving resources. The modules will be deployed on a cloud platform infrastructure (e.g., Microsoft Azure).

The different components of the GAIA platform communicate each other and with third party applications through REST APIs. Resources in GAIA are thus identified by a URI (uniform resource identifier) and can be accessed and modified by HTTP methods like GET, POST, PUT and DELETE.

2.2.3 Services Infrastructure

FIWARE offers a set of key enablers for IoT systems as well as infrastructures to develop prototype implementations of applications that can potentially target global challenges [FIWARE]. Its main mission is to build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors. GAIA project partners have used a number of its tools and architectural ideas for developing the first iterations of their tools (i.e., the FIWARE context broker [FIWARE Context Broker] and the FIWARE Lab [FIWARE Lab]). Based on this experiences, we were able to expand and build our tools to support the project's requirements. Since then, the project has switched some of the used solutions to more production ready systems to provide better stability and support for more devices and users while remaining close to the same principles and ideas provided by FIWARE.

2.3 GAIA Service Platform architecture

This Section provides an overview of the logical architecture of the GAIA Service Platform, describing the features offered by main functional blocks and interactions. Figure 2 shows the main functional blocks of the platform:

- Authentication and Authorization Infrastructure
- Acquisition
- Storage
- Analytics
- Recommendations
- Building knowledge base

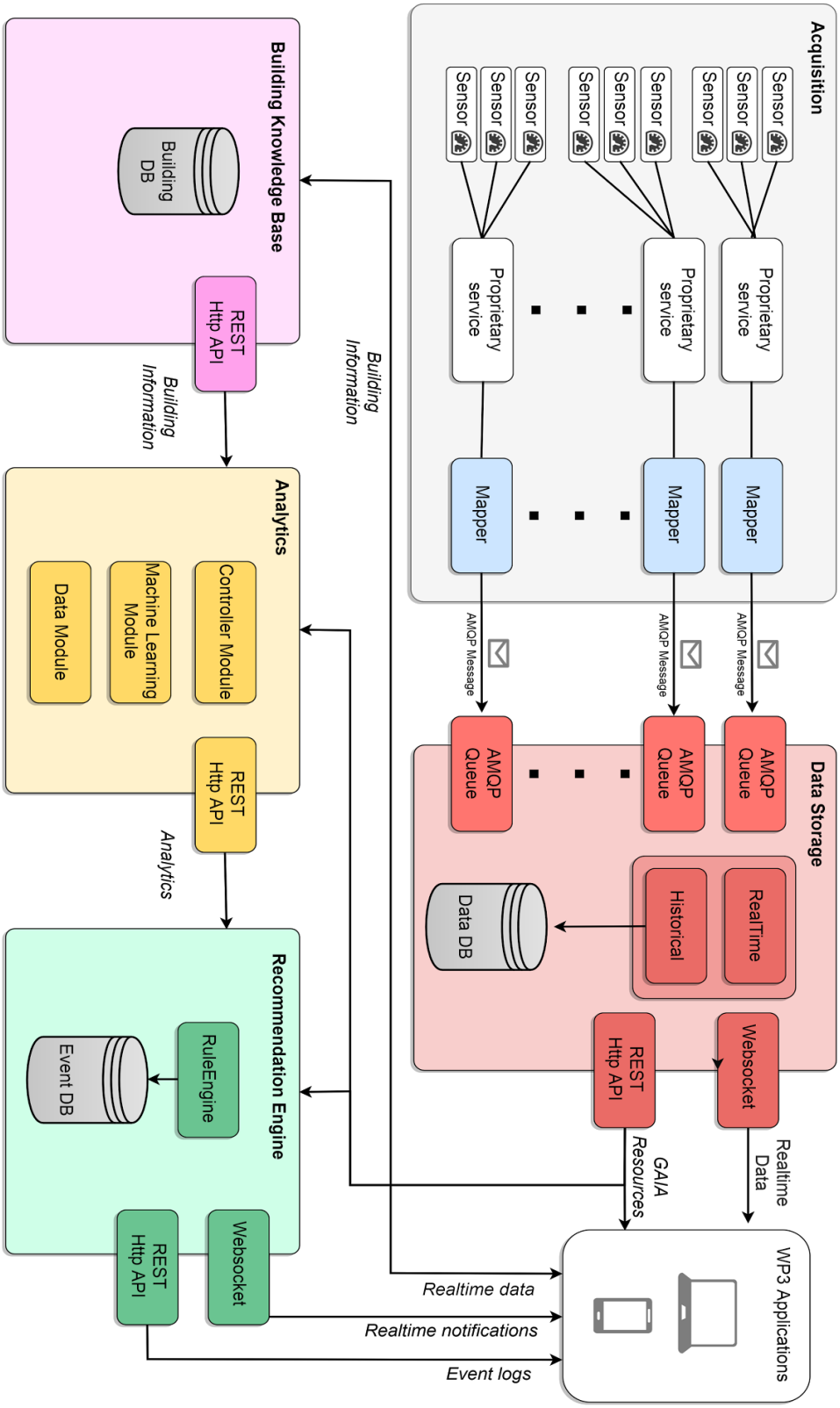

Figure 2 GAIA Service Platform logical architecture

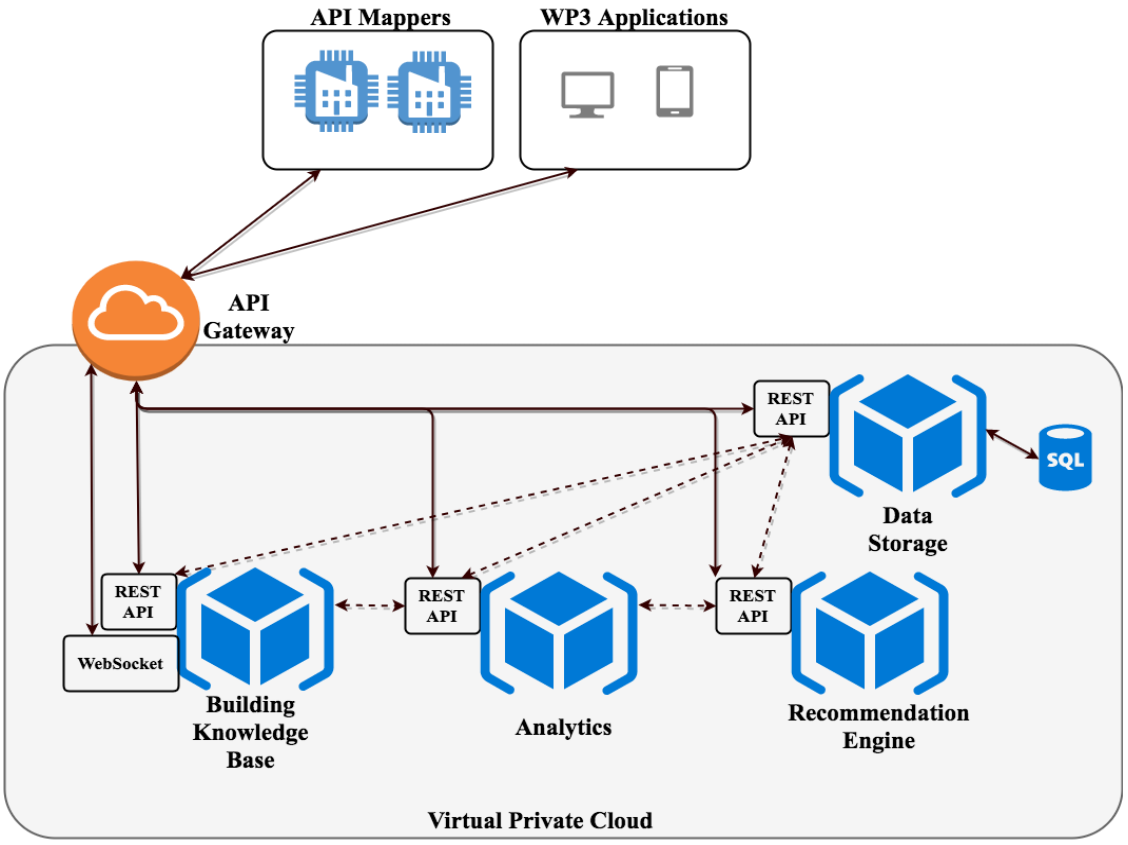

Figure 3 GAIA Microservices Architecture

Figure 3 provides an Interaction view of the architecture, highlighting the architectural style based on microservices and communication among components and the role of the IoT Gateway for this purpose. These functional blocks are briefly introduced hereafter and the main ones are described in the following sections.

Component	**Authentication and Authorization Infrastructure**
Description	It provides authentication, authorization and role management services to GAIA software components (WP2 and WP3) for secured access. It is based on the OAuth 2.0 authorization protocol [OAuth2]. It hosts the user accounts, and authorizes other services to access protected resources according to the user account profile. It provides authorization flows for web and desktop applications, browser-based applications and mobile applications.
Provided features	- create and manage user, user profiles
Relation to other components	OUT -> - authentication and/or authorization outcome
API Documentation	URL: https://swaggerhub.com/apis/kanakisn/SparkWorksAA/1.0.0

Component	API Gateway
Description	The granularity of the GAIA Service Platform APIs provided by the internal microservices is often different than a public client needs and the platform provides fine grained public APIs, hence a client needs to interact with multiple microservices to complete an operation. To cope with this, an API Gateway is introduced. which realizes the API Gateway pattern [Microservices Patterns, 2017]. It provides a single entry point for all clients either just proxying a request to the appropriate internal microservice or fanning out a request to multiple microservices if applicable.
Provided features	- implements the API Gateway Pattern, proxying/routing a public request to the appropriate microservice(s)
Relation to other components	OUT -> the proxied request to the appropriate microservice
Documentation	URL: https://api.sparkworks.net/swagger-ui.html

Component	Acquisition
Description	It deals with the heterogeneity of the sensors and API exposed by the different proprietary platforms with the aim of making data uniform before sending them to the GAIA infrastructure. The block is composed by a set of “API Mappers”, one for each proprietary platform, acquiring (pull) or receiving (push) data and, after the needed transformation for converting acquired data into the GAIA naming and data structure convention, sending a message in a queue through the Advanced Message Queuing Protocol (AMQP), to be consumed by the Data Storage block.
Provided features	- gather measurements from sensors, insert them into appropriately formatted messages, and send this message to the Ingestion block through a message-queue based technology (AMQP)
Relation to other components	OUT -> - provide Data Storage with measurement Data
Documentation	URL: https://github.com/SparkWorksnet/device-mapper-template/blob/master/README.md

Component	Data Storage

Description	It receives uniformly formatted messages from the API Mappers. These messages contain the value of the measurement, the URI of the observed resource and the timestamp. This module deals with both the storage of the data in a database supporting time-range queries and the computation of real time statistics of the measurements (e.g., average, maximum, minimum). Data are exposed via HTTP REST APIs (time range, latest value, and summary), as well as via WebSocket (STOMP protocol) for pushing real-time measurements to the subscribed clients (e.g., the building manager application)
Provided features	- store data related to buildings and measurements - allow access to this data through a resource-based model
Relation to other components	IN <- - read measurement Data from message queues (typically fed by API Mappers) - read data provided by end-users through participatory-sensing enabled applications (WP3) OUT ->: - Provide uniformly- accessible resource data to the Building Knowledge Base, Analytics and the Recommendation Engine, as well as to external clients (i.e., WP3 Applications).
API Documentation	URL: https://api.sparkworks.net/swagger-ui.html

Component	**Building Knowledge Base**
Description	It stores all the useful information about the school buildings (e.g., maximum number of hosted people, area, volume, schedules, type of heating and cooling systems etc.). It exposes REST API providing all the methods to get/edit/delete data resources.
Provided features	- store school building information - allow applications to access and modify these data through REST APIs
Relation to other components	IN <- - receive building information from the Building Manager System UI (BMS UI) OUT ->: - Provide building information to Analytics and the Recommendation Engine (WP2), as well as to WP3 Applications and other authorized clients.

API Documentation	URL: https://buildings.gaia-project.eu/gaia-building-knowledge-base/swagger-ui.html

Component	**Analytics**
Description	It processes data retrieved from the Data Storage and the Building Knowledge Base and exposes on-demand aggregation- and statistical-derived information through REST APIs. It also detects a set of basic patterns of the observed parameters behaviour in order to provide more complex data analysis like anomaly detection and clustering.
Provided features	- query measurement data statistics over different temporal intervals - cluster buildings - detect and store power consumption anomalies
Relation to other components	IN <- - receive input data from the Data Storage Block - receive input data from the Building Knowledge Base OUT -> - Provide information to the BMS application to be visualized by end-users, as well as to any other authorized client.
API Documentation	URL: https://analytics.gaia-project.eu/gaia-analytics/swagger-ui.html

Component	**Recommendation**
Description	It generates recommendations for promoting energy saving behaviours, based on the analysis of data made available by the Data Storage and Analytics blocks. Its design and implementation is based on the use of a rule engine which checks if a set of conditions are verified to trigger some recommendation actions. Conditions may range from a simple comparison between a measurement and a threshold to more complex rules based on the use of multiple logical operators and the use of more complex and meaningful information provided by the Building Knowledge Base and the Analytics blocks. Generated recommendations will be consumed by end-user applications, in the form of real time notifications and logs reporting on the occurrence of specific conditions and rule firing events (e.g., useful for report generation by end-user applications). Real time notifications are accessible via a WebSocket and can be used by the Building Manager System UI (BMS) to alert the

	building manager and suggest possible actions, while the event log is exposed through simple HTTP REST API.
Provided features	- create and modify rules for a building and/or a building area - generate and disseminate recommendations - log the occurrence of events which can be of interest for energy-saving purposes
Relation to other components	IN <- - receive input data from the Data Storage, Building Knowledge base OUT ->: - Provide recommendations and event reports to applications (i.e. WP3 BMS UI)
API Documentation	URL: https://recommendations.gaia-project.eu/docs/index.html Developer's manual available at the GAIA website http://gaia-project.eu (Resources menu)

2.4 GAIA Service Platform Architecture and the IoT World Forum Reference Model

In the following, we briefly describe the IoT World Forum Reference Model, in order to clearly characterize and position the GAIA Service Platform, with respect to this model.

The IoT World Forum is an event that annually brings together stakeholders from the industry, government and research domains to discuss and promote adoption of IoT technologies [Stallings, 2015]. In this context, a group of industries (e.g., IBM, Intel and Cisco) proposed in 2014 an IoT reference model, with the aim of providing guidelines for accelerating IoT deployments, promoting replicability and collaboration (Figure 4).

We refer to the Cisco White Paper [Cisco, 2014] for the detailed introduction of the seven-layered architecture of the IoT reference model, and hereafter we discuss how the GAIA Service Platform provides features pertaining to four levels of the IoT World model, namely *Edge Computing*, *Data Accumulation*, *Data Translation*, and *Application*.

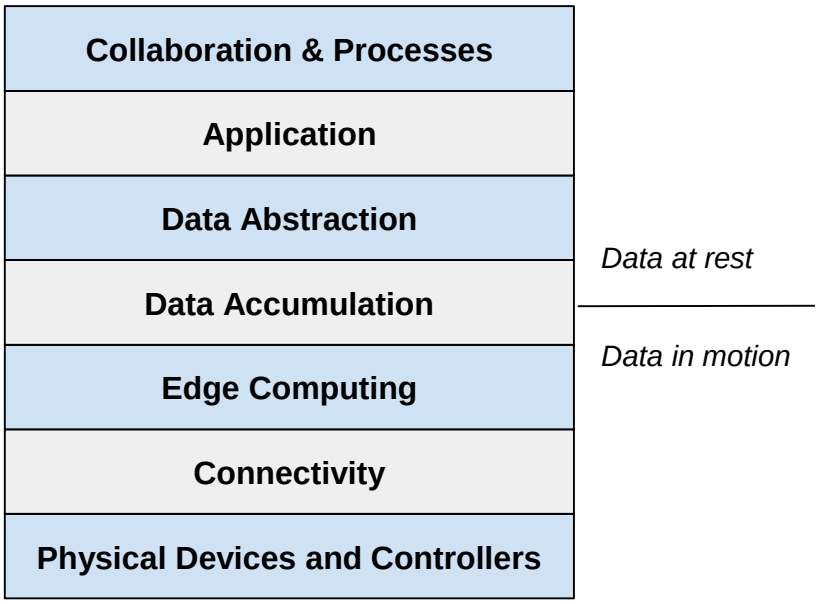

Figure 4 IoT World Forum Reference Model [Cisco, 2014]

Edge Computing

This level should convert network data flows into information that is suitable for storage and higher level processing. For instance, evaluation, formatting, reduction operations can be performed on data generated by a distributed network of sensors.

The *API Mapper* components in the *Data Acquisition* Block are software components that may be appropriately customized to perform such operations. Moreover, the *API Mappers* are components external to the GAIA Service Platform, which may be deployed in the more appropriate location, according to different criteria (e.g., distance to sensor networks, resource availability and openness of IoT gateways, etc.).

Data Accumulation

At this level, "data in motion" are converted into "data at rest". This means that this level is responsible for organizing and storing data. Some data might be combined and aggregated with previously stored information, including data coming from non-IoT sources. This level provides applications with query-based access to data, instead of a real-time and stream-based one. The Data Storage block actually performs such kind of operation, by performing processing and storage operations on data received through message queues and making them available through REST APIs, while also allowing application to receive real-time data through WebSocket channels.

Data Abstraction

The data abstraction functions are focused on rendering data and its storage in ways that enable developing simpler, performance-enhanced and secured applications. At this level, several features may be implemented, such as consolidating data into one place or providing access to multiple data stores through data virtualization, protecting data with appropriate authentication and authorization, indexing data to provide fast application access, etc.

In GAIA, deployed sensors produce (periodically or asynchronously) events that are sent to Data

Accumulation level via the Platform Message Broker. Heterogeneous external services are capable of publishing the IoT data (generated or gathered) to an MQTT endpoint. The Data Abstraction layer is then able to receive new measurements asynchronously and format them to the internal format of the platform. IoT data are then forwarded to the Data Accumulation engine for processing and analysis using the AMQP protocol. IoT data are then processed and can be accessed from the Application API. Finally, the Authentication and Authorization block provides secured access to GAIA data sets and services.

Application
This is where information interpretation occurs. The IoT Reference Model does not strictly define the scope of applications, while it highlights how well-designed underlying levels (up to the Abstraction one) may greatly simplify the design of applications. The Analytics and Recommendation engine may be considered as application software in that it interprets available information and produces new application-specific knowledge.

2.5 GAIA Conceptual Model

In this section we describe the GAIA conceptual model, which represents the main entities handled by the GAIA Service Platform and offered to external clients. To define the base entities model we followed the Semantic Sensor Network ontology [SSN Ontology]. The SSN ontology can describe sensors in terms of capabilities, measurement processes, observations and deployments. A preliminary version of the GAIA base entities is provided in the conceptual model in Figure 5. The base entities described in the conceptual model are: the *Resource*, the *Site*, the *SiteInfo*, the *SiteSchedule* and the *Measurement*.

The *Site* entity represents a specific area, such as a building or a room, following the composite data model. The *Resource* entity represents an abstract data source. For instance, the most straightforward example of a *Resource* is a sensing device deployed on a *Site* while at the same time the entity holding participatory sensing information is represented as a *Resource* too. Finally, the *Measurement* entity represents a specific instant of Resource measurements.

The *Rule* entity represents a logical rule whose internal behavior is to evaluate a condition and trigger an action. It contains the required information to support its execution (e.g., the textual suggestion to be sent to the user and some custom fields such as thresholds to be applied to the measurements).

The *Site* Entity can have some entities associated that contains some descriptive information of the building/area, such as the schedule (opening and closing time, lectures, etc.) (*SiteSchedule*), information about the heating and cooling system, as well as general information about the site, such as description and square meters (*SiteInfo*).

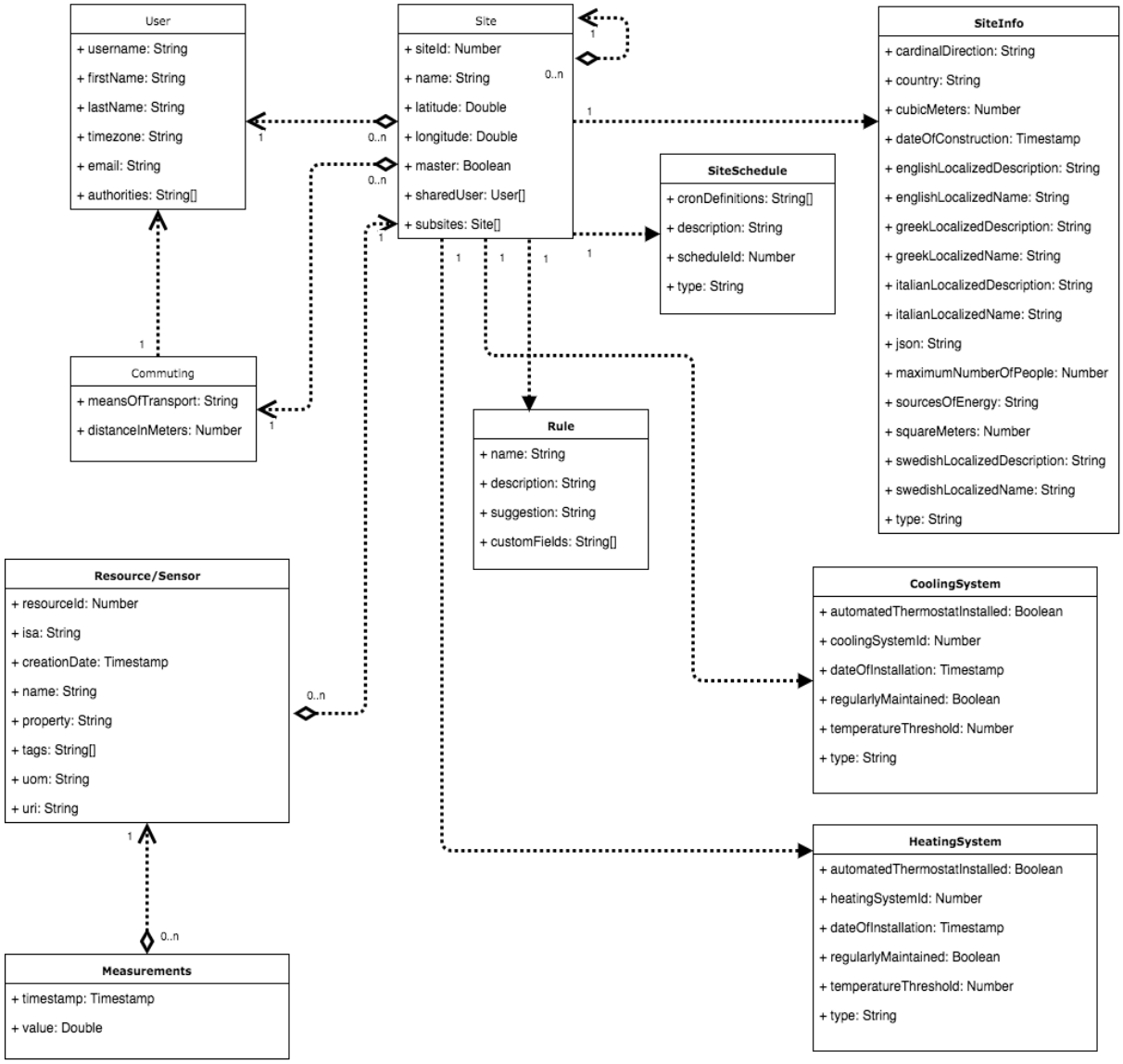

Figure 5 GAIA Conceptual Model

2.6 Guidelines for the description of functional blocks

In the following subsections, each block of the logical architecture will be described in terms of requirements specification, design and implementation details and exposed APIs. The following reference section structure will be adopted in the following section and appropriately slightly adapted to the specific characteristics of each block:

- *Requirements*: requirements will be specified by providing a Code, a Description text, and Priority information.
- *Design*: design of each functional block will be documented through a set of main class and sequence diagrams and corresponding descriptive content
- *APIs*: Interfaces will be described in terms of adopted protocols, endpoints and exposed resources (naming conventions and operations allowed on resources)
- *Implementation details*: implemented software will be described in terms of adopted

technologies, implementation status, etc.

As regards requirements specification, the following keywords are used throughout the section to better specify the relevance of the requirement within the GAIA Project:

- "MUST" means that the item is an absolute requirement;

- "SHOULD" means that there may exist valid reasons not to treat this item as a requirement, but the full implications should be understood and the case carefully weighed before discarding this item;

- "MAY" means that an item deserves attention, but further study is needed to determine whether the item should be treated as a requirement.

System-level requirements are further assigned a "priority" value chosen according to the "HIGH, MEDIUM, LOW" ordinal scale. The meaning of each rating is specified in the following schema:

- "**High"** means that satisfaction of the requirement is essential for the architecture and that the architecture cannot be designed without taking it into consideration. Also first-time implementations of the architecture will need to provide the designated capability in order to be considered as "valid";
- "**Medium"** means that the architecture cannot be designed without taking the requirement into consideration, but related features should not to be considered as mandatory for first-time implementations and prototypes;
- "**Low"** means that the requirement represents a desirable or useful feature for the system, but it should not be considered as mandatory both for architecture design and implementation activities.

All requirements using either the "MUST" or "SHOULD" keyword have to provide also a priority level according to the presented scale. The "comments" attribute can be finally exploited to provide further meta-information for each considered requirement.

3 Authentication & Authorization Infrastructure

3.1 Requirements

Code	Description	Priority
Aa.1	The service should provide federated authentication	HIGH
Aa.2	The service should provide delegated authorization	HIGH
Aa.3	Any communication channel should be secured	HIGH
Aa.4	The service should provide role-based access	HIGH
Aa.5	The service should handle GAIA specific roles	HIGH

3.2 Design

The SparkWorks AA hosts the user accounts, and authorizes other SparkWorks or third-party services to access protected resources according to the user account and the resource owner restrictions. SparkWorks AA provides user registration and administration for User accounts and Resource owner services via a user interface (UI). Moreover, resource owners are able, through the AA Infrastructure, to restrict access to their resources to certain user groups, or provide user-specific contents. The SparkWorks AA provides authorization flows for browser-based, mobile and desktop applications.
In the context of the GAIA requirements, new user roles were added in the Spark Works AA Infrastructure:

- Teacher
- Student
- Local BMS Manager
- Global BMS Manager
- GAIA Administrator

A GAIA Teacher is a superset of the GAIA Student privileges since a GAIA Teacher is also able to review and approve or decline all pending requests from common users to grant the GAIA Student role for his site (including all sub-sites). A GAIA Student has read-only access to the resources data belonging to the Spark Works platform sites (including all sub-sites) that has access on. A Local BMS Manager has additionally the privilege to add data on a resource of his site though the participatory sensing API and, furthermore he is also able to review and approve or decline all pending requests for Student/Teacher grant of his Site (including all sub-sites). Finally, a Global BMS Manager is able to add data on any resource of GAIA (including all GAIA sites and sub-sites) and he is also able to review and handle all of the pending requests for Student/Teacher/Local BMS Manager grant of GAIA.

3.3 Implementation

The main component of the SparkWorks AA infrastructure is the Spark Works Authorization Server (AS) backed by a relational database persisting user accounts and protected services authentication attributes. The SparkWorks Authentication & Authorization Infrastructure is based on the OAuth 2.0 [OAuth] authorization protocol.

An AA flow includes several entities with distinct OAuth2 roles. An end-user is backed as the Resource Owner, any protected API as a Resource Server, any third party application as a client application and finally the authorization API which is backed by the Authorization Server (AS).

3.3.1 Client Application Role

A client application has to register with SparkWorks AS and get a client id and a client secret back. It does not collect user credentials and its role is to obtain a token from the AS either on its own behalf or on behalf of an end-user. This token is used to access protected resources on any Resource server.

3.3.2 Authorization Server Role

The AS main responsibility is to provide the interface for users to confirm that they authorize any client application to act on their behalf. Hence, the AS authenticates the users, authenticates the clients and provides an interface for new users and clients registration. Moreover, another crucial responsibility of the AS is to compute the token content and grant the tokens to the clients.

3.3.3 Grant types

Hereafter OAuth 2.0 grant flows provided by SparkWorks are described.

Authorization code

The authorization code (Figure 6) is obtained by using an authorization server as an intermediary between the client and resource owner. Instead of requesting authorization directly from the resource owner, the client directs the resource owner to an authorization server, which in turn directs the resource owner back to the client with the authorization code. Before directing the resource owner back to the client with the authorization code, the authorization server authenticates the resource owner and obtains authorization. Because the resource owner only authenticates with the authorization server, the resource owner's credentials are never shared with the client [RFC6749, 2012].

Implicit

The implicit grant is a simplified authorization code flow optimized for clients implemented in a browser using a scripting language such as JavaScript. In the implicit flow, instead of issuing the client an authorization code, the client is issued an access token directly (as the result of the resource owner authorization). The grant type is implicit, as no intermediate credentials (such as an authorization code) are issued (and later used to obtain an access token) [RFC6749, 2012].

Password

The resource owner password credentials (Figure 7) can be used directly as an authorization grant to obtain an access token. Even though this grant type requires direct client access to the resource owner credentials, the resource owner credentials are used for a single request and are exchanged for an access token [RFC6749, 2012].

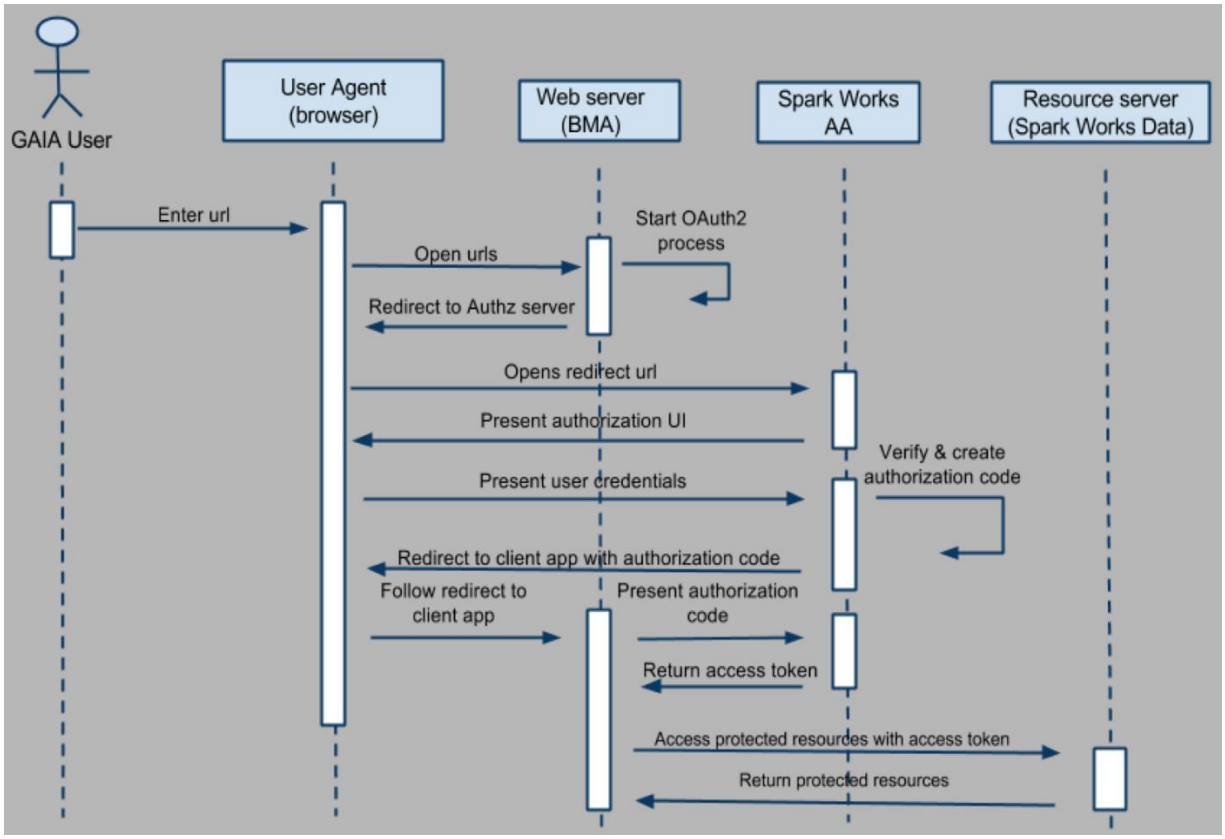

Figure 6 Authorization code grant flow

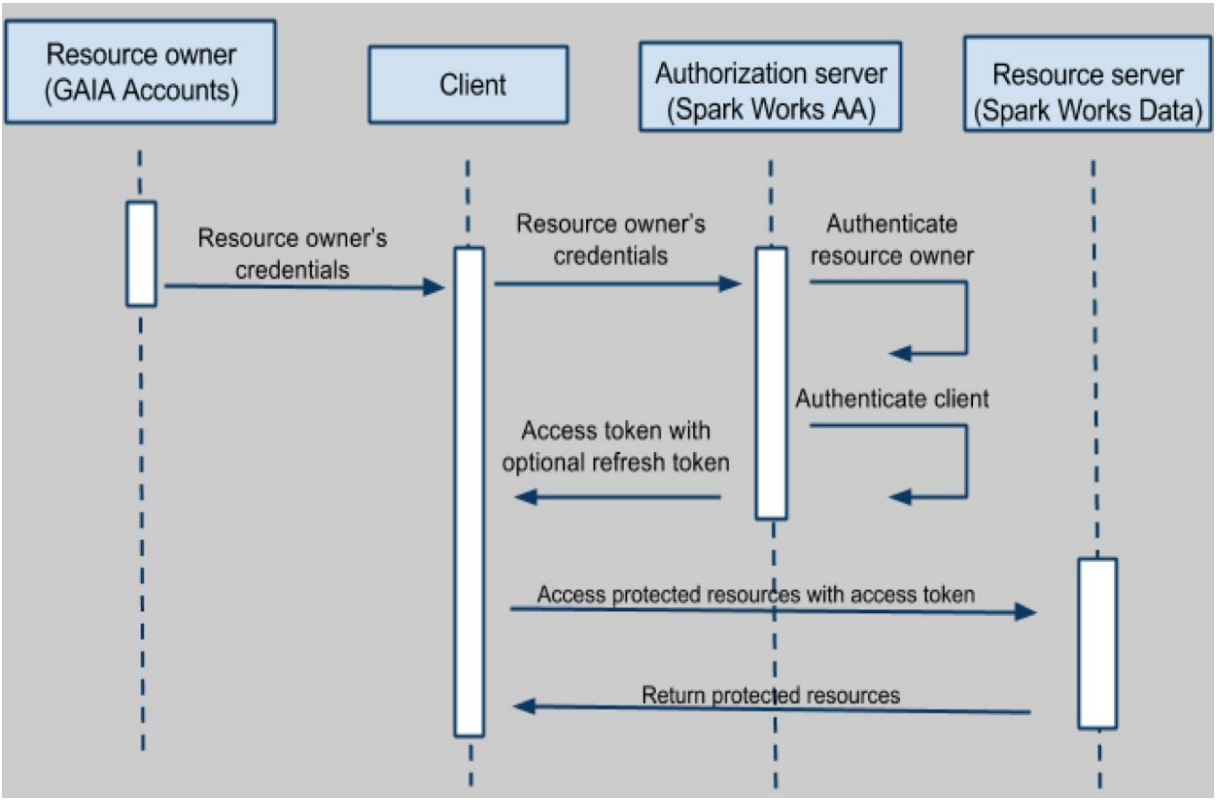

Figure 7 Resource owner grant flow

4 Data Acquisition

The *Data Acquisition* block deals with the heterogeneity of the sensors and API exposed by the different proprietary platforms with the aim to make data uniform before sending them to the GAIA infrastructure for storage and distribution purposes.

4.1 Requirements

Code	Description	Priority
Da.1	The module should be able to integrate with any IoT-ready device	HIGH
Da.2	The API Mapper should be able to poll and push messages from/to the devices	HIGH
Da.3	The messages exchanged should follow a naming convention	HIGH
Da.4	Unique identification of sensors should be strictly defined through uniform resource identifier	HIGH
Da.5	List of characteristics for each sensor type should be defined	HIGH

4.2 Design

4.2.1 API Mapper

The API Mapper acts as a translation proxy for data acquisition. In the context of the GAIA design, it is responsible for polling the devices infrastructure through proprietary APIs and translating the received measurements in a ready to process form for the SparkWorks platform. In general, the SparkWorks API Mapper transforms data to and from the SparkWorks API. The data input type can be, based on each device capabilities (i.e., poll based or/and push based). In more details, the API Mapper is capable to receive data from the IoT devices, but also to send messages/commands to the devices. Furthermore, according to the system design, the API Mapper proxies introduce scalability and modularity in the platform. The messages exchanged through the deployed API Mappers for the GAIA service should follow a specific naming convention, called GAIA Uniform Data Model.

4.2.2 GAIA Uniform Data Model

The messages injected in the GAIA Service platform should follow a specific template. Each message should be a comma separated triplet, indicating the unique sensor URI, the actual measurement value and a timestamp. Sensor names follow a URI scheme implicitly enclosing resource provider information. Each sensor should have a unique name and the name should contain as a prefix the identifier of its gateway, if present. Any device containing multiple instances of the same type of

sensor should use an index suffix. Moreover, each sensor name should be prefixed by its client identifier as registered in the SparkWorks AA service. Any data arriving in the SparkWorks platform that do not match a valid client will be ignored.

Hence, a valid resource URI should look like:

gaia-clientId{/gatewayID}/deviceID/sensorName/{/index}

A predefined set of sensor naming conventions is listed in Table 1.

Table 1 List of sensing input types for GAIA schools and encoded names

Sensor	Encoded Name	Unit Of Measurement
Temperature	temp	Degrees Celsius
Humidity	humid	%
Motion	pir	N/A
Water leak	flood	N/A
Noise	sound	dB
Light	light	lux
Various gas concentrations	Chemical type name (o2, co, co2)	ppm
Atmospheric pressure	press	kPa
Wind speed	wspeed	m/sec
Wind direction	wdir	Degrees (0-360)
Rain gauge	rain	mm
Radiation (background, alpha, beta, gamma)	radiation	µSv/h
Voltage	vol	V
Power Factor	pwf	N/A
Active Power	actpw	W
Reactive Power	reactpw	VAR
Apparent Power	appw	VA

Active Energy (in)	con	Wh
Reactive Energy (in)	reactcon	VARh
Apparent Energy (in)	apcon	VAh
Current	cur	mA

4.2.3 Participatory Sensing

To complement existing GAIA sensor infrastructures and provide additional data from the monitored GAIA buildings, the platform provides Participatory Sensing (PS) capabilities. Apart from the data collected by the deployed GAIA sensors, the GAIA platform is able to collect data on a non-automated way from GAIA users owning a privileged GAIA account, i.e. a Local or Global BMS Manager or a GAIA Administrator. In such way, GAIA participants are able to manually enter additional data readings e.g., luminosity or noise levels from the GAIA buildings. Participatory Sensing users is able to inject data through dedicated GAIA platform resources, the so called "Participatory Sensing" resources.
As shown in Figure 8, a user with appropriate permissions is able to register a dedicated GAIA "Participatory Sensing" resource and assign this newly created resource to a GAIA site through the Data Storage API. Afterwards, any user with permissions on this "Participatory Sensing" resource is able to inject appropriate data to this resource again through the Data Storage API.

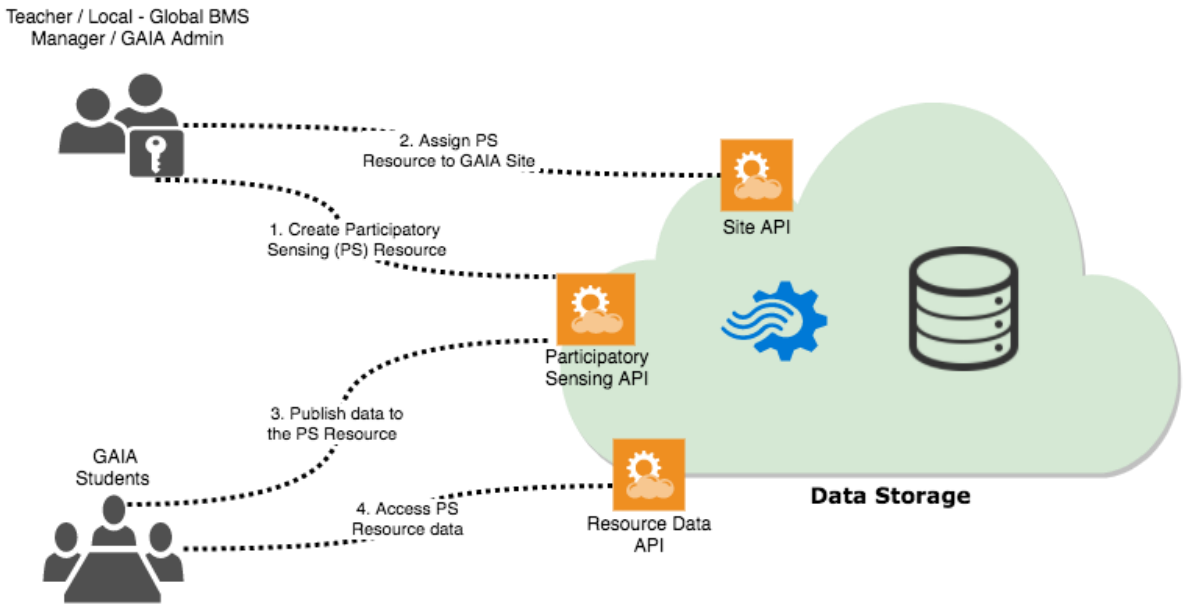

Figure 8 Participatory Sensing (PS) Flow

4.3 Implementation

SparkWorks offers two solutions for integrating with external services and get IoT sensor data, using polling and message queues (see Figure 9).

The first solution is based on polling and was used to get data from several sources, including, for instance, the Synelixis Weather Stations installed in a subset of the school buildings of the project [GAIA1.1]. Synelixis data were accessible through the Synfield API that provided historical information through a RESTful API from Synfield main server. The API is polled by a REST client at a fixed number of minutes (5) for updated data. When new data are found, they are formatted to the GAIA uniform format of and forwarded to the Sparks Processing engine for processing and analysis (see Section 5.2.1). Data can then be accessed through REST APIs.

The second solution follows a pub-sub approach and requires that the external service is capable of publishing the IoT data (generated or gathered) to an MQTT endpoint. The SparkWorks listener application is then able to receive them as streams, format them to the internal format of SparkWorks and forward to the Sparks Processing engine for processing and analysis. The data can then be accessed from the SparkWorks API. Messages inside the MQTT broker can be formatted in multiple formats ranging from plain text to any proprietary protocol. For the plain text messages, the following protocol is used: the topic of the message refers to the device and sensor that generated the message while the actual payload represents the value generated.

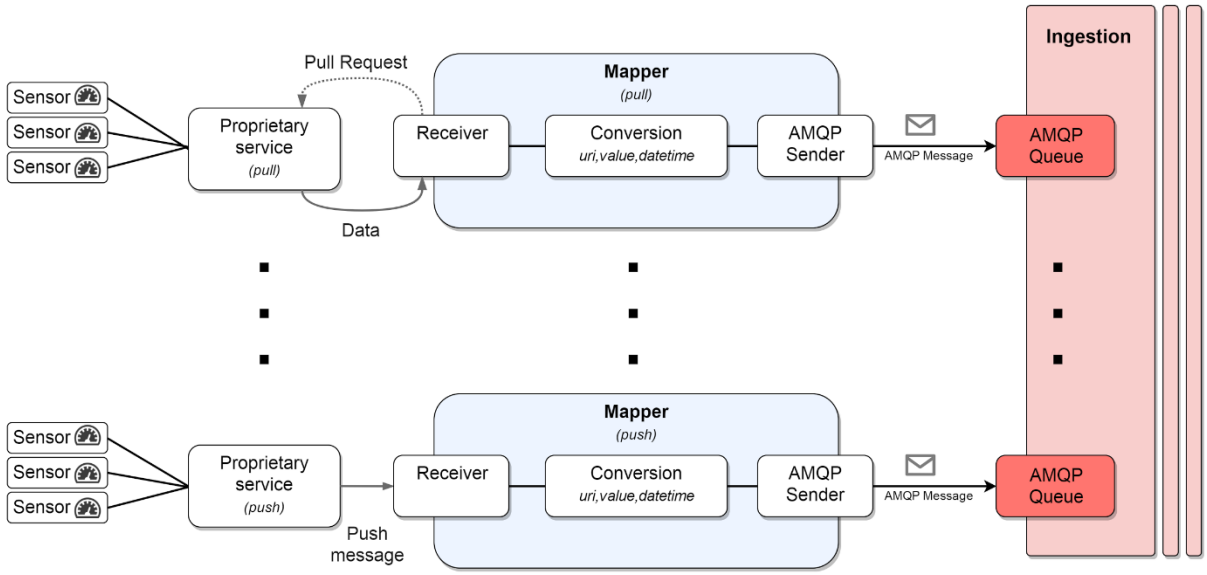

Figure 9 Heterogeneous Data Sources Integration through API Mappers

In order to integrate the different heterogeneous data sources available in the GAIA context several API mappers were implemented. In more details the available API mappers are listed below:

Meazon API Mapper

The Meazon API Mapper was implemented by CTI to integrate the data generated by a number of Meazon devices installed in a subset of the Greek public schools (mainly the 1st Gymnasium of Rafina). It generates data for the power consumption of the buildings, and also the environmental conditions inside a small number of classrooms in the schools. The data arrive to the Mapper from the MQTT broker and are then translated and forwarded to the SparkWorks API.

Synfield API Mapper

The Synfield API Mapper, was implemented by CTI to integrate the data generated by a number of Synfield weather stations installed in a subset of the Greek public schools. It generates data for the weather conditions outside the school buildings including wind speed, wind direction and rain height. The data are polled periodically via the proprietary Synfield API and are then translated and forwarded to the SparkWorks API.

Greenmindset API Mapper

The Greenmindset API Mapper, was implemented by CTI to integrate the data generated by the main infrastructure of sensing nodes installed in all Greek public school buildings. It generates data for the power consumption, environmental and weather conditions inside and outside the school buildings. The data arrive to the Mapper from the MQTT broker and are then translated and forwarded to the SparkWorks API.

EA API Mapper

In EA it was not feasible to access the data from the existing BMS installation. To overcome this issue an additional infrastructure was installed based on a set of drop-in sensors that are described in Section 0. A Python based version of the Sparkworks API Mapper was implemented by CTI in order to provide the sensor information generated to the Sparkworks API. Data from the sensors of the drop-in sensors contain information about the environmental conditions inside the classrooms of EA. The energy consumption sensors installed in EA are of the same design as the ones in the rest of Greek public schools and provide data through the Greenmindset API Mapper.

Söderhamn API Mapper

The Söderhamn API Mapper was developed by Spark Works to integrate the existing BMS installation of Söderhamn school into the GAIA system. The Söderhamn API Mapper polls the building BMS user interface in a configurable frequency and extracts the latest sensor measurements for a set of collected sensors. Afterwards, the data are transformed and injected in Data Acquisition through the message broker.

Prato API Mapper

Prato API Mappers were implemented by CNIT including an API Mapper implementation for Netsens platform and an API Mapper implementation for Synelixis platform. Both implementations are “polling” mappers following the first solution provided from the API Mapper template. They are querying data from Netsens and Synelixis data sources and the transformed data are pushed to Data Acquisition API through the message broker.

Sapienza API Mapper

Sapienza API Mapper was implemented by Over for Sapienza buildings where the source of data is the NanOMeters and OBoxes platform, described in [GAIA1.1]. Consumption data of buildings monitored by Over are acquired by its devices. Both NanOMeters and OBoxes provide a set of REST APIs to read collected consumption data. NanOMeter data are acquired through a middle server that collects them

and pushes them to the Data Acquisition API, whereas OBoxes publish their data by directly connecting to the Data Acquisition Module.

5 Data Storage

5.1 Requirements

Code	Description	Priority
Di.1	The component should expose exactly the same API for external and internal communication	HIGH
Di.2	The component should expose a REST API for public access	HIGH
Di.3	The component should ship a proprietary library to external clients for communication	HIGH
Di.4	The component's APIs should be considered to support Atomicity, Consistency, Isolation, Durability (ACID)	HIGH

5.2 Design

This component is based on the SparkWorks IoT Framework platform, which is designed to enable the easy and fast implementation of applications that utilize an IoT infrastructure. It offers high scalability both in terms of users, number of connected devices and volume of data processed. The SparkWorks platform accommodates real-time processing of information collected from mobile sensors and smart phones and offers fast analytic services.

The SparkWorks platform delivers a set of services that are critical for all IoT installations:

Continuous computation engine – the real-time processing engine provides fast, and reliable processing of an unbounded number of streams of data collected from IoT devices, smartphones and web-services. The SparkWorks computation engine is very fast, being able to process a large amount of data collected from sensor nodes within just seconds.

Online Analytics – data collected from the data streams and the output of the continuous processing are easily selected, extracted and processed to support business intelligence. The online analytics engine allows to organize large volumes of data and visualize them from different points of view.

End-to-end security – communication across the components of the SparkWorks architecture and supported services are compliant with the current standards for Internet security. Communication throughout the service infrastructure is encrypted using data encryption standards like AES and TLS/SSL technologies.

Access management – authorization of users and access to data can be easily managed in real-time down to specific user, device or time of day.

Storage & Replay – data entering the SparkWorks system can be persisted in their original format and associated with the output of the continuous processing engine. Data streams can be forwarded at a later time to different components. Offline processing of data is facilitated for archiving services or for benchmarking different versions of components.

Developing applications based on SparkWorks is very simple and can be done with any programming language. The services described above can be accessed via a well-defined set of APIs.

5.2.1 Overall Framework Architecture

The SparkWorks IoT Framework cloud platform (Figure 10) is designed to enable an easy and fast implementation of applications that utilize an Internet-of-Things infrastructure. It offers high scalability both in terms of users, number of connected devices and volume of data processed. The platform accommodates real-time processing of information collected from mobile sensors and smartphones and offers fast analytic services. The Cloud Services offer real time processing and analysis of unlimited IoT data streams with minimal delay and processing costs. Storage services use state of the art solutions like NoSQL and time series databases to ensure maximum scalability and minimal response times. Access to data retrieved from IoT installations connected to the proposed framework is granted using OAuth2.0 authentication to provide the easiest integration with external services.

The platform delivers a set of services that are critical for all IoT installations, described hereafter.

Continuous computation engine

The real-time processing engine provides fast, and reliable processing of an unbounded number of streams of data collected from IoT devices, smart phones and web-services. The proposed computation engine is very fast, being able to process a large amount of data collected from sensor nodes within just seconds.

Online Analytics

Data collected from the streams of data and the output of the continuous processing are easily selected, extracted and processed to support business intelligence. The online analytics engine allows to organize large volumes of data and visualize them from different points of view.

End-to-end security

Communication across the components of the proposed architecture and supported services are compliant with the current standards for Internet security. Communication throughout the service infrastructure is encrypted using data encryption standards like AES and TLS/SSL technologies.

Access management

Authorization of users and access to data can be easily managed in real-time down to specific user, device or time of day.

Storage & Replay

Data entering the proposed system can be persisted in their original format and associated with the output of the continuous processing engine. Data streams can be forwarded at a later time to different components. Online processing of data is facilitated for archiving services or for benchmarking different versions of components.

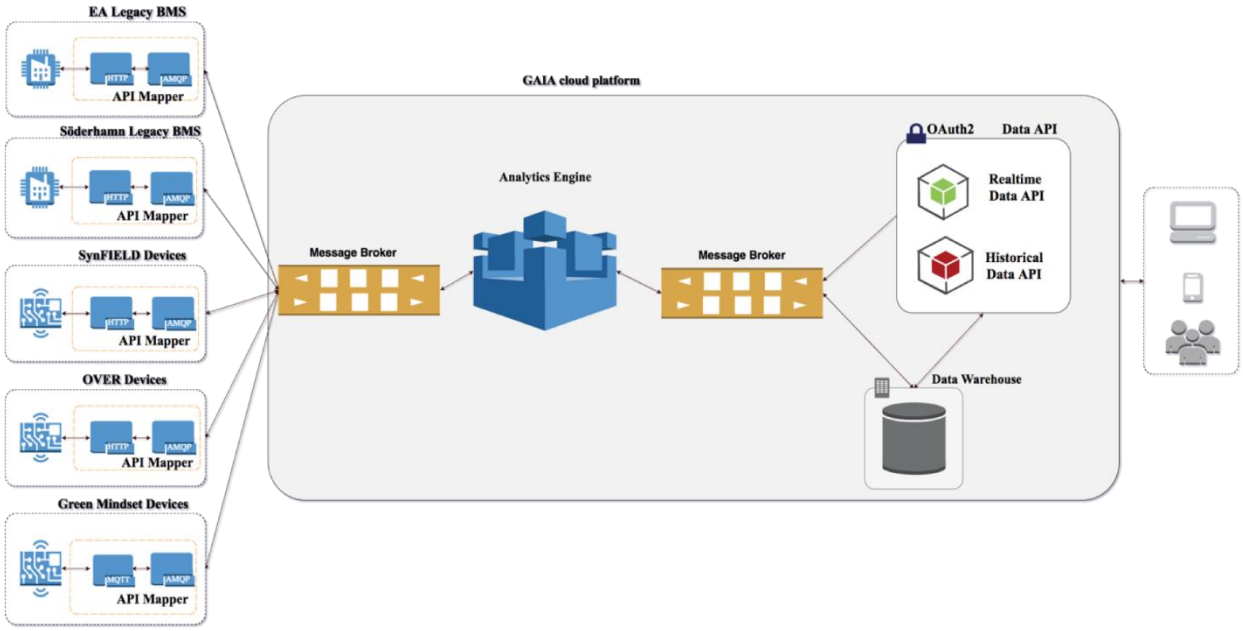

Figure 10 SparkWorks Framework

The SparkWorks Edge Devices solutions allow the processing and storage of data locally in the IoT installation site with minimal installation and maintenance hassle. A simplified version of the SparkWorks Cloud Services can be installed in a local barebones processing node (i.e., Raspberry Pi) when data ownership is important or when Internet connectivity is not always guaranteed. Amongst other services the SparkWorks framework allows to offload data processing to the SparkWorks cloud services when the processing power of the edge device is not sufficient, or when off-site storage is needed (i.e., for data backups).

5.3 Implementation

5.3.1 Processing Engine

In order to provide analytics on the increasing growth of data we present a system architecture which receives streaming data from multiple type of sensors and provide real time analytics over them (see Figure 11). In general, Spark Engine is a process engine which provides the analytics and a storage system which is used for storing those results. The process engine receives events from multiple sensors and executes aggregate operations on these events. The output of the engine is stored at a storage system.

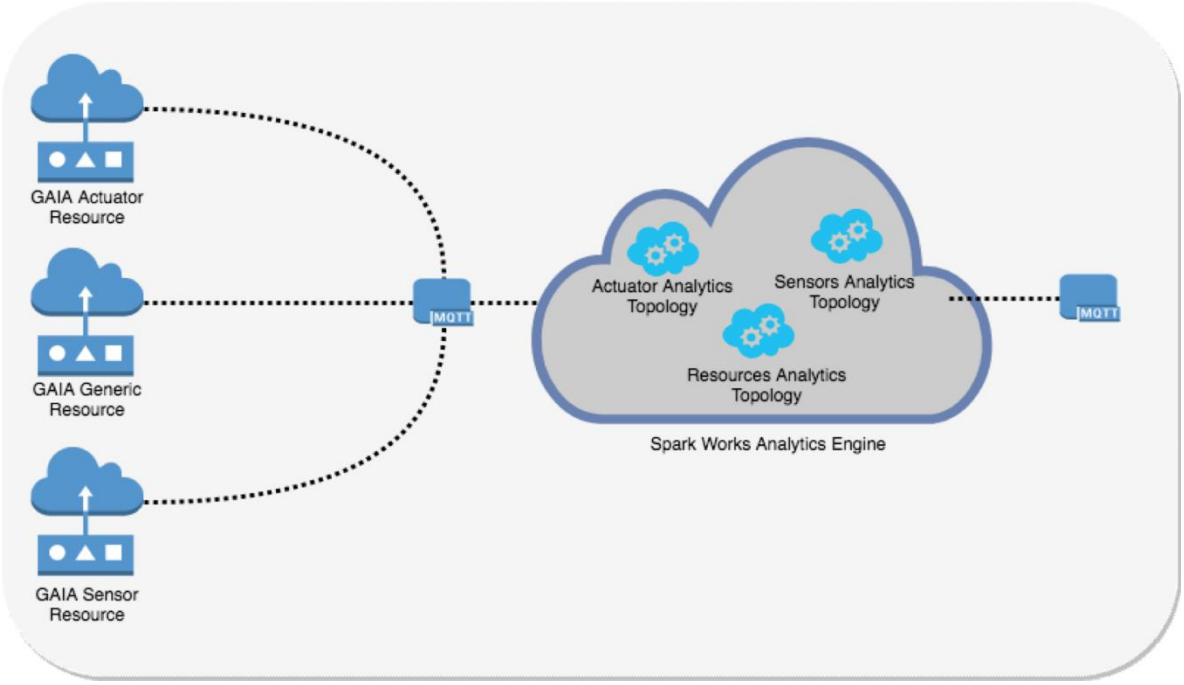

Figure 11 SparkWorks processing engine analytics architecture

5.3.2 Data processing chain

Sensors and actuators produce (periodically or asynchronously) events that are sent to the Processing Engine via RabbitMQ. Those events are usually tuples of pairs: value and timestamp. All data received are collected and forwarded to a queue. For there, they get processed in real time by a storm cluster. The storm cluster has a number of topologies for processing based on the data type. Each topology is responsible for a unique type of sensor such as general measurement sensors (temperature, humidity, wind speed etc.), actuators, power measurement sensors, etc. The produced analytics is outputted into summaries which are stored permanently either to a file system or to a database.

The process engine is composed by topologies for every type of sensor. Each topology has the ability to be easily modified in order to accommodate aggregation operations. We will present the structure of a generic component, aggregator, which is used by topologies.

Aggregators process time intervals. First of all, they store all the events of the time interval and for each new incoming event they process (using functions such as: min/max/mean...) all the stored values of this interval. After this process, they update the existing interval value which is used by the next process level.

Spark Engine receives this data and separates sensors to different types. The engine, which is implemented with Apache Storm, is consisted of topologies. As we mentioned above each topology is responsible for a specific type of sensor. Each topology is consisted of a chain of aggregators which we call process blocks or process levels. The process blocks can aggregate data for specific time intervals (see Figure 12).

Events which enter the storm cluster are processed consecutively. First the storm topology performs

aggregation operations on the streaming data i.e., for a temperature sensor, storm will calculate the average values of the 5min-interval (see Figure 13) and it will store it to memory and disk for further process (when the topology receives more than one events for the same 5min-interval, it calculates the average of those events). Every consecutive 5min-interval aggregate is kept in memory (topology keeps 48 interval values 'k' for 5min, hour, day, month intervals for each device) / stored in disk. Next step is to update the hour intervals. Topology updates the 5min-intervals inside the buffer of the hour processor and stores the average of those 5min-intervals.

The process is same for the daily processor but topology also stores the max/min of the day (based on the hour intervals) and the same for monthly and yearly processors. For power consumer sensors the scheme (topologies inside storm) is the same with the difference that topology has to calculate and store the power consumption.

Aggregators are used to perform aggregation operations on input streaming data.

Our topologies use aggregation for **Power Consumption calculation** (calculate the power consumption of the stream values), **Sum calculation** (summarize the streaming values), **Average Calculation** (calculate the average of the streaming values).

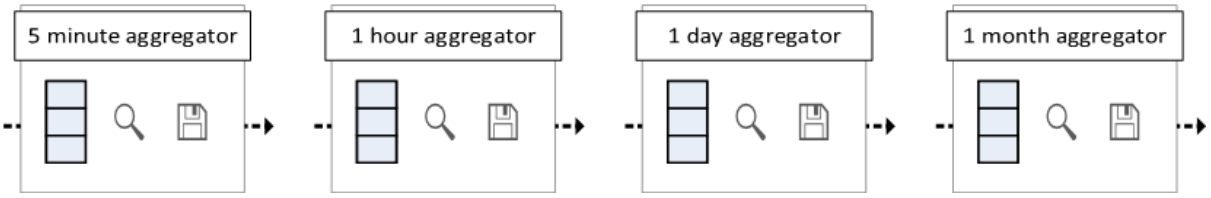

Figure 12 SparkWorks basic analytics process chain

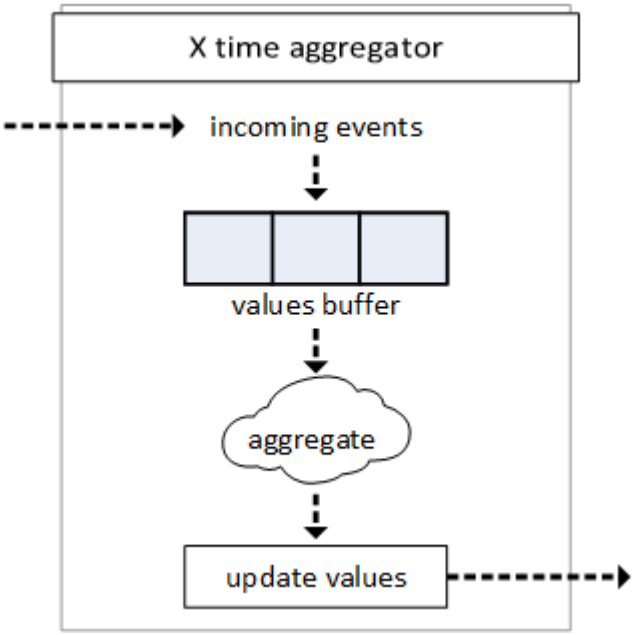

Figure 13 Aggregator module

5.3.3 Public Data Storage API Client

The 1.0.0-SNAPSHOT version of the public Data Storage API Java client is available for use under the

artefact name: “cs-client” on group: “net.sparkworks.cs” on the Spark Works artefacts repository [Sparks Public Repo].

6 Building Knowledge Base

The Building Knowledge Base stores all the useful information about the school buildings (e.g., maximum number of hosted people, area, volume, schedules, type of heating and cooling systems etc.). It exposes REST API providing all the methods to get/edit/delete data.

This section describes the first version of supported data model, the criteria behind its design and the CRUD (Create – Read – Update – Delete) REST operation implemented.

6.1 Requirements

Code	Description	Priority
Kb.1	It must be able to store data about the structure of the building	High
Kb.2	It must be able to store data about the network measurement deployed in the building.	High
Kb.3	It must be able to store specific information on each part of the building (i.e., building itself, floor and room)	Medium
Kb.4	It may be able to store information about the employed heating and cooling systems.	Low
Kb.5	It may be able to store information about the user transportation choices.	Low
Kb.6	It must provide CRUD operation accessible by REST web services	High
Kb.7	It may store information about the person in charge of managing a building.	Low
Kb.8	It may store information about relation between persons and area	Low

6.2 Design

The Entity-Relation model of the database is represented in Figure 14.

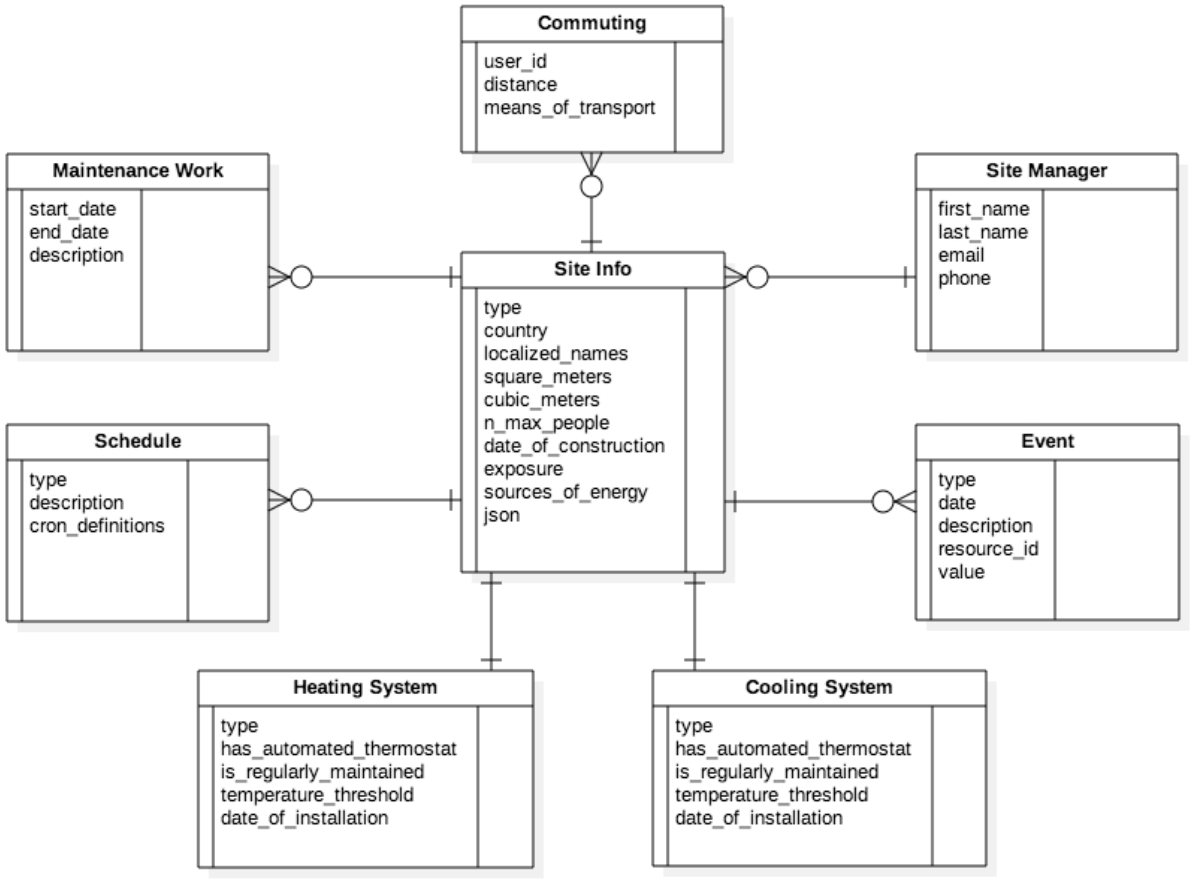

Figure 14 Entity Relation diagram

The Building Knowledge Base represents a logical extension of the Data DB (Data Storage module). The latter is in fact capable of storing the topology of the network of sensors, gateways and other devices that have been installed in every building and area, together with the data they collect, but it contains very few information about the characteristics of every facility.

The core of the Building database is represented by the **SiteInfo** entity.

- A **SiteInfo** has to be associated with an existing Site of the Data DB. A Site can represent either a whole building or only a partial area of it, thus depending on which one it is related to, a **SiteInfo** can store metadata about both of those entities.
- For each SiteInfo, the database also stores data about the people that are in charge of managing it (**SiteManager**) as well as the facility schedules (**Schedule**) and any relevant event (**Event**) that has occurred inside it, such as power outages, power consumption anomalies etc.

Furthermore, the Building Knowledge Base can store information about a site heating and cooling system (**HeatingSystem** and **CoolingSystem**), like the relative temperature thresholds or if there is an automated thermostat installed, while it also enables to keep track of any maintenance work

(**MaintenanceWork**) that has been done in a particular area or building.

Finally, there is also the possibility to store the commuting (**Commuting**) of users, together with the employed means of transport and travelled distance, in order to collect data about the user mobility.

6.3 Implementation details

The Building Knowledge Base is currently implemented using a PostgreSQL powered relational database.

The web services that allow to interact with the Building Knowledge Base have been implemented using the Java Spring Framework and are deployed in a Wildfly Application Server [Wildfly].

7 Analytics module

The Analytics module processes data retrieved from the Data Storage and the Building Knowledge Base and exposes on-demand aggregation- and statistical-derived information through REST APIs. It also detects a set of basic patterns of the observed parameters behaviour in order to provide more complex data analysis like anomaly detection and clustering.

7.1 Requirements

Code	Description	Priority
An.1	It must be able to determine a typical pattern of consumptions per each day of the week	Medium
An.2	It must be able to recognize anomalies if values aggregated from the GAIA platform differ by a certain amount from those ones of the typical pattern	Medium
An.3	It should provide clustering functionalities in order to aggregate typical patterns of multiple resources	Medium
An.4	It may forecast consumption of a given day on the base of typical patterns computed	Low
An.5	It must provide general statistics such as average consumption per area or per resource	High
An.6	It should provide statistics on average at least with the granularity data of “MONTH, DAY, HOUR, QUARTER HOUR”	High
An.7	It must provide REST API implementing common CRUD operation to allow the interface with WP3 applications	High
An.8	It must provide a way to annotate, in a textual form, discovered anomalies	High

7.2 Design

The Analytics Module is made up of three main sub-modules, each one providing a different type of data analysis:

- **Statistics Module:** this module is responsible for providing on-demand general statistics of either a building, an area or a specific device in a given time interval.
- **Clustering Module:** this module is responsible for analysing all the sites in the system, whether they represent a building or a sub-area, in order to classify them in relation to their physical features and consumption patterns.
- **Anomaly Detection Module:** this module is responsible for analysing the consumption of sites and resources, searching for possible anomalies.

Each module retrieves information from both the Building Knowledge Base and the Data Storage and is accessible to clients from a REST interface. Figure 15 shows a high level architecture of the Analytics Module.

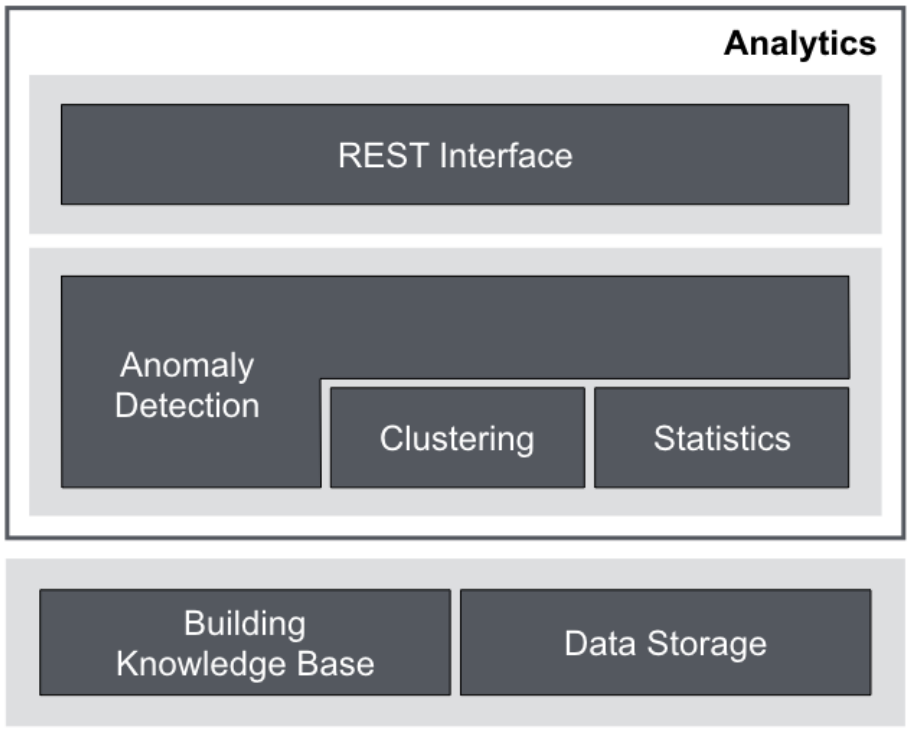

Figure 15 Analytics Module architecture

Statistics Module. The Statistics Module enables to retrieve general statistics about a site or a resource in the system through a set of REST endpoints that the client can fine-tune at different levels using the following parameters:

- **property**: the physical quantity (power consumption, voltage etc.) to consider for the analysis.
- **from/to**: the time interval to consider, specified by expressing its initial and final timestamps.
- **granularity**: the granularity at which the data has to be analysed, which allow to retrieve statistics of consumption either by minutes, hours, days or months.
- **days**: an array of days of the week (MONDAY, TUESDAY, etc.) that can be used to select a subset of days that will be considered during the analysis. The omitted days will be ignored. This enables for example to execute complex analysis on workdays and weekends, giving to the client the possibility to select the right days based on a specific calendar or schedule.

Furthermore, when requesting statistics about a site, the client can specify to weight the data by:

- maximum **number of people** that can be hosted in the site;
- site **square meters**;
- site **cubic meters**.

The analysis summary returned by the Statistics Module contains the following information:

- **unit of measurement** of all the data in the response;
- list of **measurements** taken into account during the analysis;
- **average** value over the whole period;

- **minimum** value over the whole period;
- **maximum** value over the whole period;
- **standard deviation** over the whole period;
- **variance** over the whole period.

Clustering Module. The Clustering Module aims at classifying sites analysing the dataset of metadata in the Building Knowledge Base as well as their energy consumptions in fixed time periods. Notice once again that each site can represent either a building or an area, thus the classification is done on both. The purpose of this analysis is to understand what sites share similar features and behaviours and to provide this information to other modules and clients. Hence, while the clustering is performed in the background inside the Analytics Module, the Clustering Module exposes a REST interface to retrieve the site clusters.
For each site, the clustering algorithms consider the following features:

- date of construction
- square meter
- cubic meter
- maximum number of people
- sources of energy
- type of heating system
- type of cooling system
- average energy consumption over a year
- average temperature over a year
- average energy consumption over a month
- average temperature over a month
- average energy consumption over a season
- average temperature over a season

These features have been chosen in order to allow the algorithm to consider three different aspects of a site when evaluating its similarity to other facilities in the system:

- its architectural structure
- its energy consumptions
- its climate

Each one of these features is then mapped to a component in an *n*-dimensional space in order to represent each site as a vector in such space. Thus sites are organized in a multidimensional vector space and are clustered together according to properly defined mathematical distance functions, such as the cosine similarity or the Euclidean distance. The output of the Clustering Module is therefore a collection of sets storing sites that are considered to be similar according to the defined features. This information can be leveraged by other modules and applications to analyse a site in relation to other similar sites that may have additional data available, thus augmenting the capability of involved algorithms.
Anomaly Detection Module. The Anomaly Detection Module is responsible for inspecting the energy consumption of devices and sites to recognize anomalies in their behaviours. The algorithm works by analysing historic consumption data of every resource in order to infer their typical consumption

pattern. If enough data is available to proceed, the algorithm evaluates the energy consumption data and marks as an anomaly any value that is above a certain threshold of what is considered to be the typical consumption in that same conditions.

The algorithm used to compute the typical pattern is the *vector quantization* technique. As a K-Means algorithm it divides into groups a given set of items. Unlike K-Means, which works with single data points, the vector quantization algorithm is studied to cluster vectors of data points and it is typically used in processing signals or in data compression area.

Using this technique, typical pattern of building in a given time interval is computed. The first step is to fill the training set with consumption data belonging to the historic consumption of a resource. Consumptions are aggregated using the daily granularity, so a typical daily pattern is computed with 24 values, one for each hour of the day. After that, the vector quantization algorithm is used to perform a clustering of these daily patterns. As the K-Means, the algorithm takes as a parameter the number of clusters in which the set must be divided.

Given the result of vector quantization, the data of the period requested by the user is analysed and compared to the relative typical consumption. In particular, a percentage difference D is computed as $D = (V_1 - V_2)/(V_1 + V_2) * 2 * 100$ where V_1 is the value that is being compared and V_2 is the typical consumption. Then, if this difference is greater than a certain threshold the relative measurement is marked as an anomaly. Figure 16 shows an example of the result generated by this algorithm where no anomaly is present, while Figure 17 shows an example where an anomaly is actually detected.

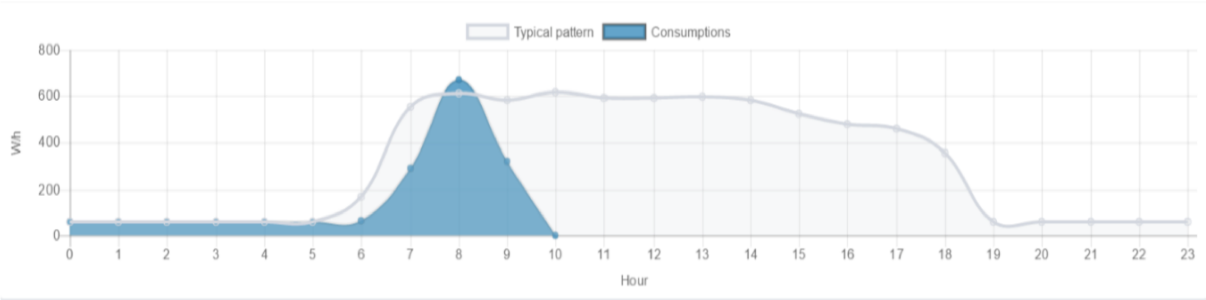

Figure 16 Analytics: no anomaly detected

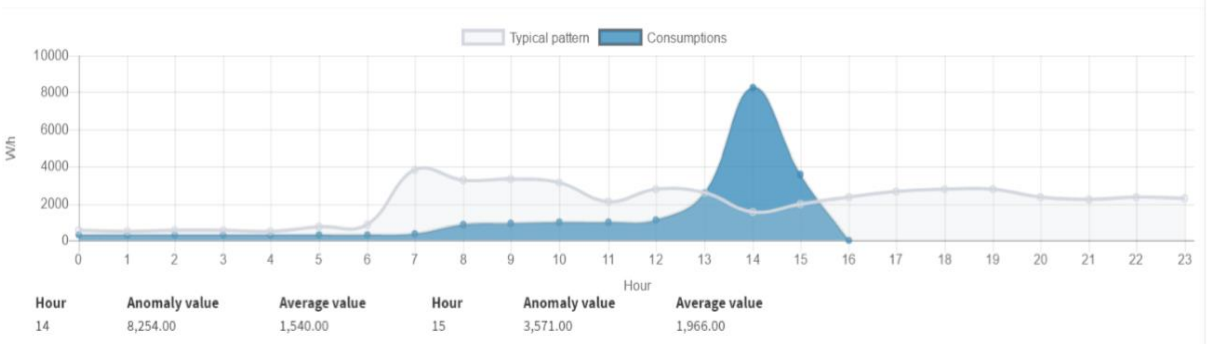

Figure 17 Analytics: anomaly detected

7.3 Implementation details

The Analytics Module is currently implemented using the Java Spring Framework for the persistency and web layers. Furthermore, the Statistics, Clustering, and Anomaly Detection modules leverages algorithms and functionalities offered by open source Java libraries, including Java Apache Commons Math and Java Weka.

The Analytics Module is deployed inside a Wildfly Application Server [WildFly].

8 Recommendation Engine

The GAIA Recommendation Engine produces energy-saving recommendations, to be appropriately delivered to end-users through applications, by leveraging contextual information provided by other GAIA Service Platform modules (i.e., measurements from the Data Storage, building information from the Building Knowledge base, analytics from the Analytics module). The Recommendation Engine primarily provides suggestions for energy-saving behaviour changes (targeting building managers, students and teachers as well), but can also provide building managers with suggestions for technical maintenance interventions or building renewal actions.

In GAIA, the conditions that trigger the production of recommendations and the recommendation content should be customizable according to the school building's characteristics (geographical location, internal space allocation, habits, etc.).

8.1 Requirements

Code	Description	Priority
Re.1	It must be able to query the Data Storage block APIs for gathering sensors measurements	High
Re.2	It should be able to query the Analytics block APIs for gathering analytics on sensors measurements	Medium
Re.3	It should be able to query the Building Knowledge Base APIs for gathering information on school buildings	Medium
Re.3	It must generate recommendations at the occurrence of energy efficiency relevant conditions	Medium
Re.4	It must log events that are considered relevant to end-user applications' purposes (e.g., report generation)	High
Re.5	It must push notifications at the occurrence of energy efficiency relevant conditions and recommendations generation	High
Re.6	It must expose an API for accessing events	High
Re.7	It should allow the specification of customized rules per school building/area/sensor (e.g., customized threshold values, customized notification message, etc.)	Medium
Re.8	It should allow to define logical group of rules for simplified management and visualization purposes	Medium
Re.9	It may allow the specification of new rule instances and new group of rules	Low
Re.10	It may support separate notification channels for different schools	Low
Re.11	It should use the GAIA AA service	Low

8.2 Design

In order to cope with specific GAIA domain's requirements (which can be considered an example of Web of Things scenarios), we adopted a rule engine developed from scratch. A *rule* is a way of representing knowledge of the form: "If some condition is true then do some action". In GAIA, relevant knowledge is associated to physical and logical entities of the monitored environment (e.g., classrooms, temperature in the classroom, etc.), therefore a model of the application domain is needed so rules can be defined on top of it. For instance, a building may be modelled as a tree. Each node may be used to model a physical area inside the building (e.g.,a classroom) and deployed sensors and related rules (e.g., anomalous energy consumption during class activities) can be attached to such node. This representation would allow users to navigate, organize and manage rules in an easy and intuitive way.

A further requirement is the need of easily customizing different instances of a given rule according to the area the rule is attached to. The proposed rule engine allows to specify a rule generic behaviour as a Java class and provide specific settings (e.g., sensor endpoints, threshold values) for each instance. Several stable rule engines exist (including open source ones), such as Drools [Drools] and Jess [Jess], just to mention some popular and powerful ones. These engines have usually a steep learning curve and it is hard to extend the rule set since rule conditions are typically expressed in terms of Java object facts. On the contrary, our approach relies on a resource-based and REST-compliant model, described hereafter, which represents sensors, smart things and relevant domain entities and rules as web-addressable resources.

Hereafter we describe the conceptual model of the module. The rule engine knowledge of the application domain is represented through a resource-based model. A *Resource R* is any entity that can be identified through a URI and can be accessed and manipulated through Create Read Update and Delete (CRUD) operations via HTTP protocol, in accordance to main principles and best practices of the REST architectural style.

We define the following main types of *Resources*: *Area*, *Sensor*, *Parameter* and *Rule*.

The class diagram in Figure 18 shows the main classes. The *Area* represents a physical or symbolic location/area, which can be further characterized with information pertaining the domain of interest (e.g., an Area representing a school building can be enhanced with information regarding number of students, surface, yearly energy consumption, etc.). A *Sensor* represents a physical sensor that gathers monitoring parameters (i.e., resources of Parameter type) covering a specific *Area*. A *Parameter* may represent a physical parameter (e.g., temperature) gathered by a *Sensor* or a parameter derived from physical ones (e.g., relative humidity). A *GaiaRule* is a resource whose internal behaviour consists in verifying a condition and, if the condition holds, triggering an action. A Rule is further specialized as an *AtomicRule* or a *CompositeRule*. An *AtomicRule* is a rule whose condition is specified in a self-contained way, while a *CompositeRule* is defined as a composition of conditions of other rules (children rules), which can be *AtomicRule* as well as *CompositeRule*.

Figure 19 provides an example of the resource-based graph generated when instantiating the above-

mentioned model in a specific context (i.e., a school involved in GAIA experimentation activities). The root node represents the *School Area*, which contains other *Areas*. In our example the school building is made of a set of *Areas*: a *SportBlock* (e.g., the gymnasium hall) and a *TeachingBlock*, containing *Classrooms*, *Laboratories* and a *Hall*. A *PowerFactor Rule*, checking if the power factor is below a given threshold, is assigned to the *TeachingBlock* and the *Hall* Areas, using as input the *PowerFactor* Parameter measured by the corresponding *Sensors*. A *Rule* that evaluates the level of external luminosity and the use of artificial light (by analysing the value of absorbed *Active Power*) to suggest energy saving actions (e.g., switch off the light and exploit the natural light) is assigned to the Hall. Finally, a *ComfortIndex* Rule is assigned to the Sport Block to evaluate the heat index provided by the joint evaluation of Temperature and Relative Humidity, measured by Sensors.

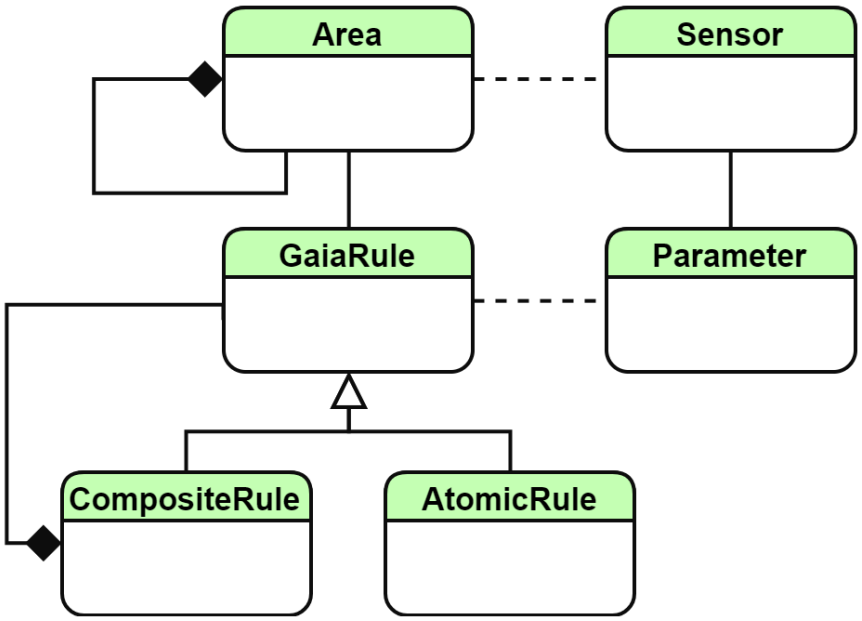

Figure 18 Recommendation Engine conceptual model

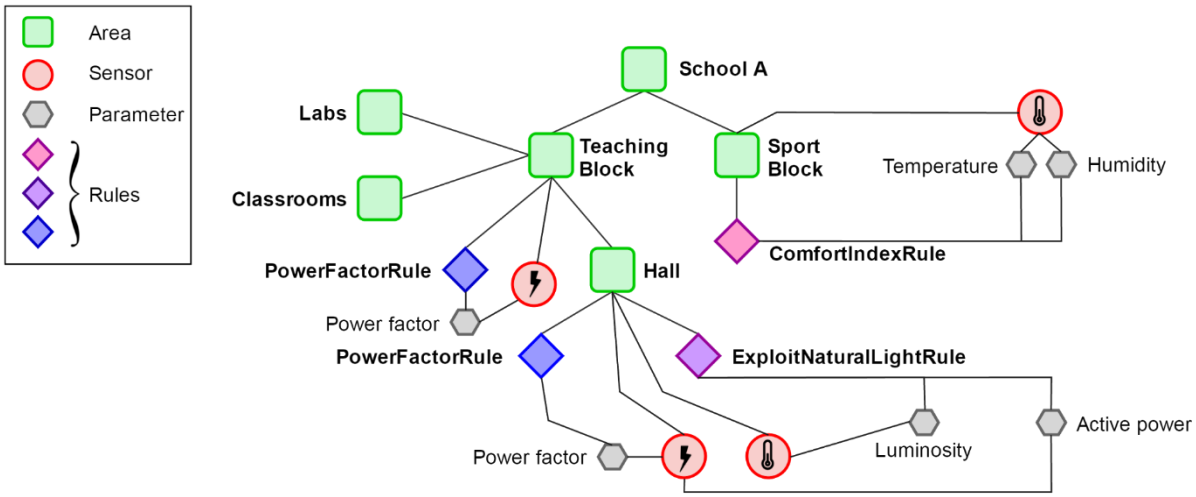

Figure 19 Instance of the Recommendation conceptual model (example)

The rules provided above as an example are a specialization of the generic *GaiaRule* (e.g., GaiaRule subclasses). The GaiaRule defines the basic structure and behaviour of a rule in the engine logic. It provides common attributes, default implementation of basic methods and some services implementing the I/O behaviour. The GaiaRule attributes are listed in Table 2. The GaiaRule also provides the handle for accessing some common services that can be re-used by more specific rules (see Table 3).

Table 2 Attributes of GAIA Rule

name	the name of the rule, useful to discriminate different instances
suggestion	a brief textual recommendation to be sent to the users
rid	the identifier of the rule on the database (generated by the DBMS)
description	a brief description of the rule's behavior (optional)
fireInterval	Interval in seconds between 2 fires of the rule (optional, default = 0)
fireCron	A cron expression to limit the fire of the rule in specific moments (optional)

Table 3 Services in the GAIA Rule

WebsocketService	service for sending push notifications through a WebSocket channel
EventService	contains the logic for logging interesting events on the database
MeasurementRepository	provides the measurements gathered by the sensor infrastructure
BuildingDatabaseService	gives access to the GAIA building database
RuleDatabaseService	gives access to the underlying rule database
ScheduleService	provides the schedules/activity calendars for the school/areas
WeatherService	interface with the weather forecast/history provided by apixu

The GaiaRule abstract class defines the general behaviour of a rule through the methods listed in Table 4. Every *GaiaRule* subclass must implement the *condition()* method (i.e., the condition to be checked) and can customize the behaviour by overriding the default *action()* method.

Table 4 Main operations of the GAIA Rule

init()	executes the needed initialization tasks, if needed, and validates the rule (i.e. it checks if all the required fields are loaded). The result of this method is checked when the rule is loaded, if false the rule is discarded.
condition()	verifies if the condition is met, this is left as an abstract method, that has to be implemented in a subclass
fire()	invokes action() if the condition() method returns true after having checked if the rule is in a valid interval evaluating the *fireInterval* and

	fireCron expression
action()	sends a push notification through the websocket channel containing the textual suggestion and useful information about what happened (e.g., thresholds, values measured) and logs the event

The Engine provides different ways of specifying and modifying rules into the system. On this perspective, rules can be grouped into 3 main categories:

Custom rules: these rules have a customized behaviour that is defined programmatically. However, different instances of these rules can be defined at run time by appropriately specifying a set of configurable parameters.

Template rules: these rules implement a basic, predefined behaviour that has to be further specified with a set of configurable parameters that better specify the scope of the rule (e.g., the type of measurement to be considered, etc.). An example of a template rule is:

- **ThresholdRule**: compares a value with a threshold. The measurement, the threshold and the comparison operator are configurable and can be provided at runtime.
- **ExpressionRule:** evaluates an expression defined by the user as a parameter containing references to measurements and other parameters.

Composite rules: rules whose condition is based on the conditions of the children rules. At present we have defined the following main types of composition:

- **AnyCompositeRule**: it is triggered if any of the children's conditions are true (simulating a logical OR behaviour).
- **AllCompositeRule**: it is triggered if all the children's conditions are true (simulating a logical AND behaviour).
- **RepeatingRule**: the rule updates a counter each time a certain condition (encoded in a children rule) is verified in a given time interval in a consecutive manner. If the value of the counter exceeds a threshold, the action of the Repeating Rule instance is triggered. This type of rule allows to evaluate a condition which takes into account how many times an event occurred consecutively (e.g., the active power is above a given threshold during a period of twenty minutes).

8.3 Implementation

The Recommendation Engine implementation is made of two main modules (Figure 20):

- a *Rule engine*, implemented as a Java application, leveraging the Spring Framework [Spring].
- a *Rule persistence* module implemented using an open source database (DB).

The Rule engine is responsible for executing user-defined rules, according to the resource-based model described above. The resource-based graph that encodes the application domain knowledge (rule instances as well as other information characterizing the domain of interest) is persisted in the

external database and loaded at runtime by the Rule engine to instantiate the corresponding rule objects. More precisely, at runtime the resource-based graph model is traversed and, for each encountered rule, the structure of the rule is replicated in the engine, which instantiates the rule class, characterizing the actual instance filling the class attributes with the values retrieved from the DB. A Scheduler periodically executes the Gaia Rules. A set of common services are implemented to handle the connection to the DB, authentication, and the handling of a WebSocket Connection DB.

Thanks to the above discussed design and implementation rules, the Recommendation engine copes with the following requirements:

- **Extensibility:** new type of rules can be created from scratch (programmatically) or by editing the configuration of already implemented customizable rules (e.g., specifying an expression to be evaluated) to cope with unforeseen application needs and/or context changes (e.g., new kind of sensor added, new recommendation scenarios, etc.).
- **Configurability:** the same type of rule can be applied to create distinct instances to be applied to different schools and different areas of the same school, with appropriate customization of thresholds, sensors' identifiers (i.e., URIs) and content of notification messages.
- **Ease of use:** the user should be capable to easily navigate the set of rules associated to a school and to its constituent areas (e.g., classrooms, laboratories, etc.) and modify them as needed (e.g., by adding, modifying and deleting rules.

Hereafter we provide more details on how these modules interact to implement and handle the GAIA rule model.

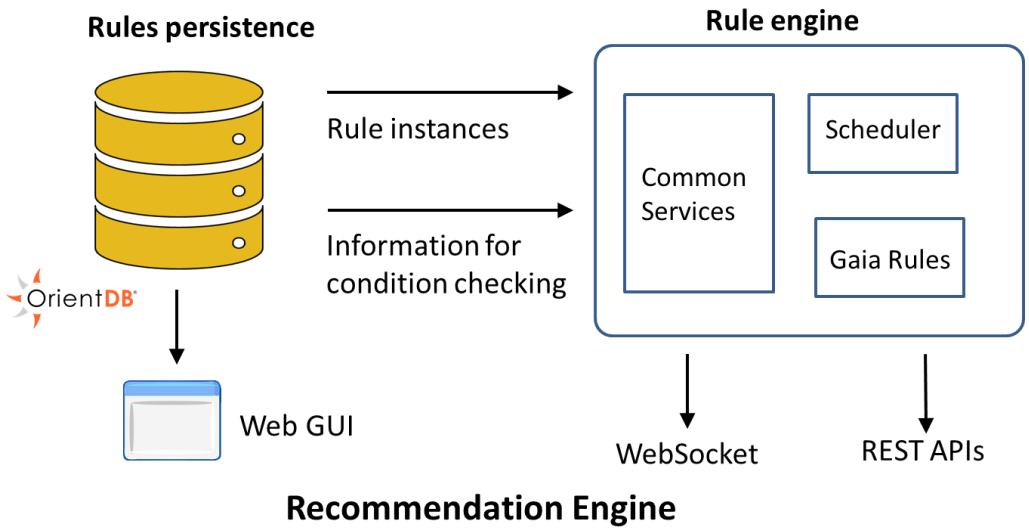

Figure 20 Recommendation Engine Architecture

8.3.1 Rules

While the specification of the rule's behaviour is done programmatically, the structure and the parameters of the rules are externalized so to be configurable. In other words, a set of specific Gaia Rules (e.g., PowerFactor, ComfortIndex, etc.) is specified as subclasses of the generic GaiaRule in the Rule Engine software. A subclass specifies some behaviour (e.g., check that Power factor is below a

given threshold). However, the parameters of such subclass can be different from areas to areas and from school to school. Therefore, each instance of such subclass can be referred to a given Area in the model and have customized parameters assigned.

These customized parameters are stored in a hybrid Graph/Document database (OrientDB 2.2.x) [OrientDB]. At runtime, the tree of rules specified in the DB is traversed and the structure of the rule is replicated in the Rule engine which produces the corresponding instances for the Rule classes and fills the fields using Java Reflection according to the values retrieved from the DB.

Annotations

Rule fields in the Rule class implementation in the Recommendation Engine software can be annotated with different labels (Java Annotations) to guide the instantiation and management of rule objects, as listed hereafter:

@LogMe(event=true, notification=true):	field to be inserted into a notification/event
@LoadMe(required=true):	field to be retrieved from the database
@URI:	field representing a URI, it will be added to a list of resources to be queried to obtain the values rules conditions are checked upon.

Hereafter we provide an example of the implementation of a Power Factor rule, extending the generic Gaia Rule. When instantiating the rule, the values of URI of the measurement resource and the PowerFactor threshold are loaded from the DB.

```
public class PowerFactor extends GaiaRule{

        @LogMe @LoadMe @URI
        public String meter_uri;
        @LogMe(event=false)   @LoadMe
        public Double pwf_threshold = 0.7; //Default provided
        @LogMe
        public Double pwf_value;
        @Override
        public boolean condition() {
                        pwf_value = new Double[1];
                        pwf_value[0] = measurements.getLatestFor(meter_uri).getReading();
                        return pwf_value[0] < pwf_threshold;
           }
      //public boolean condition(){}; Default action inherited by GaiaRule superclass
}
```

8.3.2 Persistence

The persistence layer is realized with OrientDB 2.2.x., a hybrid Graph/Document NoSQL database. It offers a persistence model that accommodates well our resource graph-based model. Moreover, the adopted database system offers a set of REST APIs for easy access to the resources and comes with an intuitive web GUI that can be used to create and edit the resource information model, including the structure and the parameters of the rules.

In our implementation, we have created a class in the OrientDB schema for every Java class

representing a rule (with the convention of using the same name); each class in the DB should contain the configurable fields needed by the rule to be executed. Since the schema supports class inheritance every rule inherits some basic properties from the *GaiaRule* reflecting the application model. Table 5 provides an example of the fields of a *ComfortIndex* Rule instance stored in the DB. Once the schema has been created it is possible to define the rules instances in the database, fill the required properties and link rules and areas into a tree structure. The Recommendation Engine application loads this structure traversing the graph persisted on the database, instantiates the right classes at runtime and fills the required fields using Java Reflection. OrientDB supports both schema-less and schema-full mode, we use a schema-hybrid approach to have classes with predefined properties (some of them mandatory) which can be extended with new fields at instance level. This feature is needed to support the specification of Template Rules.

Table 5 Example of a Gaia Rule persisted in the DB

Comfort index		
Field name	**value (example)**	**description**
@rid	#25:241	The ID of the vertex in the database
@class	ComfortIndex	The class of the vertex in the database / Java class name
name	CI Room 3	Name of the rule
description	[...]	Description of the rule
suggestion	Open the window	Brief suggestion to give when triggered
temperature_uri	gaia-x/gw1/temp	URI of the temperature sensor
humidity_uri	gaia-x/gw1/humid	URI of the humidity sensor
threshold	32	Threshold for the computed comfort index

OrientDB WEB UI

Studio is a web interface for the administration of OrientDB that comes in bundle with the OrientDB server. It includes a graphic editor for the graph that can help to visualize the structure of the rules and can aid the editing of the structure and the parameters.

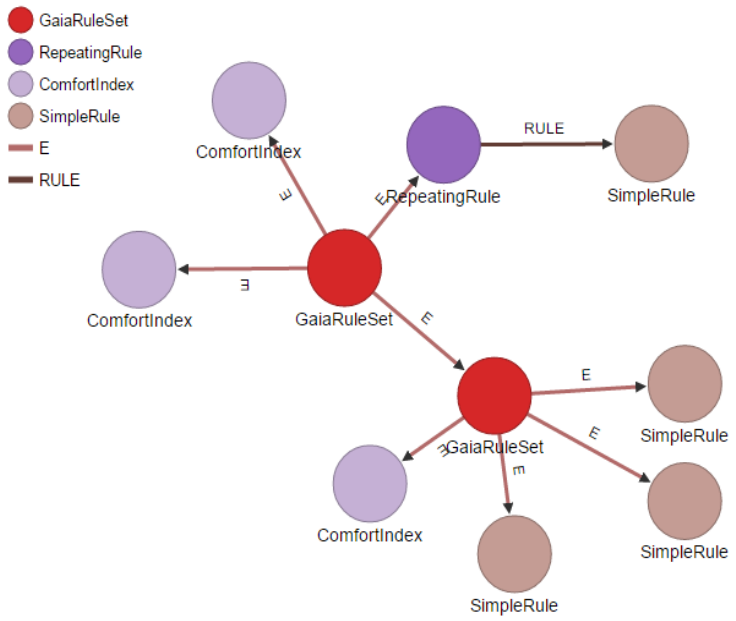

Figure 12 Graphical graph editor

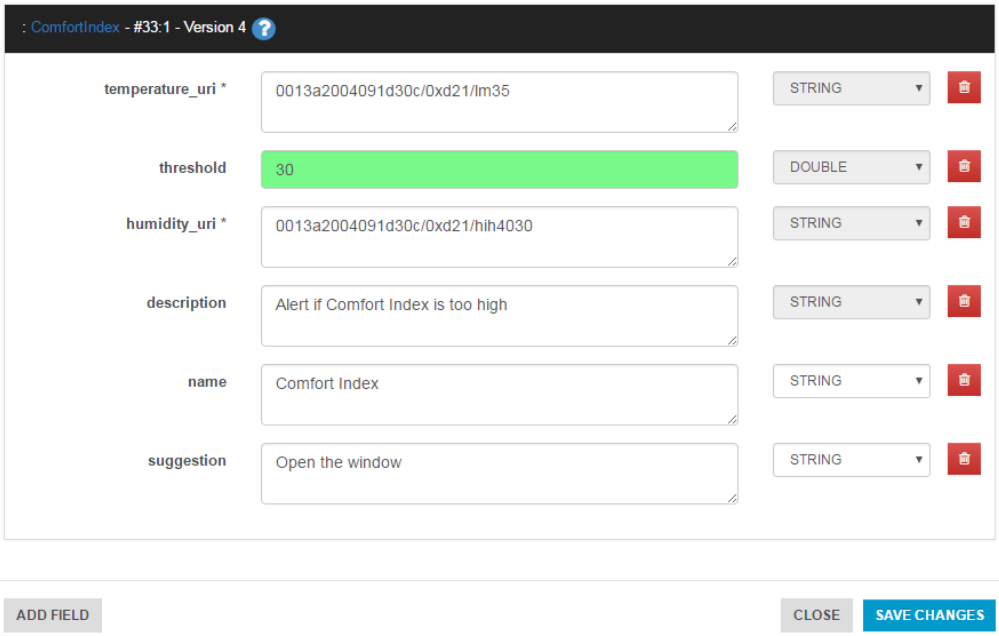

Figure 13 Interface for rule editing

OrientDB REST API

OrientDB RESTful HTTP protocol allows to talk with an OrientDB Server instance using the HTTP protocol and JSON. This API may be used for manipulating the parameters and the structure of the rules tree. The documentation is available at [OrientDB]. Several easy- to-use clients (written in different languages) are available directly from the website.

8.3.3 Execution flow

Hereafter we describe the main steps of the recommendation engine execution flow: initialization, scheduled execution, rule fired.

Initialization

In the initialization phase, the engine loads the resource model from the database and internally replicate the structure following the composite pattern. This is done by instantiating the right Java classes using the @class property of each vertex stored in the database. For each Rule resource retrieved from the DB, the corresponding Java object is created by filling the object attributes with the values stored in the DB. The object is initialized and a basic validation of the fields is executed, the object is added to the tree only if it is valid. The detailed flow of actions is reported hereafter:

1. Load the rule tree from the database.
2. Traverse the tree and replicate the structure following the composite pattern. This is done by instantiating the right Java classes using the *@class* property of each vertex stored in the database.
3. For each rule:
    a. Fill the fields annotated with *@LoadMe.*
    b. Add fields annotated with *@URI* to the URI set.
    c. Initialize the object performing basic validation.
    d. Add to the parent if the initialization method returns true.
4. Iterate over the set of URIs querying the Gaia Platform for converting every URI to a resource ID.

Scheduled execution

In the operational phase, the following tasks are periodically performed: the measurement values required by the rules are acquired through appropriate requests to the Parameters URIs at the beginning of each iteration and shared among all rule instances and the *fire()* method is invoked recursively on the composite structure of rules. The detailed flow of actions is reported hereafter:

1. Check if the rule tree needs an update (i.e., update forced by external request).
2. Update the measurements required by the rules scanning the URI set.
3. For each school fire the rules:
    a. Get the root of the school's tree.
    b. Call recursively the fire method on the composite structure.

Rule firing

Rule firing is performed through the following steps:

1. Check if the parameters of the rule (i.e. *fireCron* and *fireInterval*) allow the rule to be fired. If the result is positive goes to step 2, otherwise the flow is stopped.
2. The default behaviour is to evaluate the condition and, if true, invoke the action method (step 3).
3. Create a GaiaNotification object, user reflection to add *@LogMe* annotated fields to the values map of the notification.

4. Send the notification to the WebSocket channel relative to the school (identified by the schoolId).
5. Log the event in the database.

9 Drop-in sensor devices

Installing a dedicated sensing infrastructure composed of IoT devices and sensors is an expensive process, in terms of both time and money. Especially in the cases of schools in remote areas, where the budget for each school is limited, hardware and software failures can lead to service disruption. Also, in some cases interventions to the building's power distribution system or interior design are not allowed (e.g., historical buildings). For those cases, GAIA has implemented a solution for providing teachers and building managers with a set of tools to generate a more detailed view of the situation inside the building, in terms of monitoring environmental parameters. This approach consists in using local sensor nodes that are connected to an independent device that can connect, through a WiFi or Ethernet network, to the GAIA backend and provide data for the building. Such devices, can be used in two different ways: either as an IoT sensing infrastructure, or as enablers for classroom activities, instructed by the teachers and performed by the students.

One key issue in this case is the open/generic nature of the hardware/software stack used. By using more standardized platforms, this helps to have a more "deterministic" behaviour in terms of performance or issues that may arise, as well as having a broader selection of tools to handle malfunctions. In certain cases, this even gives room to school buildings that adopt this solution to participate in the design of their specific GAIA installation. E.g., in the case of the Söderhamn school described hereafter, students of the technical high school helped in designing and 3D-printing a specific casing solution to be used for the nodes inside the classrooms.

In the rest of this section, we try to describe both cases and present how they are used in the context of the project.

9.1 Raspberry Pi in EA / Söderhamn

As an addition to the already available sensor infrastructure ecosystem of GAIA, we have developed a drop-in sensor box based on the popular Raspberry Pi board and Grove sensors. Each node comprises a single Raspberry Pi 3 Model B, running a Raspbian operating system, together with a Grove Pi hat to easily connect the sensors on the main board. The main advantage of using the Grove family of sensors, apart from their worldwide availability, is the use of standardized connectors for interfacing with the main boards, which makes it makes very easy, as well as safe for students, to assemble/disassemble or use during lab activities.

With respect to the set of utilized sensors in GAIA buildings, we have selected the following ones:

- *Grove Digital Light sensor*[1]*,* to measure the luminosity levels inside the classrooms.
- *Grove Temperature and Humidity sensor*[2]*,* to measure the temperature and humidity inside the classrooms.

[1] http://wiki.seeed.cc/Grove-Digital_Light_Sensor/

[2] http://wiki.seeed.cc/Grove-Temperature_and_Humidity_Sensor_Pro/

- *Grove PIR Motion Sensor*[3], to measure the occupancy level of the classroom.
- *OJ-CG306 Sound Sensor*[4], to measure the noise level of the classroom.

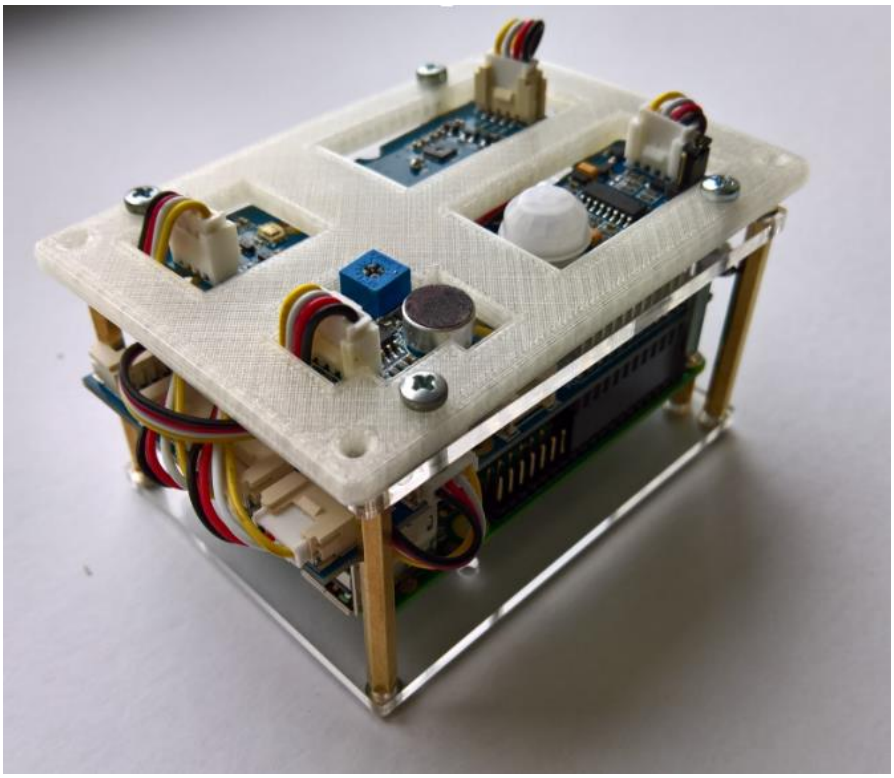

Figure 21 View of the Raspberry Pi-based GAIA devices utilised inside classrooms

In order to send the data produced by the sensor modules to the project's backend services, we essentially follow the same principles described in Section 4. Each Raspberry Pi device is actually an instance of a API Mapper that periodically, every 1 minute, collects the data from the sensors and transmits them to the Data Acquisition component. For each device a set of credentials and configuration parameters is required to setup the devices. In more detail, each device needs a *client-id* and *client-secret key* to authorize with the system and a *unique identifier* to distinguish the sensors of the device from sensors of similar devices.

In EA, the existing BMS did not allow for external connections resulting in an inability to retrieve usable for the project. Also, it was not possible to install a set of dedicated sensing devices in each classroom of the school, due to the number of rooms available in the EA building (around 7 classrooms for each educational level). As a solution, eight nodes based on the aforementioned design were installed in different classrooms of the building, so that students of EA can interact with them during the course of the project. Such devices are powered directly by a wall outlet and communicate over WiFi with the GAIA service platform.

Similarly, in Söderhamn, the BMS system provided a limited number of data to work with (a general internal and external temperature, as well as some data for water and power consumption). In order

[3] http://wiki.seeed.cc/Grove-PIR_Motion_Sensor/

[4] http://wiki.seeed.cc/Grove-Sound_Sensor/

to extend the level of detail for the conditions inside the building, a set of Raspberry Pi nodes, as described above, were purchased. The devices were built and installed by the students of the school in order to increase their understanding of the installed system and its capabilities, adding more educational value to the whole process.

9.2 Raspberry Pi in GAIA's Educational Lab kit

The Educational Lab kit comprises a Raspberry Pi board and a set of Grove sensors[5], as well as a Circuit Scribe kit[6] with electronic/circuit components like LEDs and motors, as well as a conductive ink paint to help visualize the data of the schools. The usage of the kit is twofold: first, it is used inside the classrooms by the kids to visualize the data provided by the infrastructure of their school; secondly, it is used in combination with the Grove Sensors and the participatory sensing features described in Section 4.2.3.

9.2.1 Data Visualization Scenario

For the first scenario, the Raspberry Pi gathers real-time data from the school, where the lab activity or lecture is organized, using the teacher's credentials and displays them using a circuit drawn by the students, on top of a printed floor plan of the school. The circuit drawn by the students represents a simplified version of the building's energy distribution grid and helps them understand how the data sensed by the sensor boxes in the rooms change, based on their behaviours.

The design of such activities is meant to boost the personalization factor in GAIA class activities, increasing the engagement of the participants. It also helps to make students understand quicker the concepts related to GAIA, since essentially after completing some simple GAIA "interactive installations", they can see in almost real-time the effect of their behaviour being displayed on something that they have assembled themselves. This aspect is also tied to the reality of having a range of educational levels being addressed by GAIA: such activities can be fairly simple, e.g., following a clearly defined set of premade steps to assemble an installation, or can be more complex, e.g., by letting technical high school students customize and insert their own changes to the hardware or software being used.

We have until now designed three sets of exercises:

- An introductory course for the students to understand and familiarize with the building blocks of an electrical circuit and how they can use the conductive pens to draw an actual circuit on paper and use it.
- A course that provides a rolling visualization of the parameters measured by the sensor box (i.e., temperature, relative humidity, luminosity and sound).
- A course that provides a real time visualization of the power consumption inside the building in the three phases of the installation or in three different classrooms of the building

[5] http://wiki.seeed.cc/GrovePi_Plus/
[6] https://www.circuitscribe.com/

(whatever is available in each specific case).

To gather the data, we have developed a set of Python scripts that are able to interact with the services of GAIA, and more specifically with the REST HTTP API of the Data Storage Service.

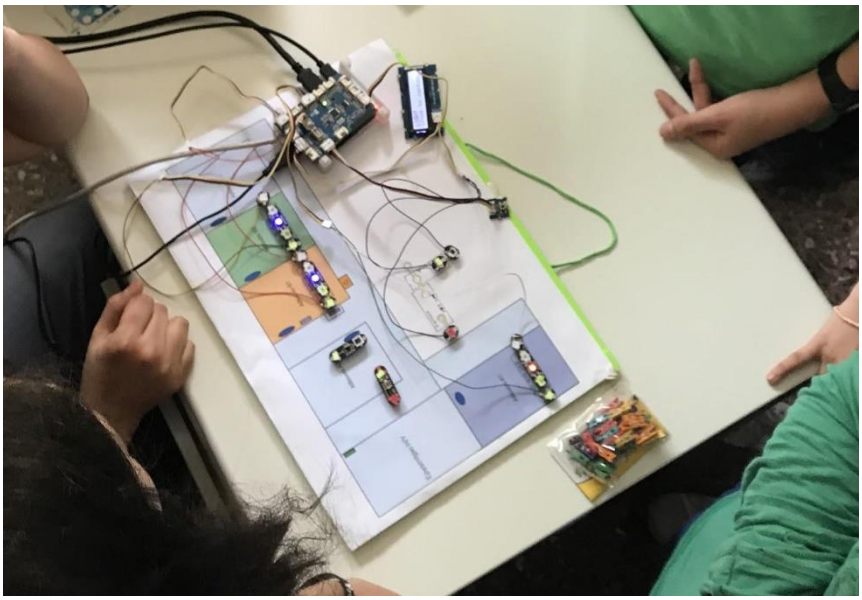

Figure 22 Snapshot from an educational lab kit-based class activity. Students use sensors and electronic components to visualise readings from the GAIA infrastructure. The circuit connections are performed by the students using cables and conductive ink on top of a printed floor map of their building.

9.2.2 Participatory Sensing Scenario

In the second scenario, we offer the possibility for the students and teaching staff to extend the infrastructure of their school using a Raspberry Pi device and some Grove sensors. As part of the educational kit, we offer tutorials about how to register a new Participatory Sensing device using the BMS UI, and then input the information of this sensor to the Raspberry Pi and feed data from its sensors to the GAIA backend as Participatory Sensors (see Section 4.2.3). The data from these sensors can be observed in the BMS UI application or using a second Raspberry Pi device implementing the first scenario. While the options we currently offer are limited due to the prerequisite use of Grove sensors, we believe that it is possible for the students and the teaching staff to use any other sensor compatible with the Raspberry Pi board, as there are many available tutorials for such processes in the Web. From our own experience so far, through the workshops organized and the mini-trials conducted in schools, we have seen some initial interest from the educators' side with respect to implementing such aspects, since several schools have already in their possession similar hardware that could potentially be utilized for such class activities.

10 Sequence diagrams for main GAIA processes

In this section, we provide some examples of interaction between WP2 components and also WP3 applications in relevant GAIA processes, namely:

- Recommendation generation and dissemination.
- Sensors data visualization.
- Participatory Sensing.
- Third-party Application' s access to GAIA Platform services

10.1 Recommendation generation and dissemination

During its workflow, the Recommendation Engine uses the services provided by a number of WP2 components and delivers the produced recommendations to the BMS UI. The sequence diagram in Figure 23 shows an example of such interactions (according to the execution flow described in Section 8.

First, the Recommendation Engine accesses the Authentication and Authorization service to request the token for accessing GAIA protected resources. At any moment in the workflow, a BMS UI client can connect to the WebSocket channel to listen for possible recommendations.

The Engine interacts with the Data Storage component to retrieve the id of sensors managed in the GAIA platform and periodically requests their latest measurements and building information (e.g., school opening and closing times, laboratory schedule). These values are used to periodically check if the rule condition is met (for the sake of clarity, the diagram shows the flow for one rule). Rules are periodically executed and if a rule condition is verified, the corresponding action is performed, by delivering the related notification over the WebSocket channel to the connected applications.

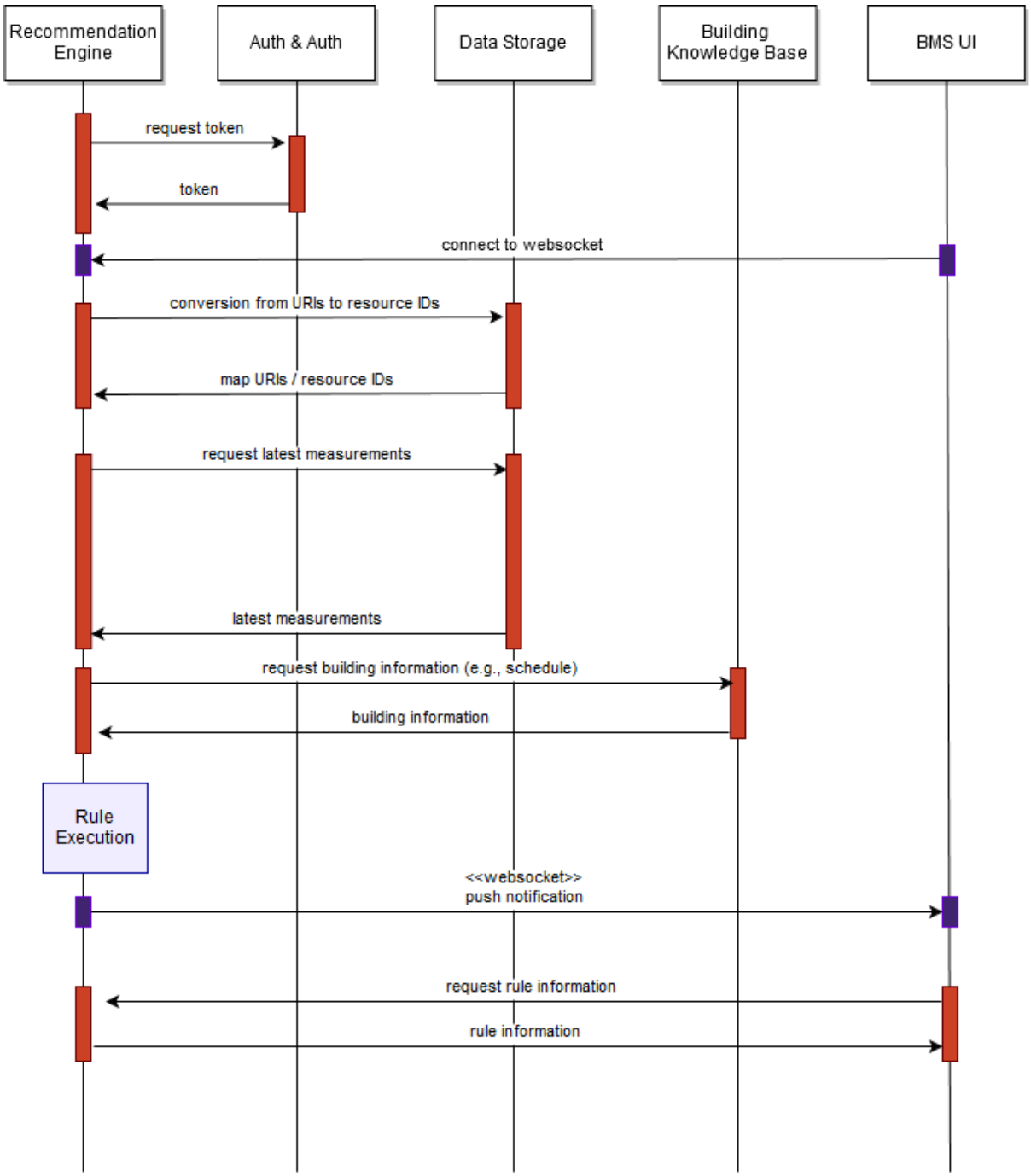

Figure 23 Rule execution and recommendation generation and dissemination

10.2 Sensors data visualization

To enable the visualisation of the data received from sensors, the sequence of actions shown in Figure 24 Visualisation of data from real sensorsFigure 24 takes place.

First, the Building Manager System UI (WP3) accesses the Authentication and Authorization service to request the token for accessing GAIA protected resources. Then, it requests the list of buildings which the current user is eligible to access. Once the user retrieves back the list of buildings, he selects from the UI the information per building. Based on this information, the user sets the criteria that will be

used to define the data set to be presented. These criteria are passed to the data storage entity which provides back the data that are then visualised by the UI.

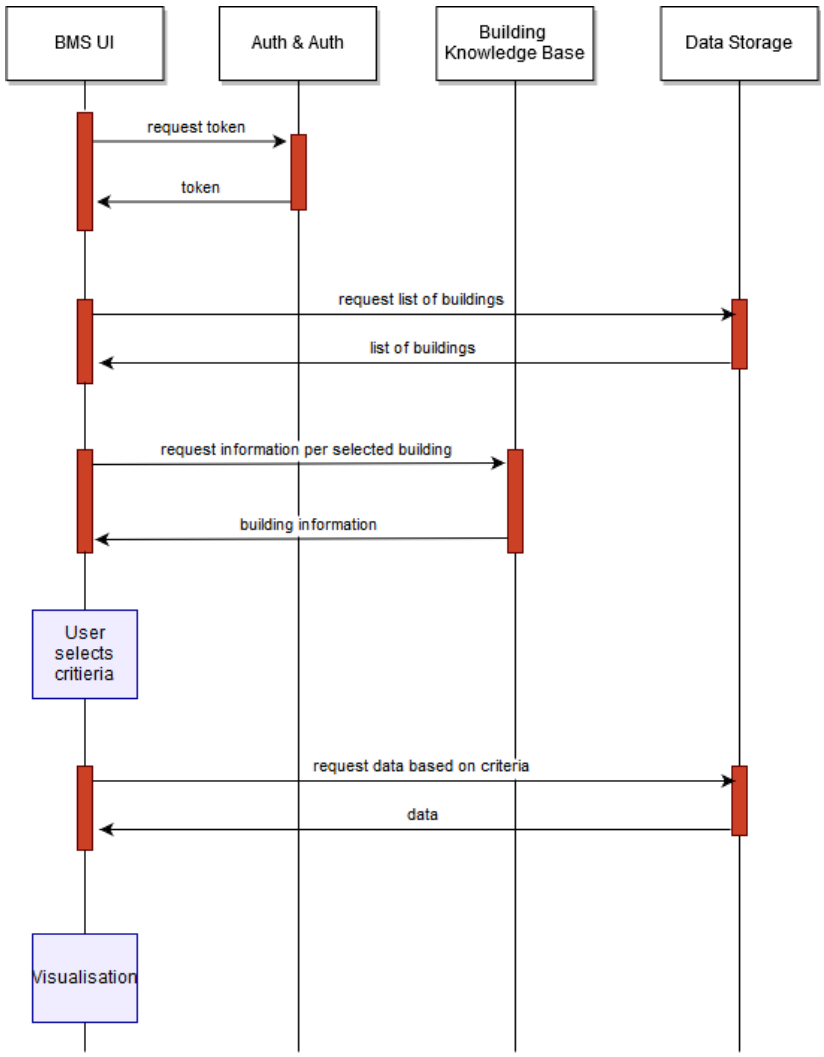

Figure 24 Visualisation of data from real sensors

10.3 Participatory Sensing

In the case of participatory sensing, the following sequence diagram takes place for the upload and visualisation of data automatically collected through the mobile application exploiting the luminosity sensor available in mobile devices.

Figure 25 shows the uploading and visualisation of data from virtual sensors. First, the Building Manager System UI accesses the Authentication and Authorization service to request the token for accessing GAIA protected resources. Then it requests the list of buildings which the current user is eligible to access from the data storage entity. Once the user retrieves back the list of buildings, he then retrieves from the data storage the list of virtual sensors or creates one. The “virtual sensor” concept is used in WP3 where the user/building manager defines an entity/quantity which can be physical or any other and the users/member of community of the building are eligible for inserting

measurements/values. A realistic example could be the temperature in a classroom which is not connected with the GAIA service infrastructure. The building manager can create using the BMS such a virtual sensor and the teachers and students can insert the readings they observe. Thus, the last two messages include the data inserted by the users and the response contains the measurement data inserted up to that moment for the virtual sensor at hand.

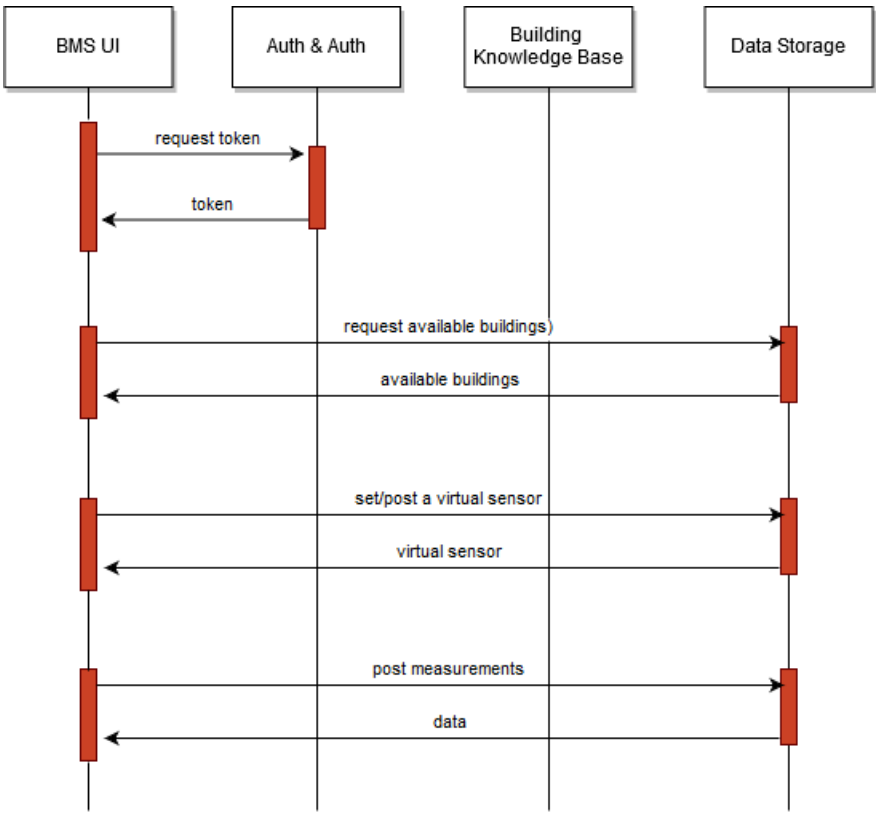

Figure 25 Uploading and visualisation of data from virtual sensors

Figure 26 shows the uploading and visualisation of data collected automatically from mobile devices. First, the GAIA mobile application accesses the Authentication and Authorization service to request the token for accessing GAIA protected resources. Then it requests the list of buildings which the current user is eligible to access from the data storage entity. Once the user retrieves back the list of buildings, he then selects a building and the information about the virtual sensors of the buildings is received back (marked as “building” in the following figure). The user selects auto-luminosity gathering (exploiting the existence of luminosity sensors in the mobile devices) and the reading from this sensor is uploaded to the GAIA infrastructure regularly (every 10sec in the current version).

Once the user selects to stop automatic upload, the Data Storage component provides back the measured values for this virtual sensor for the time span that the user is prompted to select. These measured values are referred to as “data” in the figure, while the message passing the time span selected by the user has been omitted, in order to keep the figure as simple as possible.

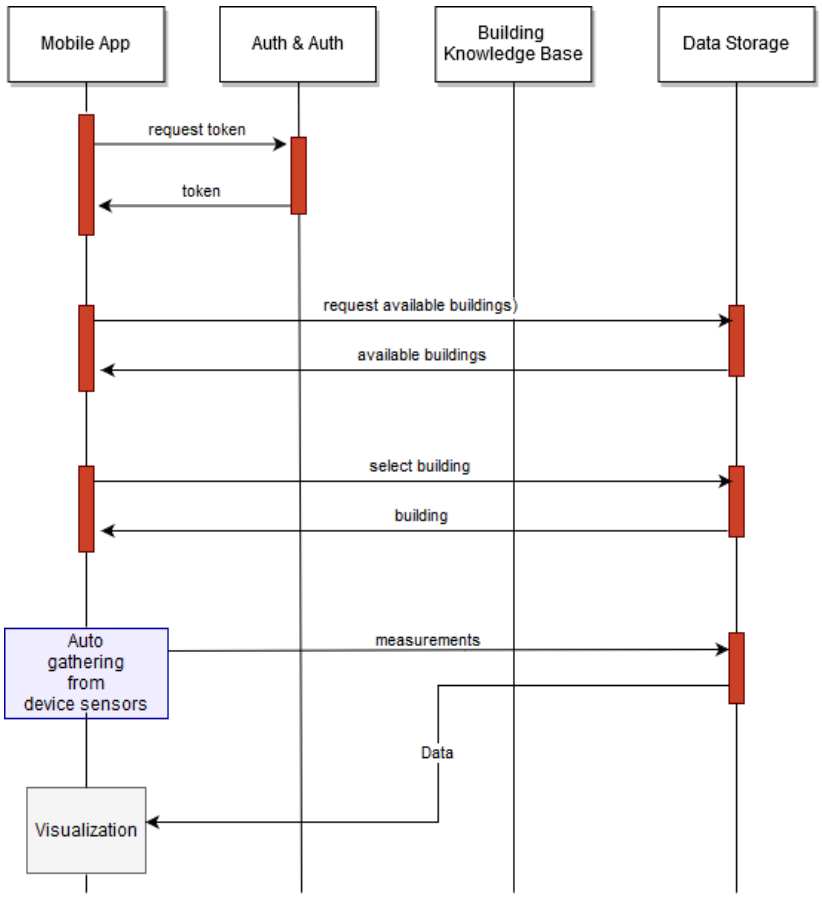

Figure 26 Uploading and visualisation of data collected automatically from mobile devices

10.4 Third-party Application' s access to GAIA Platform services

The GAIA service platform is not strictly tied to GAIA's applications and can be used to produce additional applications on top of it. Therefore, it is possible to develop third-party applications that implement workflows similar to the ones shown in the previous diagrams. Figure 27 shows an example of a possible workflow that may be supported by a third-party application, leveraging the APIs exposed by the GAIA software module.

First, the application accesses the Authentication and Authorization service to request the token for accessing GAIA protected resources. Then it requests the list of buildings which the current user is eligible to access from the data storage entity. The user then selects one of the building and requests to access anomalies detected within a given time range. This request is forward to the Analytics Module, thus triggering a request to the Data Storage for gathering data needed by the Anomaly Detection process. The results, including possible anomalies, are sent to the application and shown to the end-user. The end-user may annotate one or more anomalies for later retrieval. This annotation is stored as an event related to the building in the Building Knowledge Base.

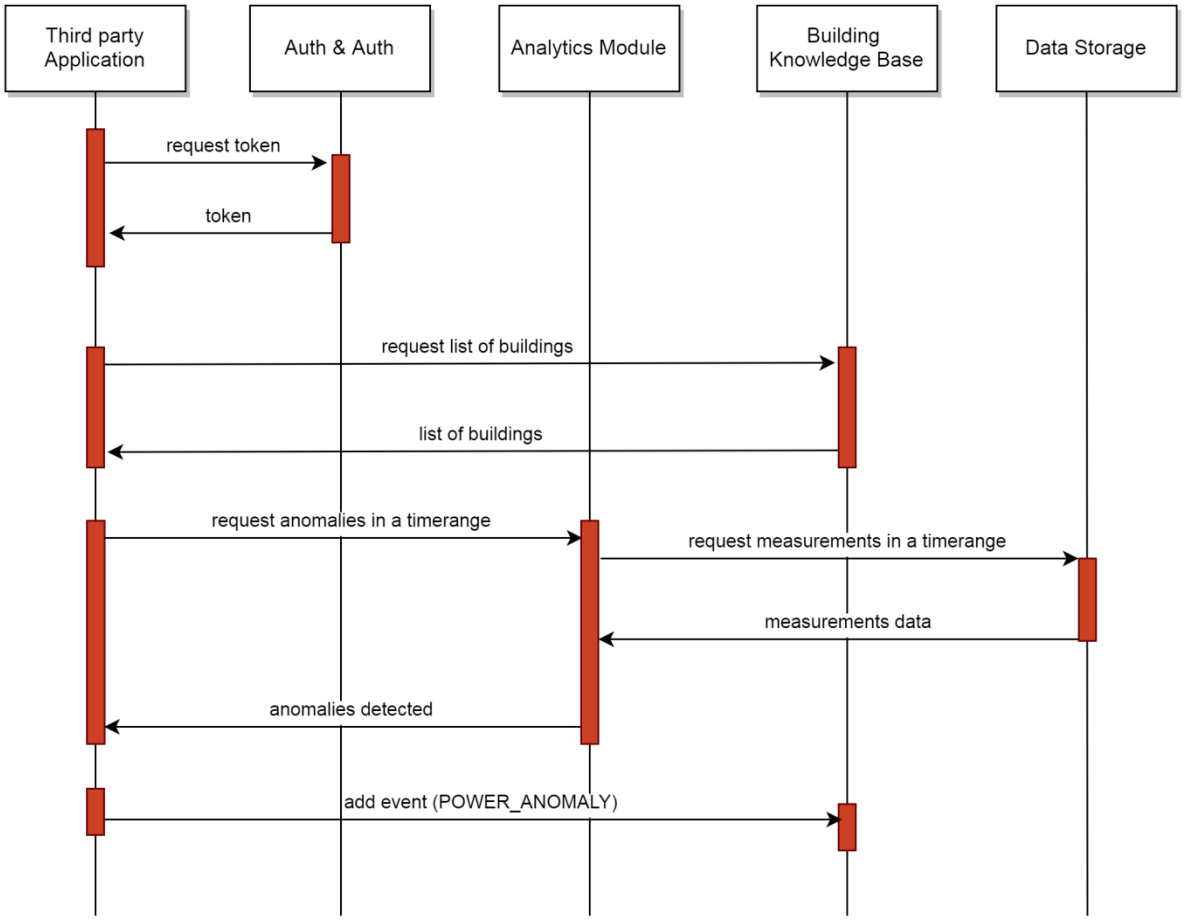

Figure 27 A third-party application accessing some GAIA services

11 Conclusions

This document presented the intermediate outcomes of WP2 and is directly to Milestone 3 of the project, “MS3 – Start of Trials”. The initial version for the GAIA infrastructure software is released simultaneously with the initial release of the WP3 applications. We presented here the overall architecture of the Service Platform and the GAIA conceptual model. Each main functional block of the architecture has been described, by presenting requirements, design and implementation details. At the end of the document, a section is devoted to describe how WP2 components interact with each other and with WP3 applications in some main GAIA processes.

In the coming months, the software will be fine-tuned, as a result of more intensive tests, and to cope with possible issues emerging during the trials in the schools. As scheduled, the main trials phase for the project will begin in the coming months. Such activities will be reported in Deliverable D2.2 “Final Infrastructure Software”, on Month 24 of the project. With respect to future work to be done for the final version of the software, for each component we have the following directions:

Authentication & Authorization
Future steps for this service include an enhanced version of the service’s User Interface. The new version will include capabilities for inviting new GAIA users to the platform and an enhanced user dashboard, where each user will be able to review pending and completed assignments, as well as fully control his account.

Data Storage
Next releases of the Data API will include lightweight capabilities of the API responses in conjunction with advanced paging and sorting capabilities. Apart from this, new endpoints will be added providing advanced querying on the Data API. Finally, redesigning the data storage abstraction model towards a more comprehensive, efficient and simplified model is another future step.

Building Knowledge Base
In the next releases of the software, we plan to slightly integrate or modify the Building Knowledge Base schema, in order to better model all the relevant information associated with school buildings. Thus, we will take into account any feedback provided by WP3 applications and WP2 modules and verify whether improvements are possible or changes are needed.

Analytics
Future work concerning the Analytics module is going to revolve around the enhancement of the deployed algorithms by validating them against the growing GAIA dataset and information available in the platform. In particular, we plan to test the accuracy of the Anomaly Detection and Clustering modules as the amount of data becomes larger, and adjust the actual implementation to improve the performance wherever possible. We also plan to include new operations and statistics about the user mobility, leveraging the information about commuting stored in the Building Knowledge Base.

Recommendation Engine

We plan the following main two lines of activity towards the next release of the Recommendation Engine:

- testing and fine-tuning of the actual implementation of the software (the engine core and rule classes implementation).
- completing the customization of recommendation scenarios according to specific schools' characteristics and needs, in cooperation with WP4 and the instantiation of related rules with customized parameters (e.g., thresholds, suggestions, URIs, etc.)

References

[Balaras] C. Balaras, et al., Energy and other Key Performance Indicators for Buildings– Examples for Hellenic Buildings, Global Journal of Energy Technology Research Updates, 2014, 1, 71-89

[Cisco, 2014] Cisco Systems, The Internet of Things Reference Model. White Paper, 2014.

[Drools] Drools, https://www.drools.org/.

[Collina et al., 2012] M. Collina, G. E. Corazza and A. Vanelli-Coralli, "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC), Sydney, NSW, 2012, pp. 36-41.

[Estrin, 2010] D. Estrin *et al.*, "Participatory sensing: applications and architecture [Internet Predictions]," in *IEEE Internet Computing*, vol. 14, no. 1, pp. 12-42, Jan.-Feb. 2010.

[Fernandes et al., 2013] J. L. Fernandes, I. C. Lopes, J. J. P. C. Rodrigues and S. Ullah, "Performance evaluation of RESTful web services and AMQP protocol," 2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN), Da Nang, 2013, pp. 810-815.

[Fielding, 2000] R. Fielding *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, 2000.

[Fielding & Taylor, 2002] R. Fielding, R.N. Taylor, “Principled design of the modern web architecture”, ACM Transactions on Internet Technology (TOIT) 2(2), 2002, pp. 115-150

[FIWARE] FIWARE Project Web Site, https://www.fiware.org/

[FIWAREContextBroker] FIWARE Context Broker, URL: https://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker

[FIWARELab] url: http://help.lab.fiware.org/

[Fowler, 2016] M. Fowler, Microservices: a definition of this new architectural term, On-line, available at: http://www.martinfowler.com/articles/microservices.html, accessed on June 2016.

[GAIA1.1] GAIA Consortium, D1.1 GAIA Design

[GAIA3.1] GAIA Consortium, D3.1 Application Prototypes

[GAIA3.2] GAIA Consortium, D3.2 – Applications Initial Release

[Jess] Jess, http://www.jessrules.com/docs/71/

[Microservices Patterns, 2017] Microservices: Patterns for Enterprise Agility and Scalability], White Paper, url: https://keyholesoftware.com/2017/03/14/white-paper-published-microservices-patterns-for-enterprise-agility-and-scalability/

[Newman, 2015] S. Newman. Building Microservices. O'Reilly, 2015

[OAuth2] OAuth 2 Project. https://oauth.net/2/

[Orbeet] Orbeet Project, Deliverable D1.2, Specs of SEOR methodology and Enhanced Display Energy Certificates, October 2015.

[OrientDB]]http://orientdb.com/docs/2.2.x/OrientDB-REST.html

[RFC6749, 2012] The OAuth 2.0 Authorization Framework, RFC6749, Internet Engineering Task Force (IETF), ISSN: 2070-1721

[Richardson & Ruby, 2007] L. Richardson, S. Ruby *RESTful Web Services*. OReilly & Associates, 2007.

[Sparks Public Repo] url: http://artifactory.sparkworks.net/nexus/repository/maven-snapshots/

[SSN Ontology] Michael Compton et a., The SSN ontology of the W3C semantic sensor network incubator group, Web Semantics: Science, Services and Agents on the World Wide Web, Volume 17, 2012, Pages 25-32, ISSN 1570-8268, http://dx.doi.org/10.1016/j.websem.2012.05.003.

[Stallings, 2015] Foundations of Modern Networking, Pearson Education, Addison-Wesley, 2015.

[WildFly] http://wildfly.org/

[Zuzac et al., 2011] I. Zuzak, I. Budiselic, G. Delac, "A Finite-State Machine Approach for Modeling and Analyzing RESTful Systems". J Web Eng 10(4), 2011, pp. 353-390.