Seventh Framework Programme

Call FP7-ICT-2013-10

Project Acronym:	**S-CASE**
Grant Agreement N^{o}:	**610717**
Project Type:	**COLLABORATIVE PROJECT**
Project Full Title:	**Scaffolding Scalable Software Services**

D6.3.1 Pilot case specifications and models

Nature:	**R**
Dissemination Level:	**PU**
Version #:	**1.0**
Date:	**5 May 2015**
WP number and Title:	**WP 6 – Evaluation (pilot case design and evaluation metrics)**
Deliverable Leader:	**DELPHIS**
Author(s):	**Marina Stamatiadou (DELPHIS)**
Revision:	**INS, FLEX**
Status:	**Submitted**

Document History

Version[1]	Issue Date	Status[2]	Content and changes
0.1	10 December 2014	Draft	ToC
0.2	15 December 2014	Draft	Updated ToC
0.3	11 March 2015	Draft	ToC and document structure finalization
0.4	10 April 2015	Draft	Section 2: class diagrams, activity diagrams, interaction diagrams Annex I
0.5	17 April 2015	Draft	Section 2: description of packages
0.6	20 April 2015	Draft	Section 3 Section 4
0.7	22 April 2015	Draft	Section 3 finalization Section 5 Annex II
0.8	28 April 2015	Peer reviewed	Updated version after INS peer review
0.9	29 April 2015	Peer reviewed	Updated version after FLEX peer review
1.0	5 May 2015	QB reviewed	Updated version after QB review

Peer Review History[3]

Version	Peer Review Date	Reviewed By
0.7	28 April 2015	INS: Davide Tosi
0.7	28 April 2015	FLEX: Alasdair Innes

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 610717.

[1] Please use a new number for each new version of the deliverable. Use "0.#" for Draft and Peer-Reviewed. "x.#" for Submitted and Approved", where x>=1.Add the date when this version was issued and list the items that have been added or changed.
[2] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted and Approved.
[3]Only for deliverables that have to be peer-reviewed

Table of contents

List of Figures

List of Tables

Abbreviations and Acronyms

Abbreviation	Explanation
API	Application Programming Interface
BMS	Business Management System
CRUD	Create Read Update Delete
DA	Data Analytics
DC	Data Collection
DoW	Description of Work
DVUI	Data Visualization & User Interaction
FR	Functional Requirement
IoT	Internet of Things
MVC	Model View Controller
NFR	Non-functional Requirement
PaaS	Platform as a Service
REST	Representational State Transfer
SaaS	Software as a Service
SE	Software Engineering
UI	User Interface
UML	Unified Model Language
URI/URL	Uniform Resource Identifier/Locator
WS	Web service
WTC	WatchTower Cloud

The S-CASE Project Consortium groups the following organizations:

Partner Name	Short name	Country	Country
Aristotle University of Thessaloniki	AUTH	Greece	GR
Centre for Research and Technology Hellas / Information Technologies Institute	CERTH	Greece	GR
University of Edinburgh	UEDIN	United Kingdom	UK
Università Insubria	INS	Italy	IT
Flexiant Limited	FLEX	United Kingdom	UK
Akquinet tech@spree GmbH	ATS	Germany	DE
Ericsson Nikola Tesla d.d.	ENT	Croatia	HR
DELPHIS – D. Daskalopoulos S.A.	DELPHIS	Greece	GR
Engineering Ingegneria Informatica SpA	ENG	Italy	IT

Executive Summary

This document is the DELPHIS pilot (**WTC**) specification document and it aims to provide a detailed work plan for the WTC scenarios. Its focus is on the detailed design of the WTC pilot by giving a thorough description of specifications, proposed architecture, interfaces and business cases. Module interactions for key processes illustrate the information flow between the modules. A detailed list of functional and non-functional requirements for each module is given and the corresponding mapping to the services to be deployed is presented.

1 Introduction

This document is the deliverable D6.3.1 and describes the work performed for the task T6.3: *Extraction of pilot case specifications and models*. It serves as a detailed design plan for modelling the DELPHIS pilot **WTC** using the S-CASE methodology while giving a thorough description of the pilot module functionalities.

WTC is not another traditional, customer-driven software for energy management. It is responsible for collecting, handling, processing and exploiting data coming from heterogeneous devices, 3rd party providers or simple users. Generic or building/equipment-specific FDD techniques are being executed on this data to detect faults and erroneous behaviours in order to produce valuable information. Reporting and alert mechanisms are responsible for notifying involved parties regarding monitoring results. Apart from the ready-made analytics generated by the its back-end engine, WTC serves also as a framework for building energy management applications on top of its core modules. Software developers will be provided with a well-defined API in order to make use of the existing WTC services and develop dedicated and customized service compositions. 3rd party services by external providers can be also integrated with such customized solutions.

WTC's innovation is based on the following set of characteristics which also become our success factors:

- Built on top of the SaaS model, it simplifies deployment, reduces customer acquisition costs and enables seamless integration with external systems
- It exposes its core features, functionalities and ready-made analytics as standalone service modules which can be directly consumed by end-users or 3rd party applications
- It offers the ability to compose custom analytics services
- It is hardware-agnostic, meaning that it can be adopted by different hardware providers

Developed exploiting the S-CASE platform and following its modular logic, it becomes fully scalable. Its functionality can be expanded to support more features and fit even more business needs without the need of major re-engineering. This is why WTC is a powerful platform addressing the needs of both end-users who need easy-to-use and customizable interfaces and developers who need portable and well-defined services to integrate with their applications. And this is how S-CASE will become the infrastructure foundation for making WTC a scalable and interoperable framework in the energy management domain.

1.1 Objectives of WP6

The main objective of WP6 is to evaluate the S-CASE platform according to pilot-specific KPIs which will validate the core project concepts and to demonstrate the feasibility of the S-CASE platform through the three pilot projects, which represent real -business world- scenarios. Pilot cases will make extensive use of the newly initialized *YouRest* ecosystem of services and introduce users to the new SE concept of rapid software development.

To do so, at first, a market analysis is being conducted (Task 6.1) in order to identify and explore existing solutions and tools in the world of multimodal user requirements, as well as potential users and actors who will benefit from the innovative results of the project. Additionally, all consortium partners contribute in creating a list of all possible exploitable items while pilot partners have already specified their envisioned cases and have already outlined their work plan for Tasks 6.1, 6.2 and 6.3. The three pilot cases will evaluate the whole project concept by adopting the new SE process implied by S-CASE.

Specific goals of the work package are:

- to establish a common strategy and planning for evaluating the S-CASE project,
- to perform detailed market assessment identifying technological trends, key players, market barriers and drivers,
- to conduct market analysis and research on state of the-art existing tools & solutions in order to place project developments into the market and issue business scenarios for project results market penetration,
- to perform an extensive evaluation of the project outcomes, considering both technical aspects and user experience through a series of KPIs and evaluation metrics,
- to define and describe WTC as one of the envisioned pilot projects to be deployed and devise its conceptual model,
- to decide on common KPIs and evaluation metrics to measure the effective improvements obtained using S-CASE environment over a contemporary SE approach,
- to identify generic user-oriented and system related, functional, as well as non-functional requirements for the realization of each WTC architecture modules,
- to design the overall architecture of the WTC infrastructure,
- to define protocols for data flow and common output for the WTC pilot platform,
- to generate a fully-functional version of WTC software for the end users to exploit,
- to validate the S-CASE platform

The work in WP6 is divided into five tasks, and this document is a result of work performed within task T6.3 for the goals listed above in its first iteration.

1.2 Methodology

The description of task T6.3 in the project Description of Work gives the description of the goals:

> *After the preparation phase of the pilot cases, a detailed work plan will be provided for each scenario. Given that pilot case requirements are identified early in the project (T.1.5) and are used as input for the early S-CASE platform release, Task 6.3 will focus on the detailed design of the pilots by providing thorough specifications, proposed architecture, interfaces and business cases. All this information will be provided in any of the formats supported, i.e. textual descriptions, UML diagrams, etc., in order to test WP2 and WP3 modules.*

The goals of the task can be summarized as follows:

- to devise WTC conceptual model through real-world business cases,
- to identify generic user-oriented and system related, functional, as well as non-functional requirements for the realization of each WTC architecture modules,
- to design the overall architecture of the WTC infrastructure,
- to define protocols for data flow and common output for the WTC pilot platform

The work in this task will continue to be completed with an updated version of this deliverable. Following the T1.5: *Pilot requirements*, the pilots requirement-related processes are split into two phases:

Phase 1: preliminary analysis of:

- the envisioned system to be developed
- involved actors
- functional requirements

thus, a preparatory pilot phase resulting in the WTC pilot high-level requirements for involved actors, described in early D1.3.

Phase 2:

- extensive requirements analysis (multimodal requirements according to S-CASE model)
- WTC pilot specifications (architectural elements, interfaces and I/Os, module interactions, data flow protocols, etc)
- service definition and categorization, mapping to final WTC pilot requirements and modules

According to the DoW, the detailed and final WTC pilot requirements will be part of final D1.3, along with the corresponding mapping between them and the services to be deployed. WTC pilot specifications will be documented in D6.3.1

1.3 Scope of this Deliverable

This deliverable provides an overview of the WTC pilot specifications and models. Target readers of D6.3.1 are the software developers of the WTC platform (back-end, front-end, UI) as it is going to serve as the basis for software implementation. The architecture described in this document is a result of the comprehensive requirements analysis conducted (Task 1.5) and is given in textual and schematic format. In more detail, this document describes:

- the architectural layers the WTC platform consists of and their respective modules,
- the interaction between the modules,
- the RAML specifications for the services to be deployed,
- a first UI design approach

1.4 Document structure

This document is structured as follows:

- Section 2 presents an overview of WTC pilot architecture, describes the modules, the context they operate in, their role, and lists the requirements for each module. The section concludes with an overview of module interactions and activity diagrams which show the information workflow.
- Section 3 gives a detailed list of all the services to be deployed and/or integrated within the WTC pilot as well as a mapping between these services and the modules they belong to. RAML specifications for the core architectural elements are also given within section 3.
- Section 4 presents the mock-up of the envisioned WTC pilot UI, based on the sketch provided in the WTC pilot paper prototype and in the final document of T1.5.

The document concludes in Section 5 and two annexes documenting WTC's functional and non-functional requirements and the mapping between functional requirements and the architectural modules.

2 Overview of the architecture

This section presents an overview of the architecture as an introduction to the WTC pilot modules and corresponding specifications. The WTC concept as a whole is outlined in Figure 1.

As shown in Figure 1, WTC employs two (2) different interfaces: the SaaS and an API. In both cases data can be sent to the system by a building's BMS, after passing a feeder that is responsible for parsing the incoming information (dynamic data). On the other hand, building users (i.e. building energy managers/admins) are also able to directly insert data into the system (static data/building attributes). This data is feeding the *data collection* layer which contains a set of services for data validation, dedicated computations according to each device's needs and specifications and data storage. *Data analytics* layer is the one that contains services for data analysis processes, knowledge discovery and data-driven FDD techniques of the WTC. *Data visualization & User interaction* is the next layer which contains those services that are related to the end-user and/or developer who interact with the system (e.g. tickets/alerts/notifications, charts, graphs, etc.). These three (3) service layers can be orchestrated by a **middleware** called *workflow orchestrator* in order to expose the SaaS interface to the end users.

Going one step further, another **middleware** called *service orchestrator* is responsible for handling the communication between the three above-mentioned layers and the *custom service composition* layer. This means that the WTC API is exposed to the developers' community who can select which services to use and in what execution order, so that a custom composition is created.

Finally, 3rd party services from external providers can also be parts of ready-made or custom service compositions.

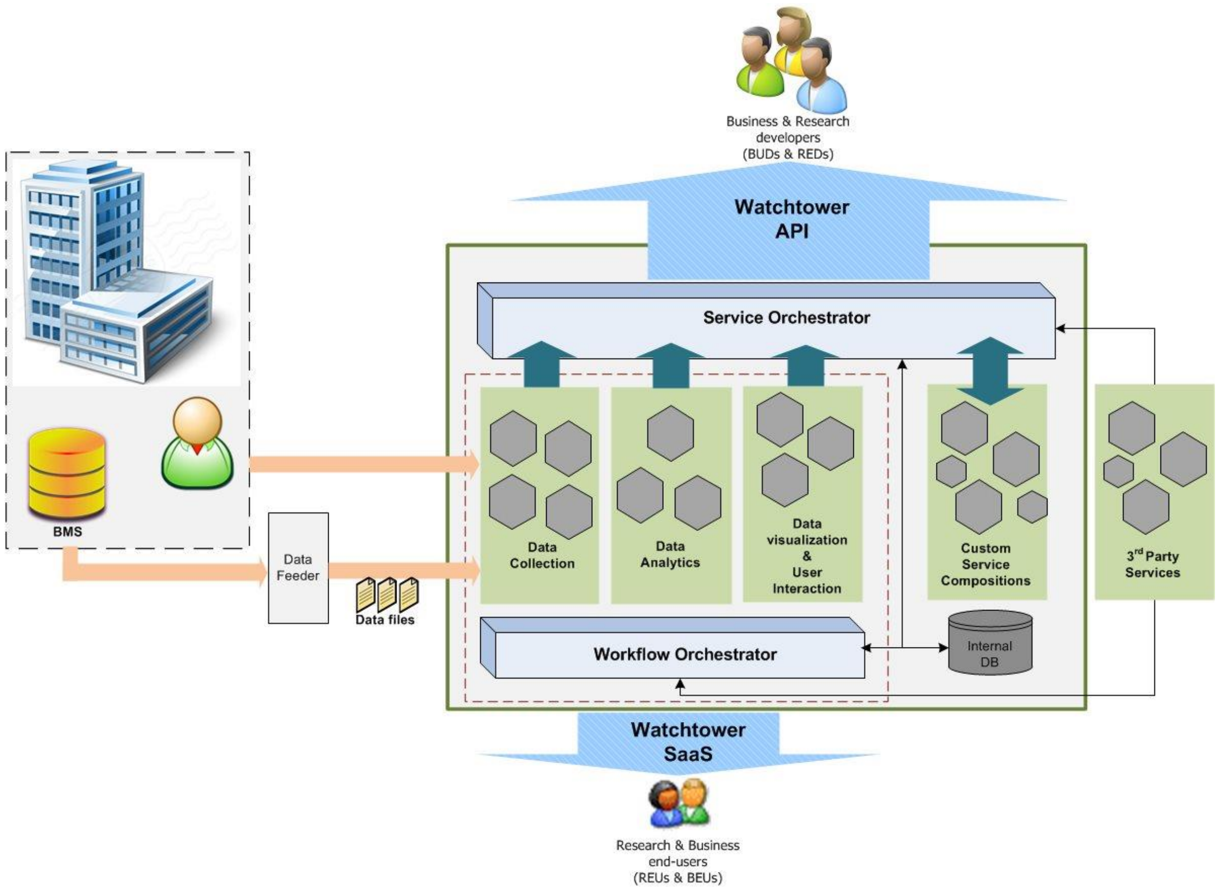


Figure 1: The WTC concept

From this concept and in order to logically group the underlying software components, we have identified four (4) distinct software packages shown in Figure 2.

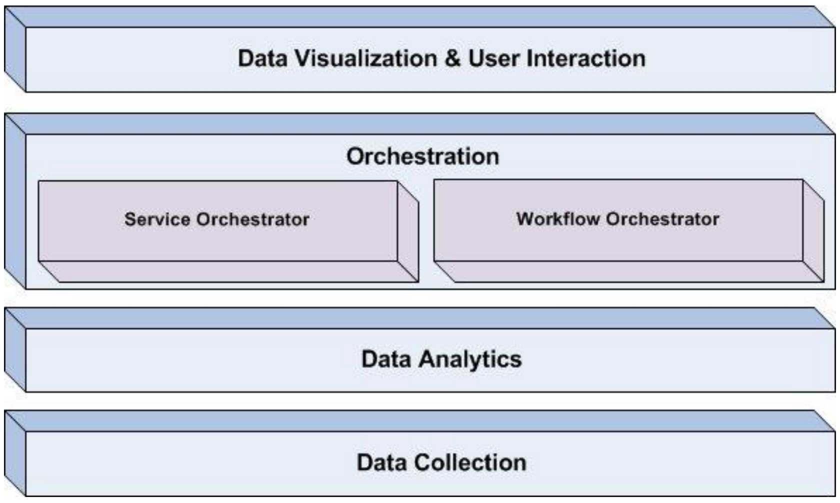


Figure 2: Modular WTC architecture & layers

Each layer is based on the MVC architectural model, as shown in Figure 3 and the resulting architectural model is better shown in Figure 4. At MVC's core is the idea that the model (the data) and the view (the user interface) should be decoupled and isolated. It separates the modelling of the domain, the presentation, and the actions based on user input into three separate classes:

- **Model:** The model manages the behaviour and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
- **View:** The view manages the display of information.
- **Controller**: The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.[4]

[4] https://msdn.microsoft.com/en-us/library/ff649643.aspx

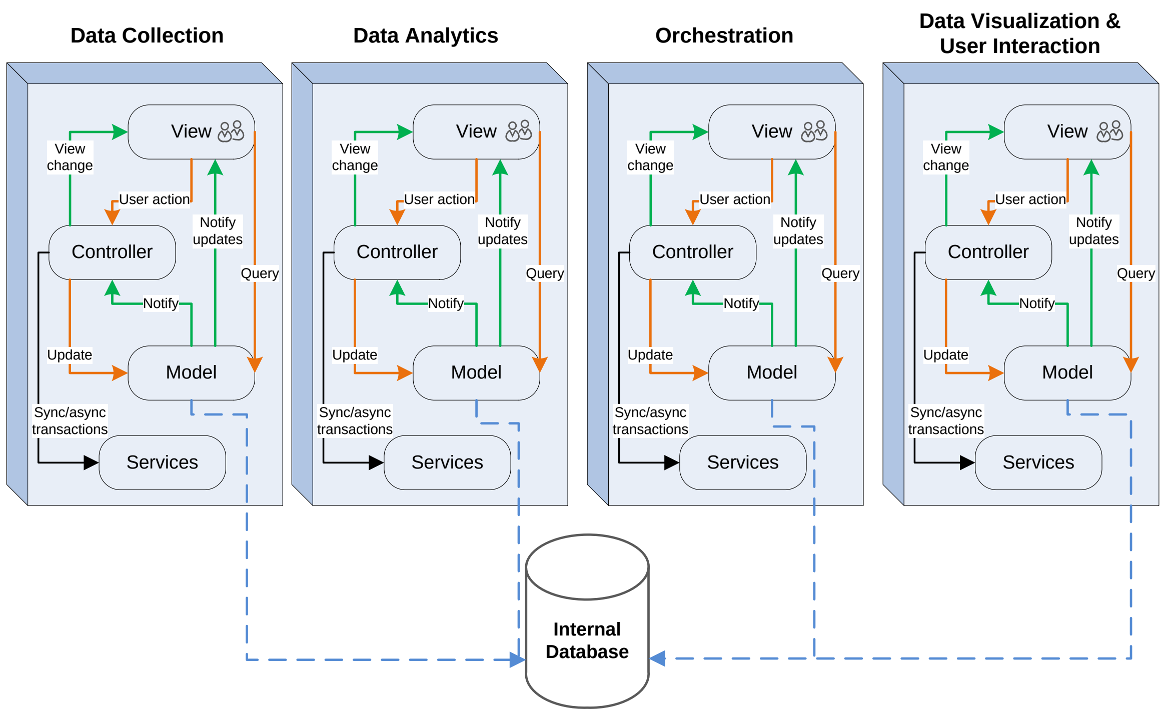


Figure 3: MVC model architecture

This essentially good idea allows each layer to be optimized and tested on its own. It also allows the secondary benefit of easier refactoring in the future, in case one of the layers needs to be replaced with a different technology, a not uncommon requirement[5]. So, by building the WTC as a MVC-based platform, we also encapsulate the three main benefits of the MVC architecture: separation of concerns, scalability, testability.

In the rest of this section we are going to give a detailed description and representation of the WTC architecture. The WTC system is divided into three layers (models, controllers/services, views) with each layer representing the corresponding MVC layer (Figure 4). Each of the three layers consists of software packages which are logical groups of software entities as defined in Figure 2.

[5] http://threecrickets.com/prudence/manual/mvc/

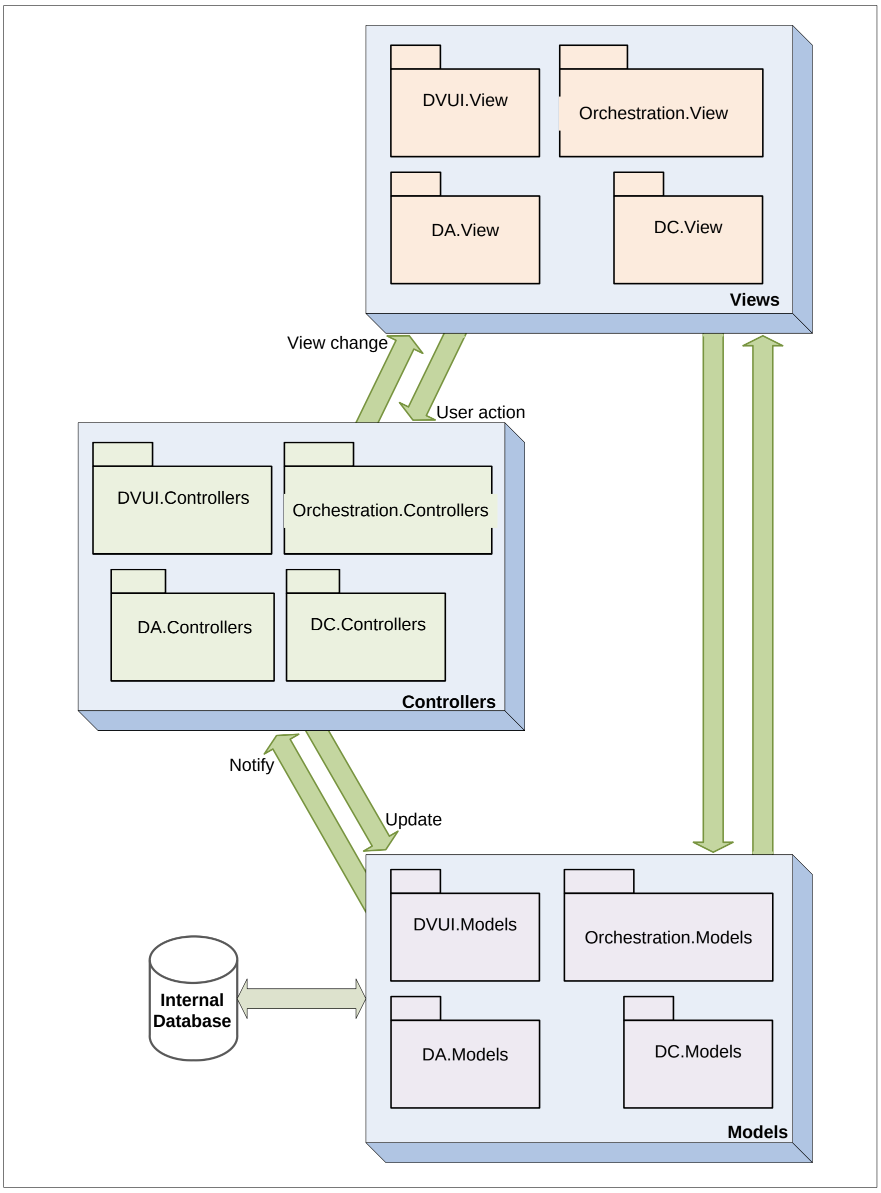


Figure 4: WTC MVC layers

2.1 Models

This layer contains all the models of the WTC and is the core of the system. It maintains and manages the state and data that the application represents. It responds to the request from the view and to the instructions from the controller to update itself. The models included in this layer are grouped into four packages, following a modular structure: Data Collection (DC), Data Analytics (DA), Orchestration and Data Visualization and User Interaction (DVUI).

2.1.1 Package DC.Models

This package contains all the models that are related to the data collection category, as shown in Figure 5. Models in this package handle data that describe all the core entities of the WTC.

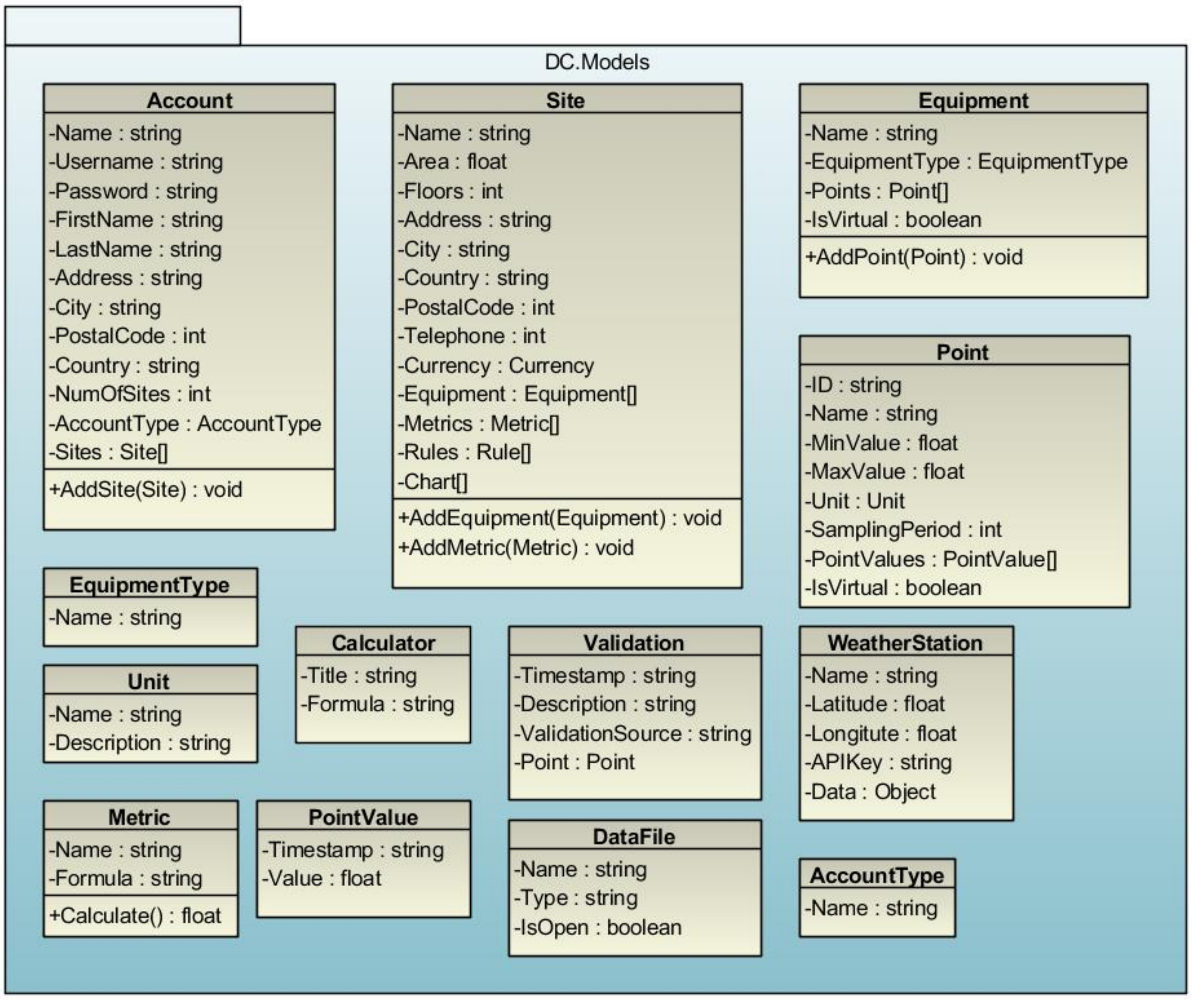


Figure 5: Data Collection models

Class description

- *Account:* the class that models user related data, namely with respect to logging in, as well as with respect to the type of user and the sites that the user should have access to.
- *AccountType:* a predefined set of *Account* types, defining the business domain each site is related to (e.g. Super markets, Banks, etc).
- *Site:* the class that models a building that may have one or more installed metering equipment. Apart from the general information (location, country, no of floors, etc) *Site* models hold information on the *Equipment* that is installed in the building, the *Metrics* measured/calculated, and finally the *Charts* and the *Rules* that apply to the specific building measurements.
- *Equipment:* it models the devices that are installed in a *Site.* For each device, the equipment type is defined. Additionally, given that each device may have more than one sensors gathering data, a number of *Points* is assigned to each *Equipment* instantiated.
- *EquipmentType:* a predefined set of *Equipement* types, defining the device type monitored (e.g. Energy /Smart meters, Refrigeration devices, HVACs, etc).
- *Validation:* it models and stores the validation results, as defined by the set of *Validation* algorithms that run for every incoming building data file.
- *Point:* it models a sensor that collects data. Apart from the general *Point* information and information regarding the measurement units, the sampling period and basic statistics, the collected measurements, are gathered and stored.

- *PointValue:* a class modelling the data measurements, represented as a *Timestamp-Value* tuple.
- *Metric:* a model that calculates simple and composite metrics based on the *Formula* defined and the *PointValue* instances.
- *Unit:* a predefined set of units, employed for unambiguously defining *Metric* instances.
- *WeatherStation:* it models weather data for a specific *Site,* as extracted from a 3rd party weather Web service.
- *DataFile:* a class that models data files containing *Point* measurements.
- *Calculator:* a model that calculates simple and composite metrics and measurements based on the *Formula* defined in the *DataFile* instances.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These model classes will be directly generated through (or by) the S-CASE engine.

Requirements

Identifier	Title
REQ_DC_1	File parsing
REQ_DC_2	Non-existing meter
REQ_DC_3	Out of limit values
REQ_DC_4	Invalid numbers
REQ_DC_5	Invalid numbers format
REQ_DC_6	Invalid time format
REQ_DC_7	Invalid sampling period
REQ_DC_8	Missing values - gaps
REQ_DC_9	Validation information
REQ_DC_10	Time series
REQ_DC_11	Power consumption
REQ_DC_12	Energy consumption
REQ_DC_13	Data values for not existent meters
REQ_DC_14	Events
REQ_DC_15	Meter mapping
REQ_DC_16	Virtual meters
REQ_DC_17	Building attributes
REQ_DC_18	Building devices
REQ_DC_19	Value limit
REQ_DC_21	Insert incoming data
REQ_DC_22	Reject incoming data

REQ_DC_23	Store validation information
REQ_DC_24	Save static data
REQ_DC_25	Duplicates

2.1.2 Package DA.Models

This package contains the models that are related to the data analytics category, as shown in Figure 6. Models in this package handle data that describe the entities of the WTC that involve post processing of the data and fault detection mechanisms.

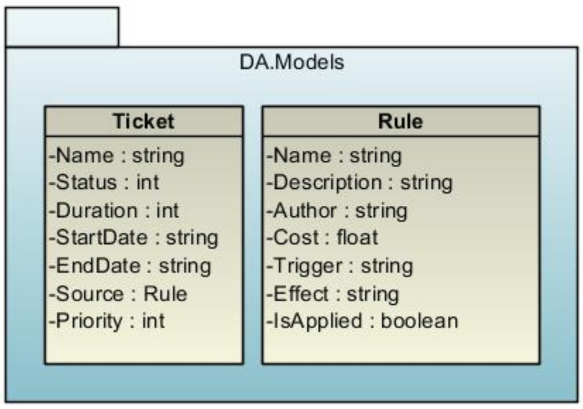


Figure 6: Data Analytics models

Class description

- *Rule:* the class that models the concept of a rule applied on a *Point* of an *Equipment,* installed on a specific *Site.* Besides the general data, a *Rule* comprises a *Trigger* and *Effect* section, as well as a *Cost* section.
- *Ticket:* whenever a *Rule* is applied and given that the *Trigger* condition is satisfied, a *Ticket* is generated. For each *Ticket,* the start and end date, along the source *Rule* and the ticket *Priority* are stored for future reference.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These model classes will be directly generated through (or by) the S-CASE engine.

Requirements

Identifier	Title
REQ_DA_1	Define metrics
REQ_DA_2	Calculate metrics
REQ_DA_3	Create rules
REQ_DA_4	Browse rules
REQ_DA_5	Apply rules
REQ_DA_6	Edit rules
REQ_DA_7	Delete rules

2.1.3 Package Orchestration.Models

This package contains the models that are related to the orchestration of services, as shown in Figure 7. Models in this package handle data that describe the possible service compositions as supported by WTC.

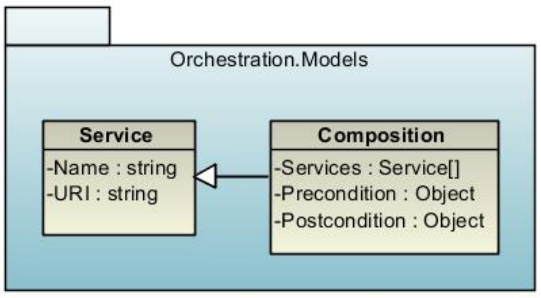


Figure 7: Orchestration models

Class description

- *Service:* the class that models and stores the *Name* and *URI* of each *Service* involved in a *Composition* of Services.
- *Composition:* the class that models an aggregation of Services, along with the *Precondition and PostCondition* for the *Composition.* Practically, the *Composition* implementation follows the S-CASE Storyboard approach, when it comes to modelling Simple and Composite storyboards.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These model classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DA_8	Service composition
REQ_DA_9	Browse services
REQ_DA_10	Test dataset
REQ_DA_11	Composition workflow

2.1.4 Package DVUI.Models

This package contains all the models that are related to data visualization and user interaction, as shown in Figure 8. Models in this package handle all composite structures that are projected to the users (end users & developers) of the WTC.

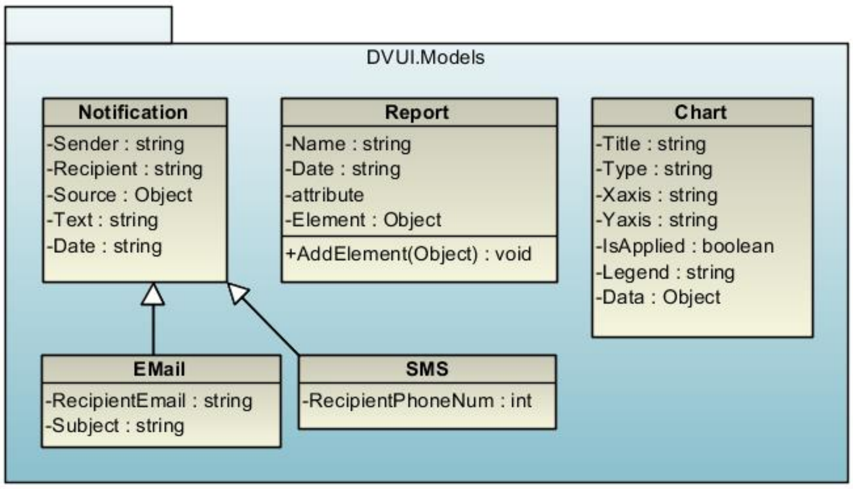


Figure 8: Data Visualization and User Interaction models

Class description

- *Report:* the class that models all information projected to the user with respect to a specific *Site.* It provides a dynamic dashboard compilation from objects of *Element* type that contain metadata (*Tickets, Metrics, Charts, Notifications,* etc)
- *Chart:* for each one of the *Points* selected and for user-specified periods, the user may generate charts (line, bar and pie charts) to be included in the *Report.*
- *Notification:* the class that models and stores any notification sent from the WTC to the users or from the WTC admins to the users.
- *Email/SMS:* classes that inherit from and implement concrete *Notification* objects.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These model classes will be generated by the S-CASE engine.

Requirements

Identifier	Title
REQ_DVUI_1	Generate ticket
REQ_DVUI_2	Send e-mail
REQ_DVUI_3	Generate report
REQ_DVUI_4	sms
REQ_DVUI_5	Create chart
REQ_DVUI_6	Delete chart
REQ_DVUI_7	Edit chart
REQ_DVUI_8	Save chart
REQ_DVUI_9	Browse charts library
REQ_DVUI_10	Compare charts
REQ_DVUI_11	Chart granularity

REQ_DVUI_12	Apply chart
REQ_DVUI_13	Export file

2.2 Controllers

2.2.1 Package DC.Controllers

This package contains all the controllers of the models defined in Section 2.1.1. The Data collection Controllers define all the functionality expected on the *DC.Models* that describe the core entities of the WTC. The *DC.Controllers* are depicted in Figure 9. As shown, each *DC.Controller* class comprises at least one *DC.Model* object, as well as a number of methods that implement the expected functionality. *DC.View* objects are defined only in the case where respective visual elements are provided to the user.

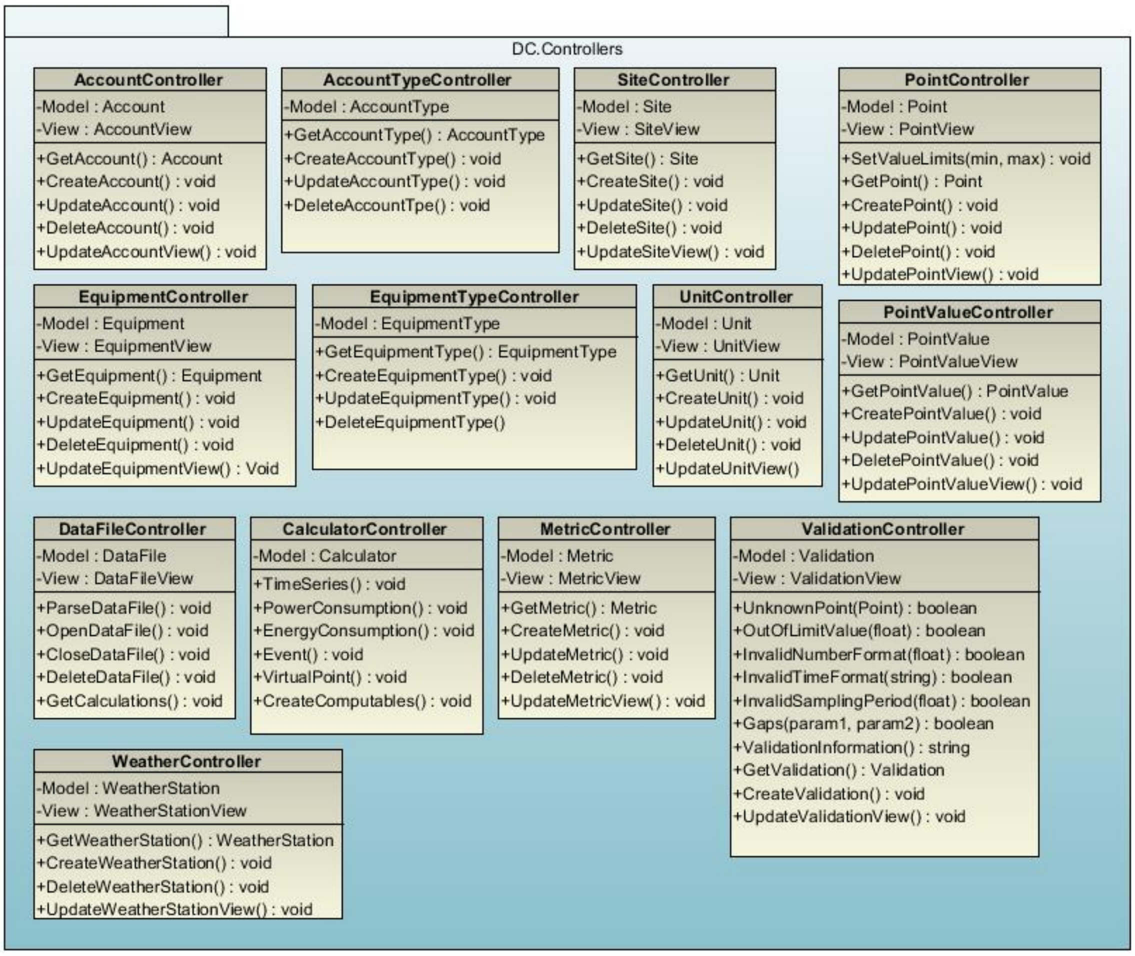


Figure 9: Data Collection controllers

Class description

- *AccountController:* the class that handles the CRUD functionality of *Account*, through the *CreateAccount(), GetAccount(), UpdateAccount(), DeleteAccount()* methods. *AccountController also* implements an *UpdateAccountView()* method, which updates the respective View class.
- *AccountTypeController:* it handles the CRUD functionality of *AccountType*, through the *CreateAccountType(), GetAccountType(), UpdateAccountType(), DeleteAccountType()* methods.

- *SiteController:* the class that handles the CRUD functionality of *Site*, through the *CreateSite(), GetSite(), UpdateSite(), DeleteSite()* methods. Additionally, *SiteController* implements an *UpdateSiteView()* method, which updates the respective View class.
- *EquipmentController:* the class that handles the CRUD functionality of *Equipment*, through the *CreateEquipment(), GetEquipment(), UpdateEquipment(), DeleteEquipment()* methods. Additionally, *EquipmentController* implements an *UpdateEquipmentView()* method, which updates the respective View class.
- *EquipmentTypeController:* it handles the CRUD functionality of *EquipmentType*, through the *CreateEquipmentType(), GetEquipmentType(), UpdateEquipmentType(), DeleteEquipment Type()* methods.
- *ValidationController:* the class that handles the CR (C:Create, R: Read) functionality of *Validation*, through the *CreateValidation(), GetValidation()*methods. Additionally, *ValidationController also* implements an *UpdateValidationView()* method, which updates the respective View class. The core of this Controller, though, is the definition of methods that are related to validation tasks performed on the data, and are related to the existence of a *Point,* the range/validation testing to the *PointValues* and the sampling periods defined.
- *PointController:* it handles the CRUD functionality of *Point*, through the *CreatePoint(), GetPoint(), UpdatePoint(), DeletePoint()* methods. Additionally, *PointController* implements an *UpdatePointView()* method, which updates the respective View class. Finally, a *SetValueLimits()* method is defined, to set the acceptable value range for a specific *Point.*
- *PointValueController:* the class that handles the CRUD functionality of *PointValue*, through the *CreatePointValue(), GetPointValue (), UpdatePointValue(), DeletePointValue()* methods. Additionally, *PointValueController* implements an *UpdatePointValueView()* method, which updates the respective View class.
- *MetricController:* it handles the CRUD functionality of *Metric*, through the *CreateMetric(), GetMetric(), UpdateMetric(), DeleteMetric()* methods. Additionally, *MetricController* implements an *UpdateMetricView()* method, which updates the respective View class.
- *UnitController:* it handles the CRUD functionality of *Unit*, through the *CreateUnit(), GetUnit(), UpdateUnit(), DeleteUnit()* methods.
- *WeatherStationController:* it handles the CRD functionality of *WeatherStation*, through the *CreateWeatherStation(), GetWeatherStation(),DeleteWeatherStation()* methods. It also implements an *UpdateWeatherStationView()* method, which updates the respective View class.
- *DataFileController:* the class that handles the opening, closing, deletion and parsing functionalities. In addition, *DataFileController* implements a *GetCalculations()* method, which parses the *DataFile* and calculates the respective metrics.
- *CalculatorController:* when called by the *GetCalculations()* method of the *DataFile,* the class parses the *DataFile* and, based on the *Formulas* defined in the *Calculator* model class, derives the metrics requested.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DC_1	File parsing
REQ_DC_2	Non-existing meter
REQ_DC_3	Out of limit values
REQ_DC_4	Invalid numbers
REQ_DC_5	Invalid numbers format
REQ_DC_6	Invalid time format
REQ_DC_7	Invalid sampling period
REQ_DC_8	Missing values - gaps
REQ_DC_9	Validation information
REQ_DC_10	Time series
REQ_DC_11	Power consumption
REQ_DC_12	Energy consumption
REQ_DC_13	Data values for not existent meters
REQ_DC_14	Events
REQ_DC_15	Meter mapping
REQ_DC_16	Virtual meters
REQ_DC_17	Building attributes
REQ_DC_18	Building devices
REQ_DC_19	Value limit
REQ_DC_21	Insert incoming data
REQ_DC_22	Reject incoming data
REQ_DC_23	Store validation information
REQ_DC_24	Save static data
REQ_DC_25	Duplicates

2.2.2 Package DA.Controllers

This package contains the controllers of the models defined in Section 2.1.2. The Data Analytics Controllers define all the post processing and validation functionality expected on the *DA.Models*. The *DC.Controllers* are depicted in Figure 10. As shown, each *DC.Controller* class comprises at least one *DC.Model* and *DC.View* object, as well as a number of methods that implement the expected functionality.

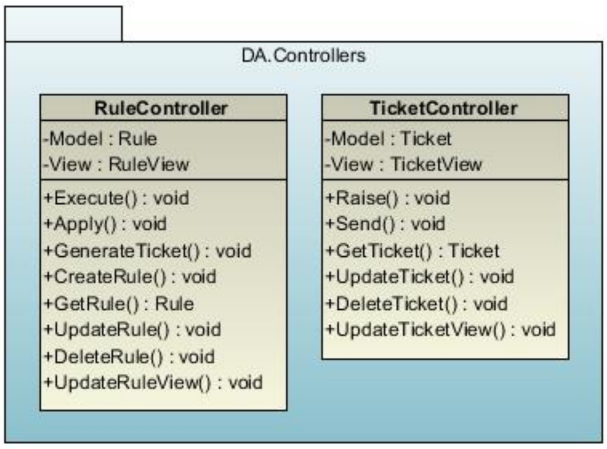


Figure 10:Data Analytics controllers

Class description

- *RuleController:* the class that handles the CRUD (C:Create, R: Read, U: Update, D: Delete) functionality of *Rule*, through the *CreateRule(), GetRule(), UpdateRule(), DeleteRule()* methods. Additionally, respective methods for applying and Executing a *Rule,* as well as for generating a *Ticket* are specified. *RuleController also* implements an *UpdateRuleView()* method, which updates the respective View class.
- *TicketController:* the class that handles the CRUD functionality of *Ticket*, through the *Raise(), GetTicket(), UpdateTicket(), DeleteTicket()* methods. Additionally, *TicketController* implements an *UpdateTicketView()* method, which updates the respective View class. Finally, a *Ticket* can be sent via the *Send()* method.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DA_1	Define metrics
REQ_DA_2	Calculate metrics
REQ_DA_3	Create rules
REQ_DA_4	Browse rules
REQ_DA_5	Apply rules
REQ_DA_6	Edit rules
REQ_DA_7	Delete rules

2.2.3 Package Orchestration.Controllers

This package contains the controllers of the models defined in Section 2.1.3. The Orchestration Controller defines all the service composition functionality expected on the *Orchestration.Models*. The *Orchestration.Controller* is depicted in Figure 11. As shown, the *Orchestration.Controller* class comprises one *Composition* Model and one *CompositionView* object, as well as a number of methods that implement the expected functionality.

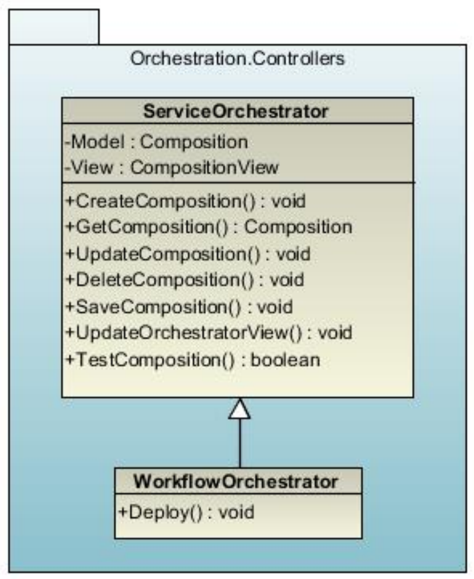

Orchestration.Controllers
ServiceOrchestrator
-Model : Composition
-View : CompositionView
+CreateComposition() : void
+GetComposition() : Composition
+UpdateComposition() : void
+DeleteComposition() : void
+SaveComposition() : void
+UpdateOrchestratorView() : void
+TestComposition() : boolean
WorkflowOrchestrator
+Deploy() : void

Figure 11: Orchestration controllers

Class description

- *ServiceOrchestrator:* the class that handles the CRUD (C:Create, R: Read, U: Update, D: Delete) functionality of *Composition*, through the *CreateComposition(), GetComposition(), UpdateComposition(), DeleteComposition()* methods. Additionally, a method for saving *Composition* details is specified. Finally, an *UpdateOrchestratorView()* method is mplemented, which updates the respective View class.
- *WorkflowOrchestrator:* this class extends the *ServiceOrchestrator* class and implements the *deploy()* method, in case the *Composition* is executed on the WTC cloud.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DA_8	Service composition
REQ_DA_9	Browse services
REQ_DA_10	Test dataset
REQ_DA_11	Composition workflow

2.2.4 Package DVUI.Controllers

This package contains the controllers of the models defined in Section 2.1.4. The Data visualization and User interaction Controllers define the expected functionality to be exposed from the *DVUI.Models*. The *DVUI.Controllers* are depicted in Figure 12. As shown, the *DVUI.Controllers* class comprises at least one *DVUI.Model* and one *DVUI.View* object, as well as a number of methods that implement the expected functionality.

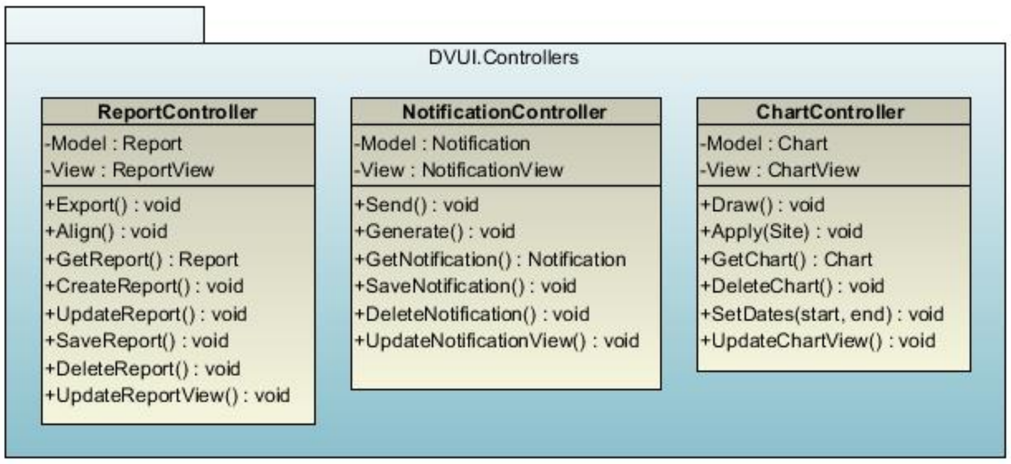


Figure 12: Data Visualization and User Interaction controllers

Class description

- *ReportController:* the class that handles the CRUD functionality of *Report*, through the *CreateReport(), GetReport(), UpdateReport(), DeleteReport()* methods. Additionally, respective methods for applying and exporting and saving a *Report,* as well as for aligning a *Report* are specified. *ReportController also* implements an *UpdateReportView()* method, which updates the respective View class.
- *NotificationController:* the class that handles the CRUD functionality of *Notification*, through the *Generate(), GetNotification(), UpdateNotification(), DeleteNotification()* methods. Additionally, *NotificationController* implements an *UpdateNotificationView()* method, which updates the respective View class. Finally, a *Notification* can be sent via the *Send()* method.
- *ChartController:* the class that handles the CRUD functionality of *Chart*, through the *Draw(), GetChart(), DeleteChart()* methods. Additionally, respective methods for applying a *Chart,* as well as for setting specific dates for the *Chart* are specified. Finally, *ChartController also* implements an *UpdateChartView()* method, which updates the respective View class.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DVUI_1	Generate ticket
REQ_DVUI_2	Send e-mail
REQ_DVUI_3	Generate report
REQ_DVUI_4	sms
REQ_DVUI_5	Create chart
REQ_DVUI_6	Delete chart
REQ_DVUI_7	Edit chart

REQ_DVUI_8	Save chart
REQ_DVUI_9	Browse charts library
REQ_DVUI_10	Compare charts
REQ_DVUI_11	Chart granularity
REQ_DVUI_12	Apply chart
REQ_DVUI_13	Export file

2.3 Views

This layer contains all the views of the WTC and is the User interface layer of the system. It contains all windows that the user may view and interact with the models of WTC through the respective controllers. Following the aforementioned architecture, views are grouped into four packages: Data Collection (DC), Data Analytics (DA), Orchestration and Data Visualization and User Interaction (DVUI) Views.

2.3.1 Package DC.Views

This package contains all the views for the models defined in Section 2.1.1. The Data Collection views provide access to the *DC.Models*. The *DC.Views* are depicted in Figure 13.

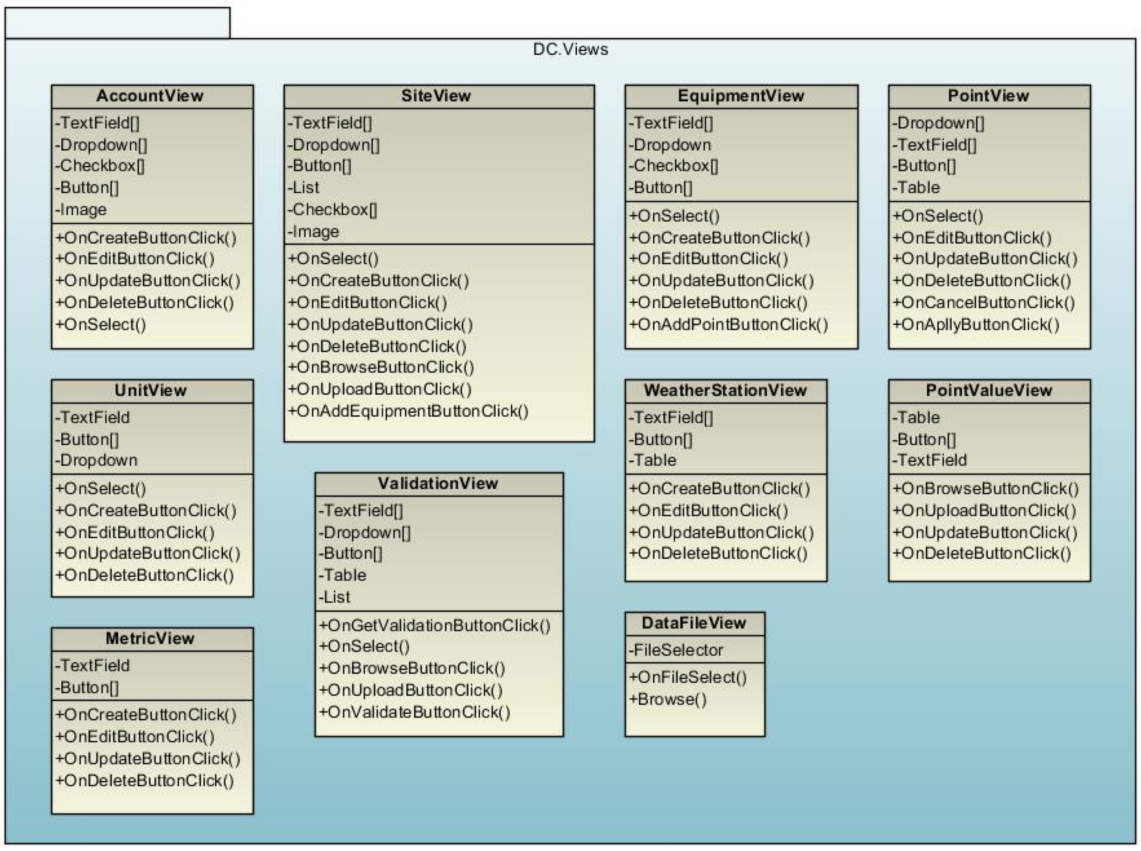


Figure 13: Data Collection views

Class description

- *AccountView:* the class that allows handling *Account* models, based on the functionality defined in the *AccountController.*
- *SiteView:* it allows handling *Site* models, based on the functionality defined in the *SiteController.* It also allows adding *Equipment* instances to a given *Site.*
- *EquipmentView:* the class that allows handling *Equipment* models, based on the functionality defined in the *EquipmentController.* It also allows adding *Point* instances to a given *Equipment.*
- *PointView:* it allows handing objects of *Point* type, based on the functionality defined in the *PointController*.
- *PointValueView:* the class that allows interaction with *PointValue* objects, i.e. the values measured for a specific *Point*.
- *UnitView:* it allows defining the *Unit* of the *PointValues* gathered by a specific *Point.* The view accommodates all the functionality defined in the *UnitController.*
- *ValidationView:* the class that allows the user to perform validation actions on the *PointValue* data collected. All *ValidationController* data-related methods are executed, through the *OnGetValidationButtonClick()* method.
- *WeatherStationView:* this pane allows adding and editing *WeatherStation* related data.
- *MetricView:* it allows handling objects of type *Metric*, which are calculated on the *PointValue* instances.
- *DataFileView:* the class that opens a dialog editor, to select a *DataFile* to process.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DC_1	File parsing
REQ_DC_2	Non-existing meter
REQ_DC_3	Out of limit values
REQ_DC_4	Invalid numbers
REQ_DC_5	Invalid numbers format
REQ_DC_6	Invalid time format
REQ_DC_7	Invalid sampling period
REQ_DC_8	Missing values - gaps
REQ_DC_9	Validation information
REQ_DC_10	Time series
REQ_DC_11	Power consumption
REQ_DC_12	Energy consumption
REQ_DC_13	Data values for not existent meters
REQ_DC_14	Events

REQ_DC_15	Meter mapping
REQ_DC_16	Virtual meters
REQ_DC_17	Building attributes
REQ_DC_18	Building devices
REQ_DC_19	Value limit
REQ_DC_21	Insert incoming data
REQ_DC_22	Reject incoming data
REQ_DC_23	Store validation information
REQ_DC_24	Save static data
REQ_DC_25	Duplicates

2.3.2 Package DA.Views

This package contains all the views for the models defined in Section 2.1.2. The Data Analytics views provide access to the *DA.Models*. The *DA.Views* are depicted in Figure 14.

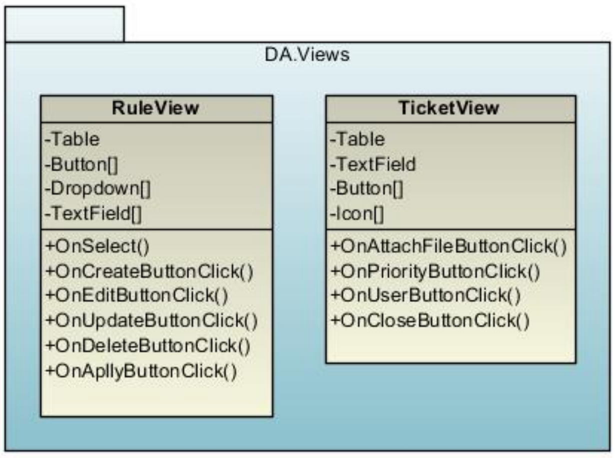


Figure 14: Data Analytics views

Class description

- *RuleView:* the class that allows handling *Rule* models, based on the functionality defined in the *RuleController.*
- *TicketView:* the class that allows handling *Ticket* models, based on the functionality defined in the *TicketController.*

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DA_1	Define metrics
REQ_DA_2	Calculate metrics
REQ_DA_3	Create rules
REQ_DA_4	Browse rules
REQ_DA_5	Apply rules
REQ_DA_6	Edit rules
REQ_DA_7	Delete rules

2.3.3 Package Orchestration.Views

This package contains all the views for the models defined in Section 2.1.3. The Orchestration views provide access to the *Orchestration.Models* through the respective controllers' methods. The *Orchestration.Views* are depicted in Figure 15.

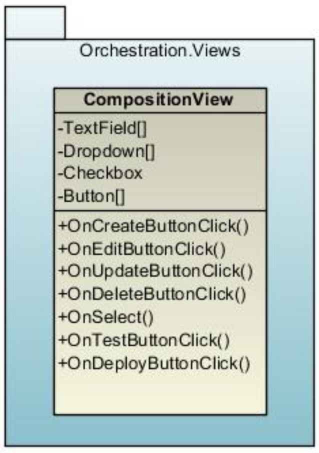


Figure 15: Orchestration views

Class description

- *CompositionView:* the class that will visualize the *Composition* process, as defined in the *ServiceOrchestrator.* Additionally, it will allow the deployment of the *Composition,* through the *OnDeployButtonClick()* method.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DA_8	Service composition
REQ_DA_9	Browse services
REQ_DA_10	Test dataset
REQ_DA_11	Composition workflow

2.3.4 Package DVUI.Views

This package contains all the views for the models defined in Section 2.1.4. The Data Visualization and User Interaction views provide access to the *Orchestration.Models* through the respective controllers' methods. The *DVUI.Views* are depicted in Figure 16.

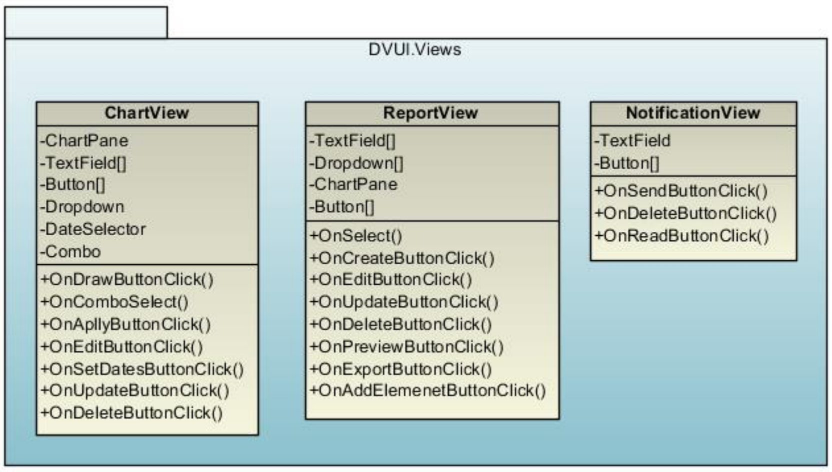


Figure 16: Data Visualization and User Interaction views

Class description

- *ChartView:* this View allows creating and editing object of *Chart* type, as defined in the *ChartController.* The *OnApplyButtonClick()* method redirects to the respective *ChartController* method, in order to apply a chart to a user selected *Point.*
- *ReportView:* the class that handles *Report* objects, as defined in the *ReportController.* The *OnAddElementButtonClick()* method allows to dynamically add a new reporting element to the report that is generated for the user.
- *NotificationView:* it allows the user to handle *Notification* objects.

Design

The DELPHIS engineers will design the classes in the form of RAML specifications.

Implementers

These classes will be generated by the DELPHIS developers.

Requirements

Identifier	Title
REQ_DVUI_1	Generate ticket
REQ_DVUI_2	Send e-mail
REQ_DVUI_3	Generate report
REQ_DVUI_4	sms
REQ_DVUI_5	Create chart
REQ_DVUI_6	Delete chart
REQ_DVUI_7	Edit chart
REQ_DVUI_8	Save chart
REQ_DVUI_9	Browse charts library
REQ_DVUI_10	Compare charts
REQ_DVUI_11	Chart granularity
REQ_DVUI_12	Apply chart
REQ_DVUI_13	Export file

2.4 Module interaction

In the WTC architecture, the interaction between the system's module follows the MVC's interaction logic. This means that each of the model-view-controller elements can talk to each other through well-defined, loosely coupled interfaces, as shown in Figure 17.

Models – Views interaction: A model and a view can talk to each other through status query interfaces, provided by the model, and change notifications, provided by the view.

Models – Controllers interaction: A model and a controller can talk to each other through state change requests, provided by the model, and notifications sent by the controller to the view.

Views – Controllers interaction: A view and a controller can talk to each other through user action events sent by the controller to the view and view update interfaces provided by the view.

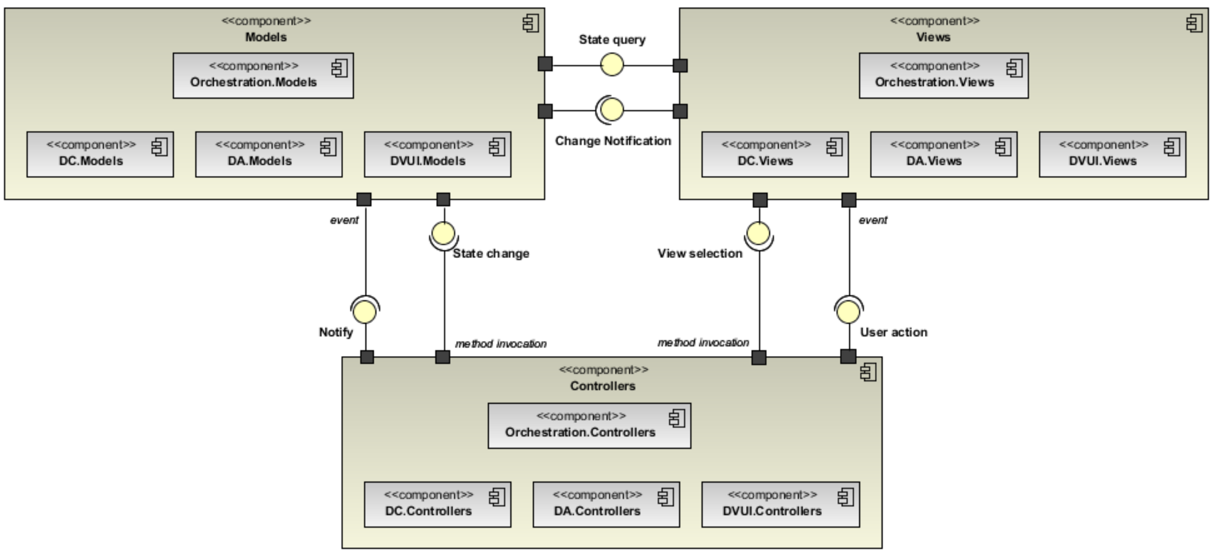
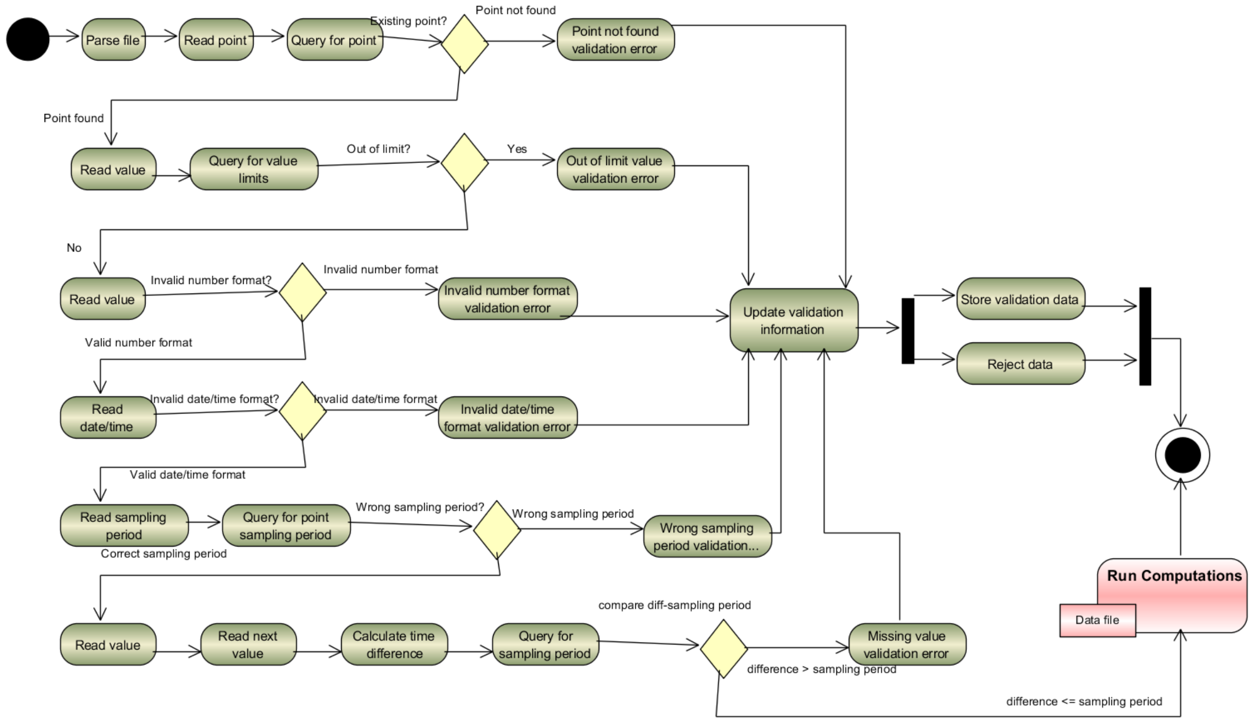


Figure 17:Module interaction in WTC MVC architecture

2.4.1 Activity diagrams / storyboards

Figure 18 shows the sequence of actions for data validation, when new building data is available in the form of data file coming from a BMS. All validation processes run sequentially, one after the other. In case that any of them fails, the system inserts the corresponding validation information into the database. If all validation processes succeed, the process continues to with the execution of the computation algorithms.



Parse file
Read point
Query for point
Existing point?
Point not found
Point not found validation error
Point found
Read value
Query for value limits
Out of limit?
Yes
Out of limit value validation error
No
Read value
Invalid number format?
Invalid number format
Invalid number format validation error
Update validation information
Store validation data
Reject data
Valid number format
Read date/time
Invalid date/time format?
Invalid date/time format
Invalid date/time format validation error
Valid date/time format
Read sampling period
Query for point sampling period
Wrong sampling period?
Wrong sampling period
Wrong sampling period validation...
Correct sampling period
Run Computations
Data file
compare diff-sampling period
Read value
Read next value
Calculate time difference
Query for sampling period
Missing value validation error
difference > sampling period
difference <= sampling period

Figure 18: Data validation activity diagram

Figure 19 shows the sequence of actions for running all the requested computation algorithms, after data validation has completed successfully. The system identifies the requested computations after parsing the incoming data file and runs them sequentially. When all calculations have finished, all data entries are stored in the system's database.

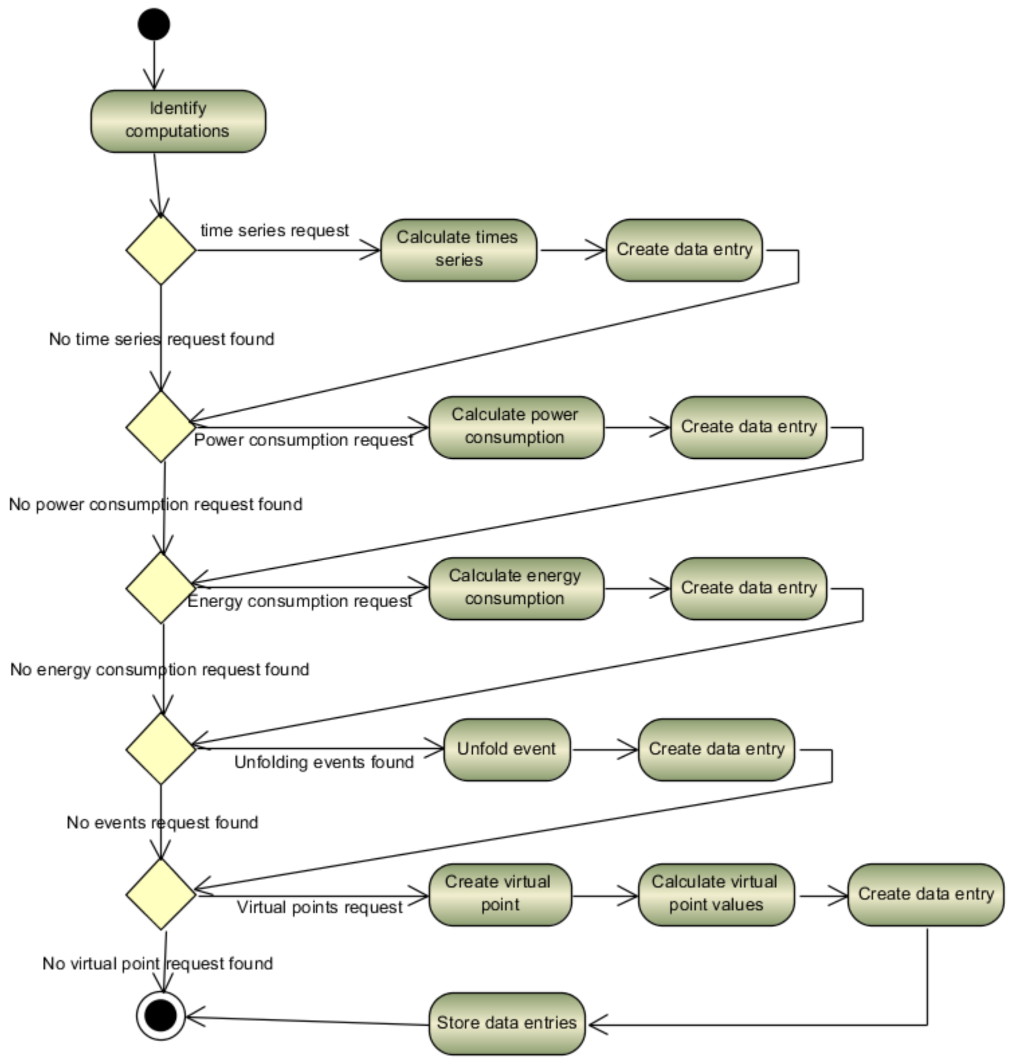


Figure 19: Calculate computations activity diagram

Figure 20 shows the sequence of actions for creating a new chart and generate its corresponding view. When the user requests a new chart creation, the system loads the configuration environment giving the user the option to parameterize all available chart elements. When the user has finished configuration, the system generates a chart preview and the user can save the newly created chart, with or without generating it (apply chart on a specific site).

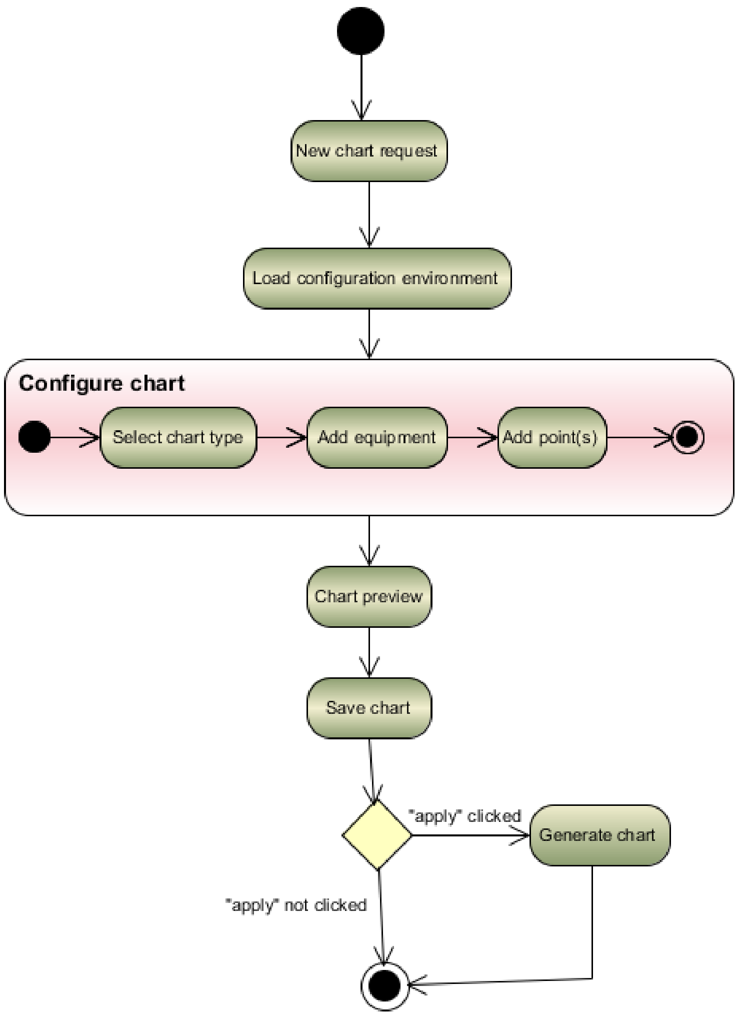


Figure 20: Create / apply chart activity diagram

Figure 21 shows the sequence of actions for creating a new service composition. When the user requests a new service composition creation the system loads the available services which can be part(s) of a composition. The user can select a number of services, define their sequence of execution and test the composition (services & sequence). If the system does not find any errors, then it generates a URI for the new service composition and the user can save it for future invocation.

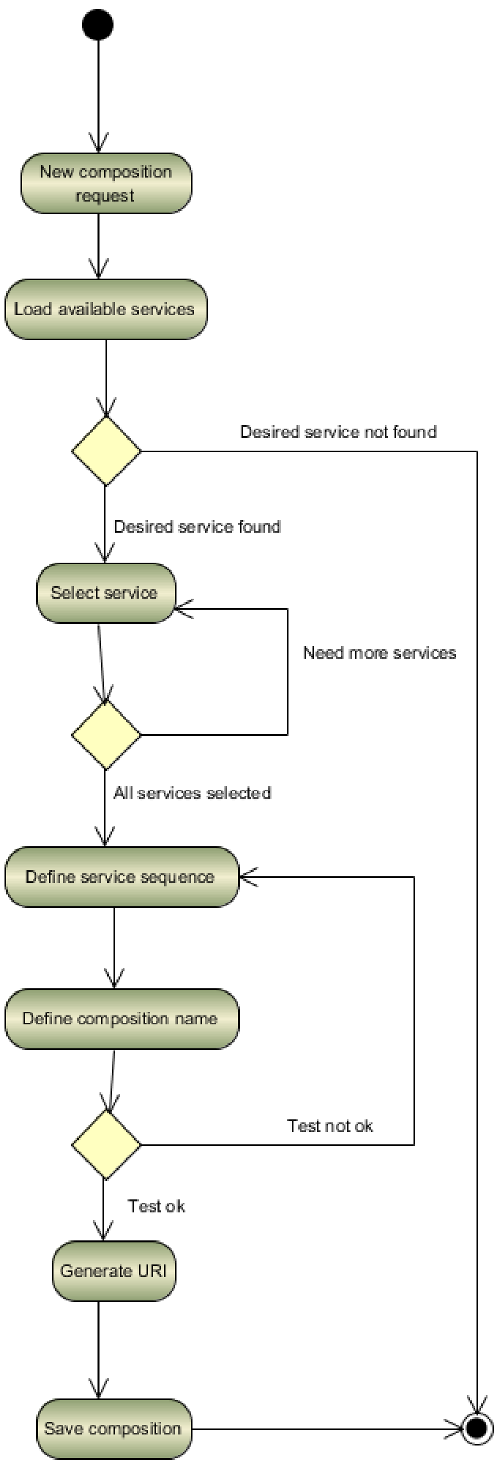


Figure 21: Create service composition activity diagram

3 WTC pilot services

Section 3 gives a detailed description of the basic services that are going to be deployed and integrated within the context of the WTC pilot. In this section, a mapping between these services and the package they belong to is given for a better understanding of the architecture.

3.1 List of services to be deployed using the S-CASE infrastructure

Table 1: Services to be developed by the S-CASE engine

Category	Web service	Description	Requirements reference
Data collection	Store data	A service executed in order to save building data into a persistent storage	REQ_DC_15 REQ_DC_21 REQ_DC_22 REQ_DC_23 REQ_DC_24 REQ_DC_25
	Create building	A service executed in order to create the logical entity of a building	REQ_DC_17 REQ_DC_18 REQ_DC_19
	Edit building	A service executed in order to edit the attributes of a logical building	REQ_DC_17 REQ_DC_18 REQ_DC_19
Data analytics	Produce metrics	A service executed in order to produce metrics from raw data	REQ_DA_1 REQ_DA_2
	Create rule	A service executed in order to create a rule	REQ_DA_3
	Edit rule	A service executed in order to edit an existing rule	REQ_DA_4 REQ_DA_6
	Delete rule	A service executed in order to delete a rule	REQ_DA_4 REQ_DA_7
User tools	Create chart	A service executed in order to create a chart	REQ_DV-UI_5 REQ_DV-UI_8
	Edit chart	A service executed in order to edit an existing chart	REQ_DV-UI_7 REQ_DV-UI_8 REQ_DV-UI_9
	Delete chart	A service executed in order to delete a chart	REQ_DV-UI_6 REQ_DV-UI_9
	Create service composition	A service executed in order to create a service composition by combining existing services	REQ_DA_8 REQ_DA_9 REQ_DA_10 REQ_DA_11
	Test service composition	A service executed in order to test a service composition over an existing dataset	REQ_DA_8 REQ_DA_9 REQ_DA_10 REQ_DA_11
	Build report	A service executed in order to create a report	REQ_DV-UI_3 REQ_DV-UI_13

3.2 List of services used to validate the S-CASE infrastructure

Table 2: Services to be developed by the WTC team

Category	Web service	Description	Requirements reference
Data collection	Validate data	A service executed in order to validate building data	REQ_DC_1 REQ_DC_2 REQ_DC_3 REQ_DC_4 REQ_DC_5 REQ_DC_6 REQ_DC_7 REQ_DC_8 REQ_DC_9 REQ_DC_15 REQ_DC_19
	Calculate computable values	A service executed in order to calculate computable data values of a building	REQ_DC_10 REQ_DC_11 REQ_DC_12 REQ_DC_14
	Virtual meters	A service executed in order to create meter entities that do not physically exist	REQ_DC_13 REQ_DC_16
Data analytics	Apply rule	A service executed in order to apply a rule on a logical equipment/device	REQ_DA_4 REQ_DA_5
	Generate ticket	A service executed in order to generate a fault ticket	REQ_DV-UI_1 REQ_DV-UI_2 REQ_DV-UI_4
User tools	Apply chart	A service executed in order to apply a chart on a dataset	REQ_DV-UI_11 REQ_DV-UI_12
	Compare charts	A service executed in order to compare two or more charts	REQ_DV-UI_9 REQ_DV-UI_10 REQ_DV-UI_11

3.3 List of services to be integrated (3rd party)

Table 3: List of 3rd party services to be integrated

3rd party service needed	Requirement reference	Description
Weather data integration	REQ_DC_20	As a system I shall be able to accept incoming weather data from different weather stations
PDF creator Image creator	REQ_DV-UI_13	As a system I shall be able to export files in pre-specified formats (images, documents)
SMS provider	REQ_DV-UI_4	As a system I shall be able to send sms to users
e-mail provider	REQ_DV-UI_2	As a system I shall be able to send e-mails
Currency converter	REQ_DV-UI_1	As a system I shall be able to monetize the cost of a ticket
File parser	REQ_DC_1	As a system I shall be able to parse and process incoming files that contain building data
Chart generator	REQ_DV-UI_5 REQ_DV-UI_7 REQ_DV-UI_10 REQ_DV-UI_12	As a user I shall be able to create a new chart As a user I shall be able to edit an existing chart As a user I shall be able to compare two or more charts by adding them in the same pane As a user I shall be able to apply a chart on a specific equipment
Authentication	NFR1	As a system I shall ensure that data is protected from unauthorized access

3.4 RAML specifications

For the WTC REST API design a set of RAML specification files have been created. RAML was chosen due to its very simple, practical yet complete language for describing REST APIs. RAML encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. RAML is built on broadly-used standards such as YAML and JSON and is a non-proprietary, vendor-neutral open spec.[6]

While one can write the RAML specifications in any text editor, we decided to use Mulesoft's web-based API Designer, which provides an embedded RAML API console with suggestions, error feedback and testing environment.

[6] http://www.mulesoft.org/documentation/display/current/Walkthrough+Design+New

3.4.1 Accounts

```
#%RAML 0.8
title: WTC-accountsAPI
baseUri: https://app.watchtowercloud.com/api
version: 0.1
/accounts:
  displayName: Accounts
  get:
    description: Get a list of the existing WTC accounts
  /{accountId}:
    displayName: Account
    get:
      description: Get the information associated with a specific account
      responses:
        200:
          body:
            application/json:
              example: |
                {
                  "accountId": "01",
                  "accountType": "Bank",
                  "accountName" : "NBG",
                  "description": "National Bank of Greece",
                  "address" : "Egnatias 12",
                  "postalCode" : "54364,
                  "city" : "Thessaloniki",
                  "country" : "Greece",
                  "sites" : [],
                  "numOfSites" : "1"
                }
    delete:
      description: Delete a specific account
    post:
      description: Create a new account
      queryParameters:
        name:
          displayName: AccountName
          type: string
          required: true
        username:
          displayName: Username
          type: string
          required: true
          minLength: 6
        password:
          displayName: Password
          type: string
          required: true
          minLength: 6
          pattern: ^[a-zA-Z]{2}\-[0-9a-zA-Z]{3}\-\d{2}\-\d{5}$
        address:
          displayName: Address
          type: string
          required: false
      responses:
        200:
          body:
            application/json:
              example: |
                {"accountId": "1234"}
        500:
          body:
            application/json:
              example: |
                {
                  "errorMessage": "The account couldn't be entered."
                }
    put:
      description: Update an existing account
```

3.4.2 Sites

```
#%RAML 0.8
title: WTC-sitesAPI
baseUri: https://app.watchtowercloud.com/api
version: 0.1
/sites:
  displayName: Sites
  get:
    description: Get a list of the existing sites
  /{siteId}:
    displayName: Site
    get:
      description: Get the information associated with a specific site
      responses:
        200:
          body:
            application/json:
    post:
      description: Create a new site
    put:
      description: Update an existing site
    delete:
      description: Delete an existing site
    /equipment:
      displayName: SiteEquipments
      get:
        description: Get the list of existing equipment associated with a site
        responses:
          200:
            body:
              application/json:
      /{equipmentId}:
        displayName: SiteEquipment
        get:
          description: Get the information associated with a specific equipment
          responses:
            200:
              body:
                application/json:
        post:
          description: Create a new equipment associated with a site
        put:
          description: Update a site's equipment
        delete:
          description: Delete a site's equipment
        /points:
          displayName: EquipmentPoints
          get:
            description: Get the list of equipment's points
          /{pointId}:
            displayName: EquipmentPoint
            get:
              description: Get the information associated with a specific point
            post:
              description: Create a new point for equipment
            put:
              description: Update an existing equipment's point
            delete:
              description: Delete an existing equipment's point
    /rules:
      displayName: SiteRules
      get:
        description: Get the list of applied FDD rules on a site
        responses:
          200:
            body:
              application/json:
      /{ruleId}:
        displayName: SiteRule
        get:
          description: Get the information associated with a specific rule
          responses:
            200:
              body:
                application/json:
                  example: |
                    {
                      "Name" : "TestRule",
                      "Description" : "Rule description",
                      "Author" : "John John",
                      "Trigger" : "If something",
                      "Effect" : "Then something else",
                      "IsApplied" : "true"
                    }
        post:
          description: Create a new rule for a site
          queryParameters:
            name:
            description:
            author:
            trigger:
            effect:
            isApplied:
        put:
          description: Update an existing rule for a site
        delete:
          description: Delete an existing rule from a site
    /metrics:
      displayName: SiteMetrics
      get:
        description:
        responses:
          200:
            body:
              application/json:
      /{metricId}:
        displayName: SiteMetric
        get:
          description:
          responses:
            200:
              body:
                application/json:
        post:
        put:
        delete:
    /charts:
      displayName: SiteCharts
      get:
        description: Get the list of charts associated with a site
        responses:
          200:
            body:
              application/json:

      /{chartId}:
        displayName: SiteChart
        get:
        post:
        put:
        delete:
```

3.4.3 Validations

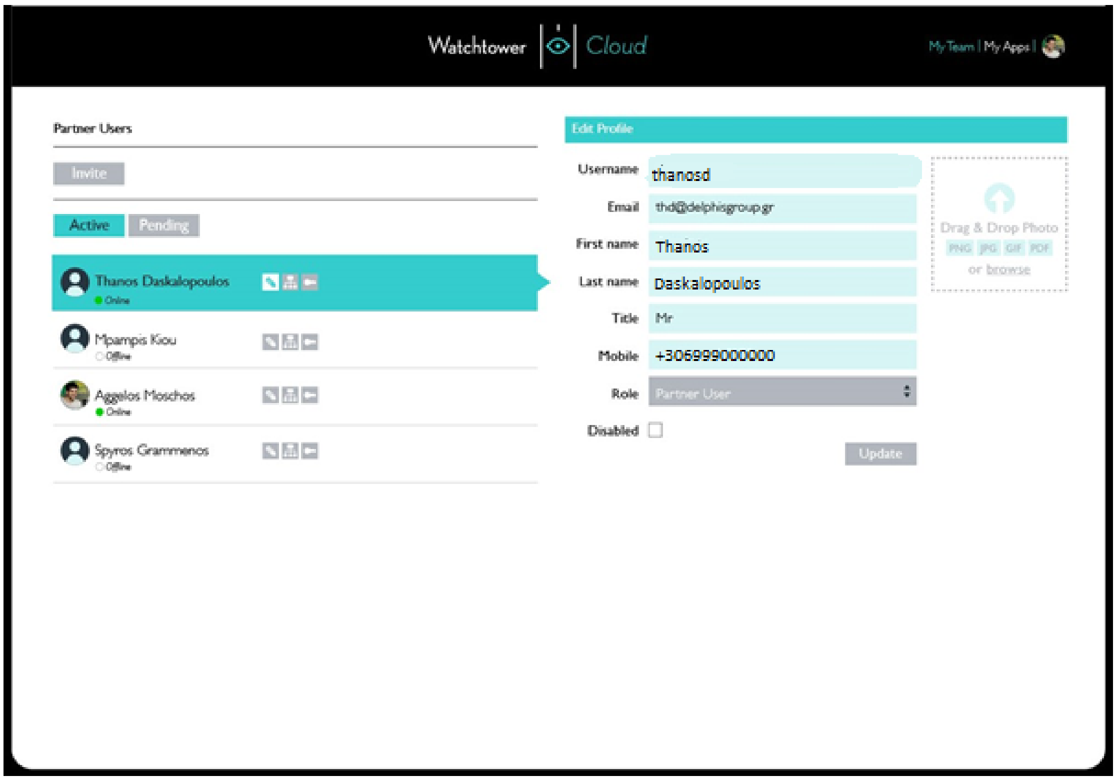
```
#%RAML 0.8
title: WTC-validation
version: 0.1
baseUri: http://app.watchtowercloud.com/api
resourceTypes:
  dataFile:
    description: A data file containing measurements
    get:
      responses:
        200:
          body:
            application/json:
              schema: |
              { "type" : "object",
                "$schema" : "http://json-schema.org/draft-01/schema",
                "id":"http://jsonschema.net",
                "required" : true,
                "values" : {
                  "pointId" : "string",
                  "timestamp" : "string",
                  "value" : "float"
                },
                "events" : {
                  "pointId" : "string",
                  "timestamp" : "string",
                  "value" : "float"
                },
                "repetitiveValues" : "array",
                "repetitiveEvents" : "array",
                "properties" : {
                  "siteId" : "int",
                  "pointId" : "string",
                  "params" : {
                    "cum_to_quarter" : {
                      "pointId": "string",
                      "virtualPointId" : "string"
                    },
                    "applyFactor" : {
                      "pointId" : "string",
                      "factor" : "int"
                    },
                    "computables" : {
                      "virtualPointId" : "string",
                      "formula" : {
                        "plus" : [],
                        "minus" : []
                      }
                    }
                  }
                }
              }
/validations:
  displayName: Validations
```

```
get:
 description: Get a list of all the available validation algorithms
/validationId:
  displayName: Validation
  get:
    description: Run a validation
    queryParameters:
      name:
        displayName: ValidationName
        type: string
        required: true
      data:
        displayName: ValidationData
        type: dataFile
        required: true
    responses:
      200:
        body:
          application/json:
```

4 UI design

In this section a number of mock-up screens are given, which represent some of the main views of the WTC.

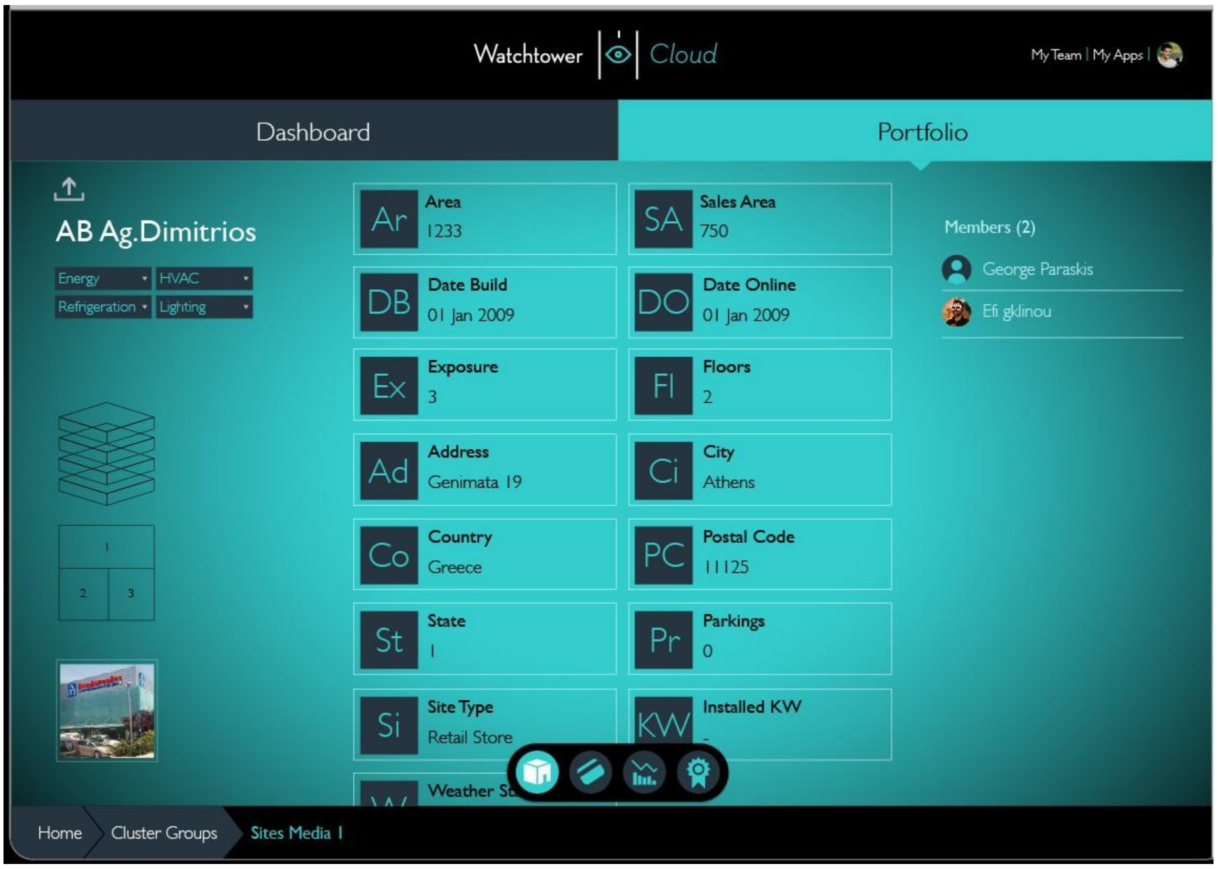
Figure 22 shows the account view when a specific account is selected. All accounts are listed in the left-hand side, while account-specific information is given in the right-hand side of the screen.



Username	thanosd
Email	thd@delphisgroup.gr
First name	Thanos
Last name	Daskalopoulos
Title	Mr
Mobile	+306999000000
Role	Partner User
Disabled	☐

Figure 22: Account view – Selected account info

Figure 23 shows the view of a selected site. Information regarding site's characteristics and attributes (area covered, location, installed KW, etc.) is given in a user friendly, tabular format.



Watchtower | Cloud

My Team | My Apps |

Dashboard | Portfolio

AB Ag.Dimitrios

Energy · HVAC · Refrigeration · Lighting

Area: 1233	Sales Area: 750
Date Build: 01 Jan 2009	Date Online: 01 Jan 2009
Exposure: 3	Floors: 2
Address: Genimata 19	City: Athens
Country: Greece	Postal Code: 11125
State: 1	Parkings: 0
Site Type: Retail Store	Installed KW: -
Weather St	

Members (2)

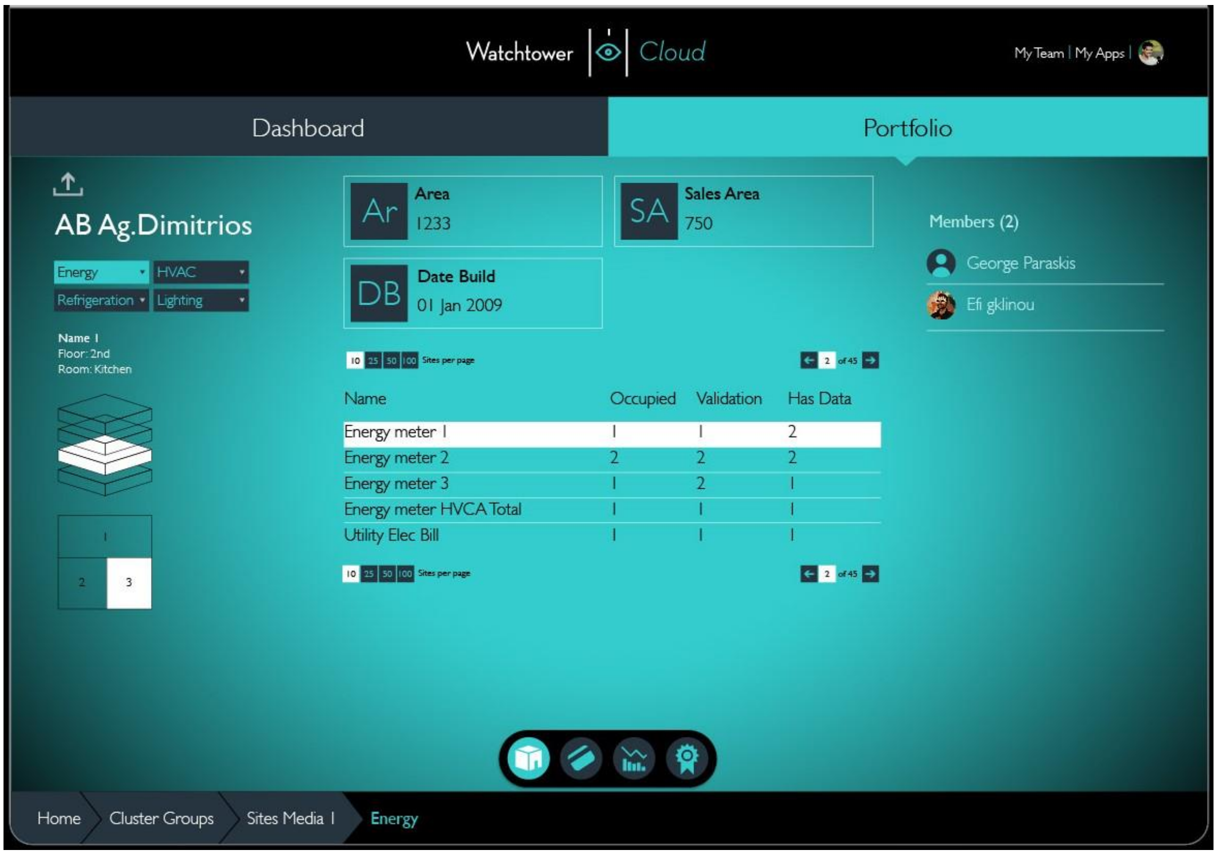
George Paraskis

Efi gklinou

Home > Cluster Groups > Sites Media I

Figure 23: Site view – selected site info

Figure 24 shows the view of a selected site's equipment. Information regarding equipment's characteristics and attributes is given in a user friendly, tabular format.



Ar Area 1233

SA Sales Area 750

DB Date Build 01 Jan 2009

Name 1
Floor: 2nd
Room: Kitchen

Name	Occupied	Validation	Has Data
Energy meter 1	1	1	2
Energy meter 2	2	2	2
Energy meter 3	1	2	1
Energy meter HVCA Total	1	1	1
Utility Elec Bill	1	1	1

Figure 24: Equipment view – selected equipment info

Figure 25 shows the view of the ticketing notification system. Tickets can be categorized into unread, pending and closed.

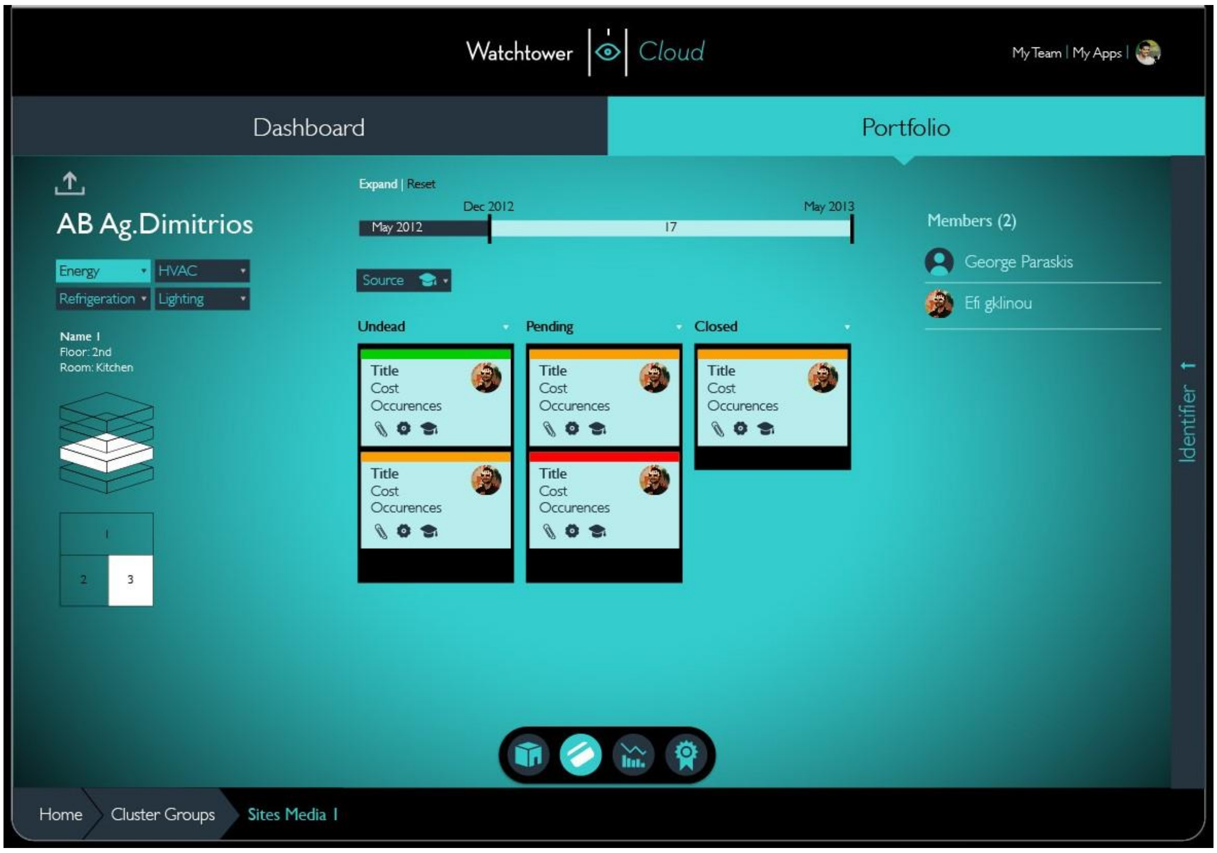
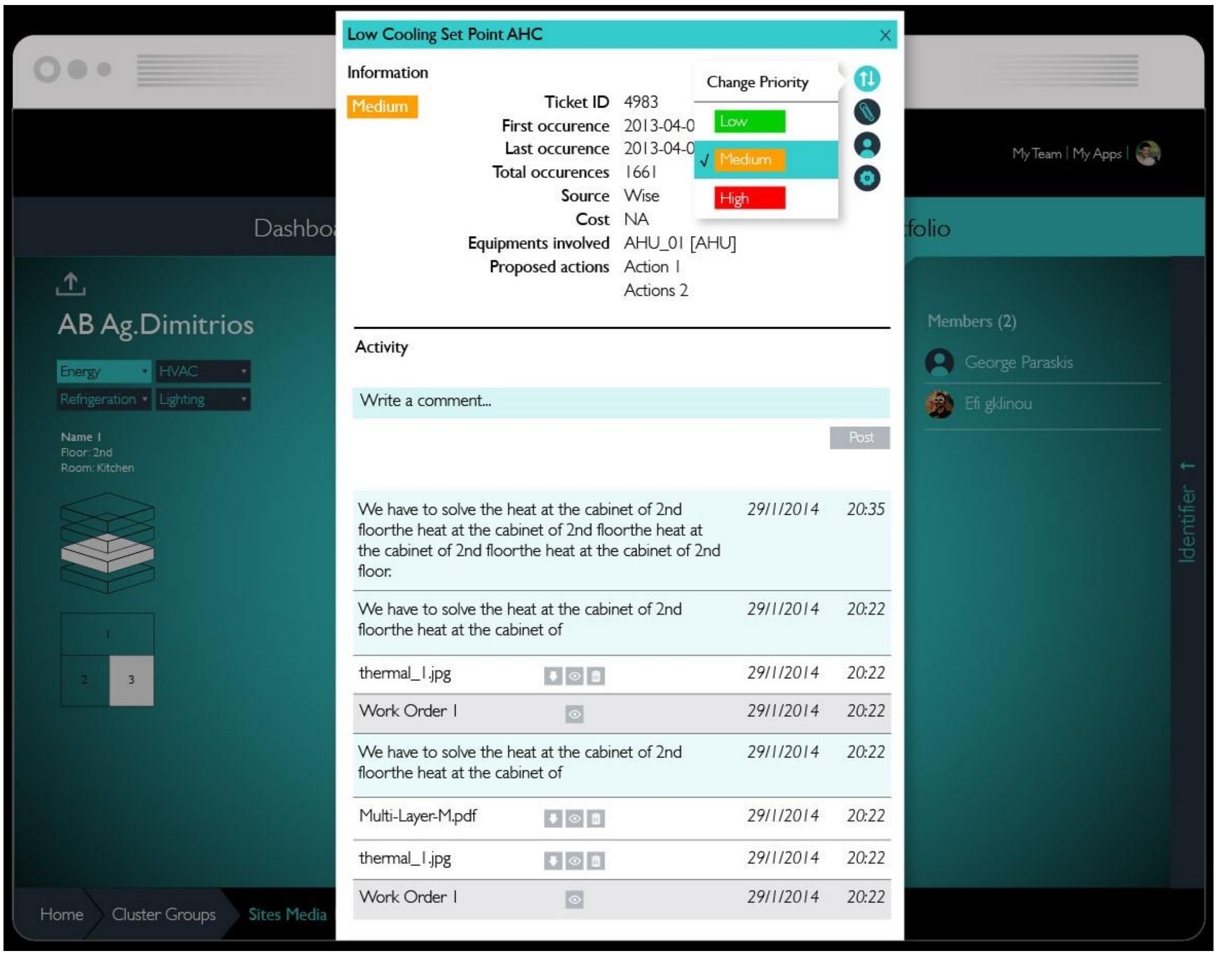


Figure 25: Ticket view – Unread, pending and closed tickets

Figure 26 shows the view of a selected ticket which has been opened. Each ticket is fully described as explained in section 2.1.4 and is given a priority. Users can also add comments for better communication and reference.



Low Cooling Set Point AHC

Information

Medium

Ticket ID	4983
First occurence	2013-04-0
Last occurence	2013-04-0
Total occurences	1661
Source	Wise
Cost	NA
Equipments involved	AHU_01 [AHU]
Proposed actions	Action 1 Actions 2

Change Priority
Low
[✓ Medium]
High

Activity

Write a comment...

Post

We have to solve the heat at the cabinet of 2nd floorthe heat at the cabinet of 2nd floorthe heat at the cabinet of 2nd floorthe heat at the cabinet of 2nd floor.	29/11/2014	20:35
We have to solve the heat at the cabinet of 2nd floorthe heat at the cabinet of	29/11/2014	20:22
thermal_1.jpg	29/11/2014	20:22
Work Order 1	29/11/2014	20:22
We have to solve the heat at the cabinet of 2nd floorthe heat at the cabinet of	29/11/2014	20:22
Multi-Layer-M.pdf	29/11/2014	20:22
thermal_1.jpg	29/11/2014	20:22
Work Order 1	29/11/2014	20:22

Figure 26: Ticket view – selected ticket info

Figure 26 shows the view of a chart pane which has been generated for a specific point. A chart pane can contain several charts. A selector is also provided giving the user the ability to change which point or metric to be drawn.

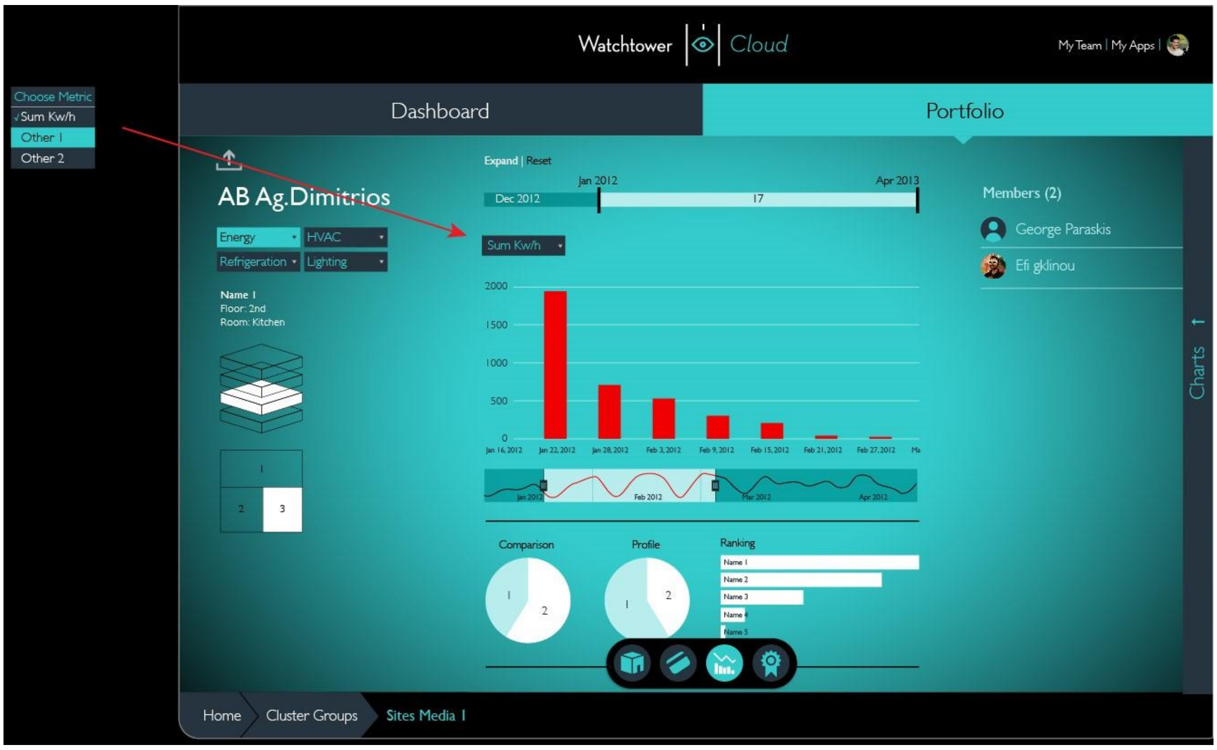


Figure 27: Chart view

5 Conclusion

In this deliverable, we presented the architectural models and specifications for the WTC pilot of DELPHIS. Starting from an overview of the envisioned system, and following the MVC pattern, four (4) distinct layers have been identified each one consisting of the corresponding Model – View – Controller classes. All architectural modules have been categorized into respective packages and the interaction between them has been defined. The list of services to be deployed within the context of the WTC pilot has been documented and grouped into the three (3) concrete categories specified also in D1.3:

- services to be deployed using the S-CASE infrastructure
- services used to validate the S-CASE infrastructure
- 3rd party services

For the core entities (models), the respective RAML specifications have been also extracted and documented using the mulesoft API designer[7]. Finally, a number of mock-up screens is given for better understanding of the UI to be designed and developed.

This document will serve as the basis for the development and implementation of the WTC pilot during the project period M19-M32.

[7] https://www.mulesoft.com/platform/api/api-designer

Annex I: WTC pilot requirements

Annex I gives an overview of all functional and non-functional requirements related to the WTC pilot. The list of requirements is provided for better reference to them.

Functional requirements

ID	REQ_DC_1: File parsing
Formalized	As a system I shall be able to parse and process incoming files that contain building data.
Area	Data collection

ID	REQ_DC_2: Non-existing meter
Formalized	As a system I shall be able to validate incoming building data in order to recognize if it refers to a non-existing meter.
Area	Data collection

ID	REQ_DC_3: Out of limit values
Formalized	As a system I shall be able to validate incoming building data in order to ensure that there are no out of limit data values.
Area	Data collection

ID	REQ_DC_4: Invalid numbers
Formalized	As a system I shall be able to validate incoming building data in order to ensure that there are no invalid numbers.
Technical Area	Data collection

ID	REQ_DC_5: Invalid numbers format
Formalized	As a system I shall be able to validate incoming building data in order to ensure that there is no invalid number formatting.
Technical Area	Data collection

ID	REQ_DC_6: Invalid time format
Formalized	As a system I shall be able to recognize invalid time (date, datetime, timestamp, etc) formats from incoming building data.
Technical Area	Data collection

ID	REQ_DC_7: Invalid sampling period
Formalized	As a system I shall be able to validate incoming building data in order to recognize invalid sampling periods.
Technical Area	Data collection

ID	REQ_DC_8: Missing values - gaps
Formalized	As a system I shall be able to validate incoming building data in order to recognize gaps due to missing data values.
Technical Area	Data collection

ID	REQ_DC_9: Validation information
Formalized	As a system I shall be able to produce information regarding the source and reason of an incoming dataset not passing the validation process.
Technical Area	Data collection

ID	REQ_DC_10: Time series
Formalized	As a system I shall be able to extract a concrete time series from cumulative incoming building data.
Technical Area	Data collection

ID	REQ_DC_11: Power consumption
Formalized	As a system I shall be able to calculate the average power consumption in kW, based on kWh data samples collected at pre-specified time intervals.
Technical Area	Data collection

ID	REQ_DC_12: Energy consumption
Formalized	As a system I shall be able to calculate the energy consumption in kWh, based on average power (kW) data samples collected at pre-specified time intervals.
Technical Area	Data collection

ID	REQ_DC_13: Data values for not existent meters
Formalized	As a system I shall be able to produce data values time series for meters that are considered as virtual (nonphysical).
Technical Area	Data collection

ID	REQ_DC_14: Events
Formalized	As a system I shall be able to recognize and handle incoming events.
Technical Area	Data collection

ID	REQ_DC_15: Meter mapping
Formalized	As a system I shall be able to map the device name of the incoming data to the correct device name stored in the database.
Technical Area	Data collection

ID	REQ_DC_16: Virtual meters
Formalized	As a system I shall be able to create virtual meters to map information that has physical meaning but not physical source.
Technical Area	Data collection

ID	REQ_DC_17: Building attributes
Formalized	As a user I shall be able to define building’s static data and attributes.
Technical Area	Data collection

ID	**REQ_DC_18: Building devices**
Formalized	As a user I shall be able to define a building's existing and virtual devices.
Technical Area	Data collection

ID	**REQ_DC_19: Value limit**
Formalized	As a user I shall be able to define a meter's value limits (min/max values).
Technical Area	Data collection

ID	**REQ_DC_20: Weather data**
Formalized	As a system I shall be able to accept incoming weather data from different weather stations.
Technical Area	Data collection

ID	**REQ_DC_21: Insert incoming data**
Formalized	As a system I shall be able to insert incoming data in the database.
Technical Area	Data collection

ID	**REQ_DC_22: Reject incoming data**
Formalized	As a system I shall be able to reject incoming data if they do not pass the validation criteria.
Technical Area	Data collection

ID	**REQ_DC_23: Store validation information**
Formalized	As a system I shall be able to store the validation information produced when an incoming dataset does not pass the validation process.
Technical Area	Data collection

ID	REQ_DC_24: Save static data
Formalized	As a user I shall be able to save the static data I have inserted to the system.
Technical Area	Data collection

ID	REQ_DC_25: Duplicates
Formalized	As a system I shall be able to check for duplicate entries in the database before inserting new values.
Technical Area	Data collection

ID	REQ_DA_1: Define metrics
Formalized	As a user I shall be able to define metrics formulas.
Technical Area	Data analytics

ID	REQ_DA_2: Calculate metrics
Formalized	As a system I shall be able to calculate metrics.
Technical Area	Data analytics

ID	REQ_DA_3: Create rules
Formalized	As a user I shall be able to create new rules.
Technical Area	Data analytics

ID	REQ_DA_4: Browse rules
Formalized	As a user I shall be able to browse existing rules.
Technical Area	Data analytics

ID	REQ_DA_5: Apply rules
Formalized	As a user I shall be able to apply rules on a device/equipment.
Technical Area	Data analytics

ID	REQ_DA_6: Edit rules
Formalized	As a user I shall be able to edit rules.
Technical Area	Data analytics

ID	REQ_DA_7: Delete rules
Formalized	As a user I shall be able to delete rules.
Technical Area	Data analytics

ID	REQ_DA_8: Service composition
Formalized	As a user I shall be able to create a service composition from existing services.
Technical Area	Data analytics

ID	REQ_DA_9: Browse services
Formalized	As a user I shall be able to see the list of available services exposed by the system.
Technical Area	Data analytics

ID	REQ_DA_10: Test dataset
Formalized	As a user I shall be able to use test energy-related datasets.
Technical Area	Data analytics

ID	REQ_DA_11: Composition workflow
Formalized	As a user I shall be able to define the sequence of services in order to produce the desired composition output.
Technical Area	Data analytics

ID	REQ_DV-UI_1: Generate ticket
Formalized	As a system I shall be able to generate ticket(s) if a rule is fired.
Technical Area	Data visualization & user interaction

ID	**REQ_DV-UI_2:** Send e-mail
Formalized	As a system I shall be able to send e-mail to users.
Technical Area	Data visualization & user interaction

ID	**REQ_DV-UI_3: Generate report**
Formalized	As a system I shall be able to generate reports in predefined file formats.
Technical Area	Data visualization & user interaction

ID	**REQ_DV-UI_4: sms**
Formalized	As a system I shall be able to send sms to users.
Technical Area	Data visualization & user interaction

ID	**REQ_DV-UI_5: Create chart**
Formalized	As a user I shall be able to create a new chart.
Technical Area	Data visualization & user interaction

ID	**REQ_DV-UI_6: Delete chart**
Formalized	As a user I shall be able to delete an existing chart
Technical Area	Data visualization & user interaction

ID	**REQ_DV-UI_7: Edit chart**
Formalized	As a user I shall be able to edit an existing chart.
Technical Area	Data visualization & user interaction

ID	**REQ_DV-UI_8: Save chart**
Formalized	As a user I shall be able to save created/updated chart.
Technical Area	Data visualization & user interaction

ID	REQ_DV-UI_9: Browse charts library
Formalized	As a user I shall be able to browse the charts library.
Technical Area	Data visualization & user interaction

ID	REQ_DV-UI_10: Compare charts
Formalized	As a user I shall be able to compare two or more charts by adding them in the same pane.
Technical Area	Data visualization & user interaction

ID	REQ_DV-UI_11: Chart granularity
Formalized	As a user I shall be able change a chart's granularity over time.
Technical Area	Data visualization & user interaction

ID	REQ_DV-UI_12: Apply chart
Formalized	As a user I shall be able to apply a chart on a specific equipment
Technical Area	Data visualization & user interaction

ID	REQ_DV-UI_13: Export file
Formalized	As a system I shall be able to export files in pre-specified formats (images, documents)
Technical Area	Data visualization & user interaction

Non-functional requirements

ID	NFR1: Authorization
Formalized	As a system I shall ensure that data is protected from unauthorized access
Module/Area	Data collection

ID	**NFR2: Security**
Formalized	As a system I shall ensure that all external communications between the server and the clients are encrypted
Module/Area	Data collection, Data analytics, Data visualization & User interaction

ID	**NFR3: Reliability**
Formalized	As a system I shall have a POFOD=0.005
Module/Area	N/A

ID	**NFR4: Availability**
Formalized	As a system I shall have an availability of 950/1000 (95%)
Module/Area	Data visualization & user interaction

ID	**NFR5: Ease of use**
Formalized	As a user I shall need less than 3 man days to get trained on how to use the platform
Module/Area	Data visualization & user interaction

Annex II: WTC functional requirements – architectural components mapping

Table 4: functional requirements – architectural components mapping

Requirement identifier	Requirement title	Data Collection	Data Analytics	Orchestration	Data visualization
REQ_DC_1	File parsing	X			
REQ_DC_2	Non-existing meter	X			
REQ_DC_3	Out of limit values	X			
REQ_DC_4	Invalid numbers	X			
REQ_DC_5	Invalid numbers format	X			
REQ_DC_6	Invalid time format	X			
REQ_DC_7	Invalid sampling period	X			
REQ_DC_8	Missing values - gaps	X			
REQ_DC_9	Validation information	X			
REQ_DC_10	Time series	X			
REQ_DC_11	Power consumption	X			
REQ_DC_12	Energy consumption	X			
REQ_DC_13	Data values for not existent meters	X			
REQ_DC_14	Events	X			
REQ_DC_15	Meter mapping	X			
REQ_DC_16	Virtual meters	X			
REQ_DC_17	Building attributes	X			
REQ_DC_18	Building devices	X			
REQ_DC_19	Value limit	X			
REQ_DC_21	Insert incoming data	X			
REQ_DC_22	Reject incoming data	X			
REQ_DC_23	Store validation information	X			
REQ_DC_24	Save static data	X			
REQ_DC_25	Duplicates	X			
REQ_DA_1	Define metrics		X		
REQ_DA_2	Calculate metrics		X		
REQ_DA_3	Create rules		X		
REQ_DA_4	Browse rules		X		
REQ_DA_5	Apply rules		X		
REQ_DA_6	Edit rules		X		
REQ_DA_7	Delete rules		X		
REQ_DA_8	Service composition			X	
REQ_DA_9	Browse services			X	
REQ_DA_10	Test dataset			X	
REQ_DA_11	Composition workflow			X	
REQ_DVUI_1	Generate ticket				X
REQ_DVUI_2	Send e-mail				X

REQ_DVUI_3	Generate report			X
REQ_DVUI_4	sms			X
REQ_DVUI_5	Create chart			X
REQ_DVUI_6	Delete chart			X
REQ_DVUI_7	Edit chart			X
REQ_DVUI_8	Save chart			X
REQ_DVUI_9	Browse charts library			X
REQ_DVUI_10	Compare charts			X
REQ_DVUI_11	Chart granularity			X
REQ_DVUI_12	Apply chart			X
REQ_DVUI_13	Export file			X