



PLATFORM ARCHITECTURE AND DESIGN

Smarth₂O

Project FP7-ICT-619172

Deliverable D6.2 WP6

Deliverable
Version 3.0 – 2 June 2015

Document. ref.:
D6.2.POLIMI.WP6.V3.2

Programme Name:ICT
Project Number:619172
Project Title:.....Smarth2O
Partners:.....Coordinator: SUPSI
Contractors: POLIMI, SETMOB, TWUL, SES,
MOONSUB

Document Number:smarth2o. D6.2.POLIMI.WP6.V3.2
Work-Package:WP6
Deliverable Type:Document
Contractual Date of Delivery:31 December 2014
Actual Date of Delivery:31 May 2015
Title of Document:Platform Architecture and Design
Author(s):Piero Fraternali, Luigi Caldararu, Sever Calit,
Jasminko Novak, Chiara Pasini, Giorgia
Baroffio, Marco Tagliasacchi, Andrea Emilio
Rizzoli.

Approval of this reportSubmitted for approval by EC

Summary of this report:.....D6.2 Platform Architecture and Design

History:See version history.

Keyword List:platform, architecture, integration, component,
services, data model

AvailabilityThis report is public



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

This work is partially funded by the EU under grant ICT-FP7-619172

Document History

Version	Date	Reason	Revised by
2.0	30/12/2014	First submission of the deliverable	A.E. Rizzoli
3.0	2/6/2015	Revision according to reviewers requests: added a detailed description of the deployment architecture. Added two annexes describing the languages, frameworks, and technologies and the IFML schemas for the Platform front-end.	Sever Calit, Luigi Caldararu, Piero Fraternali, Giorgia Baroffio, Chiara Pasini, Andrea E. Rizzoli
3.1	3/6/2015	Version after the quality check	Sever Calit, Luigi Caldararu, Piero Fraternali, Giorgia Baroffio, Chiara Pasini, Andrea E. Rizzoli
3.2	28/7/2015	Added page numbers	A.E. Rizzoli

Disclaimer

This document contains confidential information in the form of the SmarH2O project findings, work and products and its use is strictly regulated by the SmarH2O Consortium Agreement and by Contract no. FP7- ICT-619172.

Neither the SmarH2O Consortium nor any of its officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7-ICT-2013-11) under grant agreement n° 619172.

The contents of this document are the sole responsibility of the SmarH2O consortium and can in no way be taken to reflect the views of the European Union.



Table of Contents

1. INTRODUCTION	2
2. DATA MODEL STRUCTURE	4
2.1 CONSUMER DATA MODEL	4
2.2 GAMIFICATION DATA MODEL	6
2.3 GAMES PLATFORM DATA MODEL	7
3. THE SMARTH2O ARCHITECTURE SPECIFICATION	10
3.1 SPECIFICATION OF THE USER GROUPS	10
3.2 OVERVIEW OF THE ARCHITECTURE COMPONENTS	14
3.3 WATER UTILITY CUSTOMER PORTAL	17
3.3.1 <i>Main Use Cases</i>	18
3.4 GAMIFICATION ENGINE	21
3.4.1 <i>Main Use Cases</i>	24
3.4.2 <i>Component Interfaces Specification</i>	29
3.5 GAMES PLATFORM	33
3.5.1 <i>Main Use Cases</i>	34
3.6 ENTERPRISE SERVICE BUS (ESB)	37
3.6.1 <i>Component Interfaces Specification</i>	40
3.7 SMART METER DATA MANAGER	40
3.7.1 <i>Main Use Cases</i>	41
3.8 PORTAL DATA EXCHANGE MANAGER	42
3.8.1 <i>Main Use Cases</i>	43
3.8.2 <i>Component Interfaces Specification</i>	47
3.9 WATER UTILITY ADMIN PORTAL	49
3.9.1 <i>Main Use Cases</i>	51
3.10 AUTHENTICATION GATEWAY	57
3.10.1 <i>Main Use Cases</i>	58
3.11 PRICING ENGINE, AGENT BASED MODELLING AND MODELS OF USER BEHAVIOUR	60
4. DEPLOYMENT ARCHITECTURE	62
4.1 DEVELOPMENT ENVIRONMENT DEPLOYMENT	64
4.2 PRODUCTION ENVIRONMENT DEPLOYMENT	65
5. CONCLUSIONS AND OUTLOOK	69
6. REFERENCES	70
7. ANNEX 1 – LIST OF LANGUAGES, FRAMEWORKS AND TECHNOLOGIES	71
8. ANNEX 2 – USER INTERACTION FLOWS	74
8.1 IFML IN A NUTSHELL	74
8.1.1 <i>Scope and perspectives</i>	74
8.1.2 <i>Overview of IFML main concepts</i>	76
8.1.3 <i>Role of IFML in the development process</i>	80
8.1.4 <i>A complete example</i>	80
8.2 IFML SPECIFICATION OF THE CUSTOMER PORTAL BASIC VERSION	84
8.2.1 <i>Project</i>	84

8.2.2	<i>[SiteView] Consumer Portal</i>	85
8.2.3	<i>[MasterPage] UserProfile</i>	87
8.2.4	<i>[MasterPage] UserProfile (Layout)</i>	87
8.2.5	<i>Statistics</i>	88
8.3	IFML SPECIFICATION OF THE CUSTOMER PORTAL ADVANCED VERSION	90
8.3.1	<i>Project</i>	90
8.3.2	<i>[SiteView] Private</i>	91
8.3.3	<i>Statistics</i>	110
8.4	IFML SPECIFICATION OF THE CUSTOMER PORTAL ADMIN VERSION	113
8.4.1	<i>Project</i>	113
8.4.2	<i>[SiteView] Administration</i>	114
8.4.3	<i>Statistics</i>	153

Executive Summary

This document is the Deliverable **D6.2: Platform Architecture and Design**, which, according to the Description of Work, has the following goals:

- describe the information and data models, the platform components, services, and applications, communication protocols;
- describe the integration model that enables various platform modules to interact with each other.

This deliverable will be updated as necessary during the lifetime of the project, to reflect the current status of the design, as part of the documentation accompanying the planned releases of the platform implementation (D6.3 at month 12, D6.4 at month 24, and D6.5 at month 36), in order to provide the necessary information to proceed through the three deployments foreseen for the SmartH2O platform.

The deliverable is structured as follows:

- Section 1 recalls the essential concepts at the base of the design of the SmartH2O platform.
- Section 2 illustrates the revisions to the data model underlying the platform; the description extends the original data model, presented in the deliverable D3.1 Database of user information.
- Section 3 contains the specifications of the main components of the platform:
 - o The Water Utility Customer Portal (section 3.3)
 - o The Gamification Engine (section 3.4)
 - o The Games Platform (section 3.5)
 - o The Enterprise Service Bus (section 3.6)
 - o The Smart Meter Data Manager (section 3.7)
 - o The Portal Data Exchange Manager (section 3.8)
 - o The Water Utility Admin Portal (section 3.9)
 - o The Authentication Gateway (section 3.10)
 - o The preliminary specifications of the Pricing Engine, Agent Based Modelling, and Models of User Behaviour components conclude the section; they will be expanded following the progress in the technical work.
- Section 4 describes the deployment architecture: in particular how different technologies interplay in the implementation of the Service Oriented Architecture of SmartH2O.
- Section 5 concludes the deliverable with an outlook on the next steps in the design of the SmartH2O platform.
- At the end of the deliverable, after the References section, Annexes provide additional technical information on:
 - o [ANNEX 1 \(Section 7\)](#): The list of languages, frameworks and technologies used in the platform development.
 - o [ANNEX 2 \(section 8\)](#) after a brief introduction to the OMG Interaction Flow Modelling Language [Bramb2014] (the UML profile for interaction flow specification in interfaces) the specifications of the SmartH2O front-end are given, divided in three subsections: Section 8.1, Section 8.2, Section 8.3 contain the interaction flow diagrams for the front-end of the SmartH2O platform, according to the IFML OMG standard.

1. Introduction

The SmartH2O project is developing an ICT platform for improving the management of urban and peri-urban water demand, based on the integrated use of smart meters, social computation, and dynamic water pricing, powered by advanced models of consumer behaviour.

The SmartH2O platform can support the cooperation of water utilities, municipalities and citizens to implement better water management practices and policies, leading to a reduction in water consumption, without compromising the quality of life, and to an increase in resource security.

The SmartH2O project must be able to:

- Understand and model the consumers' current behaviour on the basis of historical and real-time water usage data;
- Predict how the consumer behaviour can be influenced by various water demand management policies, from water savings campaigns, to social awareness campaigns, to dynamic water pricing schemes;
- Raise the awareness of water consumers on their current water usage habits and their lifestyle implications and to stimulate them to reduce water use.
- Integrate with other systems, such as the portal of the utility company or a digital game platform, in order to provide a coherent experience to the water consumer.

The SmartH2O ICT infrastructure will thus enable water managers to close the loop between actual water consumption levels and desired targets, using information about how the consumers have adapted their behaviour to the new situation (e.g. new regulations, new water prices, and appeals to water savings during droughts). This feedback will then allow water management companies and decision makers to aptly revise the water demand management policies, enabling to maximise the water and energy saving goals.

Figure 1, originally provided in the SmartH2O description of work, describes the envisioned flow of information and control supported by the SmartH2O platform.

The behaviour of a water user is gathered by collecting data on its water consumption by means of smart meters and, at the same time, by an online social participation application (the social game) where qualitative information about the user preferences and attitudes are collected. This information is processed and the user is "profiled" in order to produce a synthetic user behaviour model, which will be then fed in an agent-based simulation model.

The social participation applications are then also used to deploy policies in the real world. The consumers will receive signals, such as incentives to save water in specific environmental conditions, or such as dynamic price information. Once the policies are deployed, the SmartH2O platform allows continuous monitoring of the users' aggregate behaviour, i.e. their water consumption, in order to suggest other actions if the original policy loses effectiveness.

This deliverable reflects the current status of the design of the technical components that map the data and control flows depicted in Figure 1 into a concrete technical space, assigning computation and integration responsibility to core software modules.

For each core module, this deliverable reports the essential technical use cases and interface that the module supports. The technical use cases must be interpreted as a derivation of the high-level use cases described in the requirements deliverable: *D2.1 Use cases and early requirements*. They embody the flow of messages, queries, and service calls that support the actual execution of the user-oriented usage scenarios specified in D2.1.

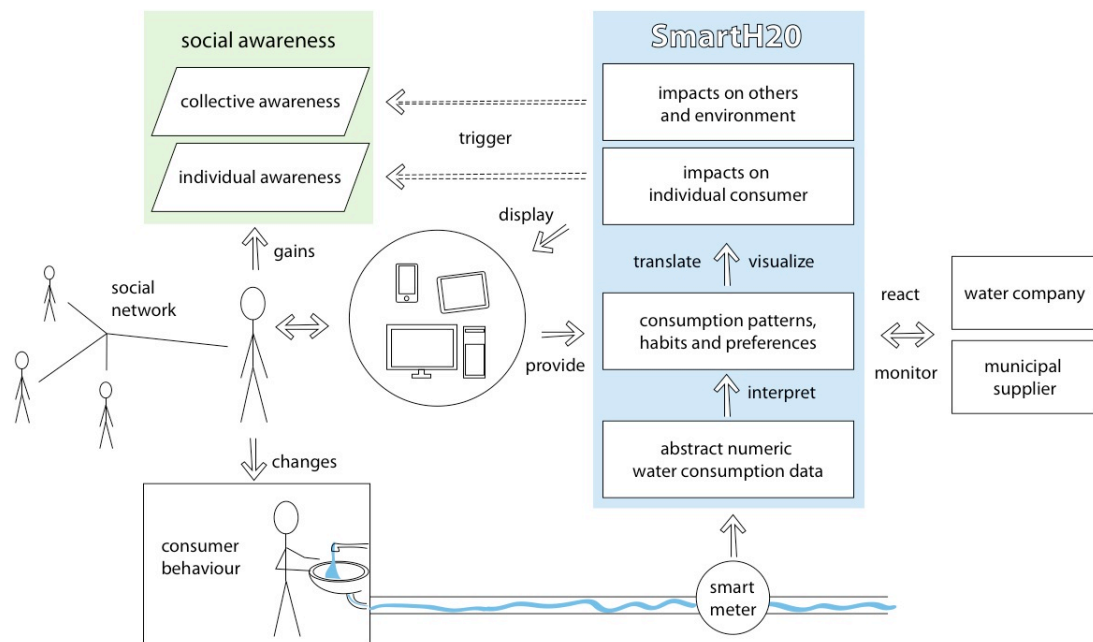


Figure 1: The flow of information and control in SmartH2O.

2. Data Model Structure

In this section we describe the updated version of the Smarth2O database, previously defined in deliverable *D3.1 Database of User Information*. Such updates stem from the progress in the specification of the user's requirements and of the system architecture, which prompted for the refinement of aspects mostly related to the user's profile data.

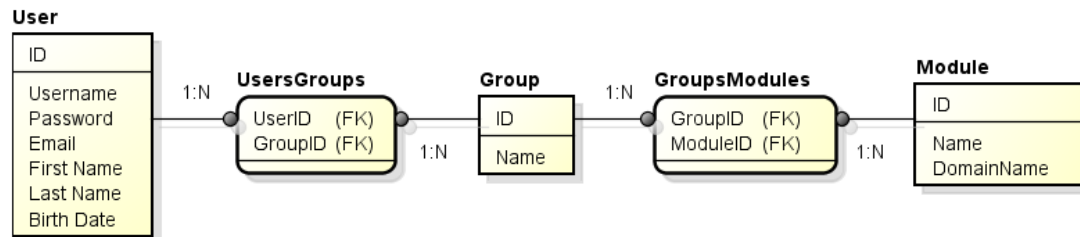


Figure 2: User Groups Relational Model.

Figure 2 recalls the data model for implementing Role Based Access Control (RBAC). Users are clustered in Groups, which represent the various classes of users. Groups are connected to Modules, which represent the interfaces to the Smarth2O resources that the class of users is entitled to access.

2.1 Consumer Data Model

According to the early mock-ups and functional requirements illustrated in *D2.1 Use cases and early requirements*, a series of changes have been made to the Consumer Data Model proposed in the deliverable D3.1 Databases of user information.

The resulting updated schema is reported in Figure 3:

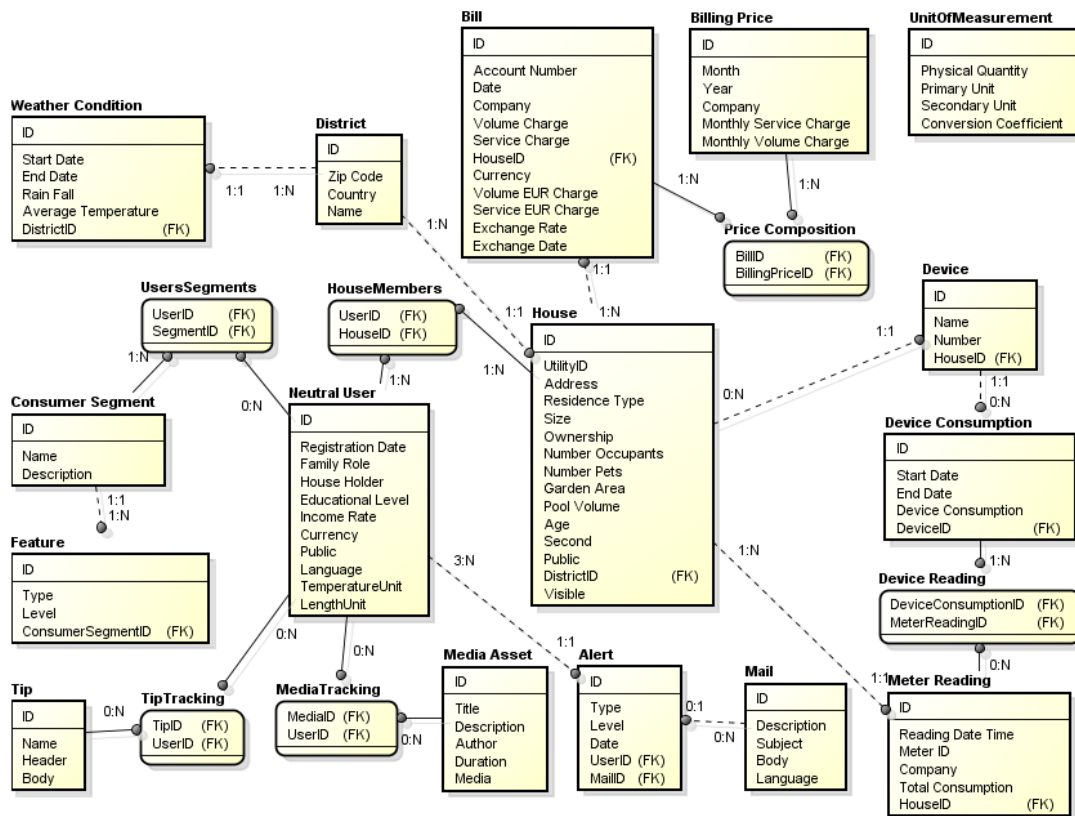


Figure 3: Consumer Relational Model.

In order to provide more appropriate and targeted incentives, users are grouped into consumer segments, using algorithms provided by the Models of User Behaviour component. The following entities have been introduced:

- **Consumer Segment**: each segment is identified by unique id, a name and a description. A segment of users is characterized by a set of features.
- **Feature**: each feature refers to a specific consumer segment, and it is identified by a unique id, a type and a level (e.g. Consumption Average: medium, Environmental Commitment: high).
- **Users Segments**: this entity stores the members of each user segment.

In order to encourage a rational use of water some tips are provided to the consumers. The following entities have been introduced:

- **Tip**: each tip is identified by a unique id, a name and the text content divided into a header and a body.
- **Tip Tracking**: this entity keeps track of the tips which have been provided to a user.

In order to warn users about possible leaks or bad water quality, alerts will be provided. The following entities have been introduced:

- **Alert**: each alert is identified by a unique id, a type (e.g. Water Quality Alert, Leakage Alert, Shortage alert), a level (e.g. low, medium, high) and the receiver user id. When a new alert is inserted, the current date is stored in order to keep track of the progress of a particular type of alert and to record past critical situations. However the latest alert of a given type will be provided to the user.
- **Mail**: an alert can be associated to a mail, in order to directly notify the user. Each email is identified by a unique id, a description, the subject of the email, the body of the email and the language.

The system deals with different physical quantities (e.g. water consumption, temperature). In order to take into account the different units of measurement used in different countries, the system will store them in order to make the correct unit conversions. The following entity has been introduced:

- **Unit Of Measurement:** each unit conversion is characterized by a unique id, the primary unit of measure, the secondary unit of measure and the coefficient to be applied in order to perform the conversion.

The system can also display water info material like videos, providing information about topics related to water saving. The following entities have been introduced:

- **Media Asset:** each media object is identified by a unique id, a title, a description, the author, the duration of the video and the URL of the media object.

Additionally some attributes have been introduced in the following entities:

- House:
 - Public: a flag stating if the house holder user agrees that his/her family is involved in a public competition and consents to appear in the public leaderboard (D2.1: use case 9.19).
 - Visible: a flag stating if the house holder user consents to disclose information about his house location to other users (D2.1: use case 9.21).
- Neutral User:
 - Currency: the currency selected by the user, which will be associated to the bill quantities.
 - Language: the language of the user.
 - Temperature Unit and Length Unit: the units of measurements selected for a given user.
 - Public: a flag stating if the user agrees to participate to the competition and to appear in the family leaderboard (D2.1: use case 9.17).
- Bill:
 - Currency: the currency of a given bill.
 - Volume Charge and Service Charge: they are needed in order to be able to sort bills and compare them.
 - Exchange Rate and Exchange Date: they are needed in order to store the conditions at the time when the bill was emitted.

2.2 Gamification Data Model

According to the early mock-ups and functional requirements, the changes illustrated in the following have been made to the Gamification Data Model proposed in the deliverable D3.1 [Bozzon2014, Karam2012].

The resulting updated schema is depicted in Figure 4.

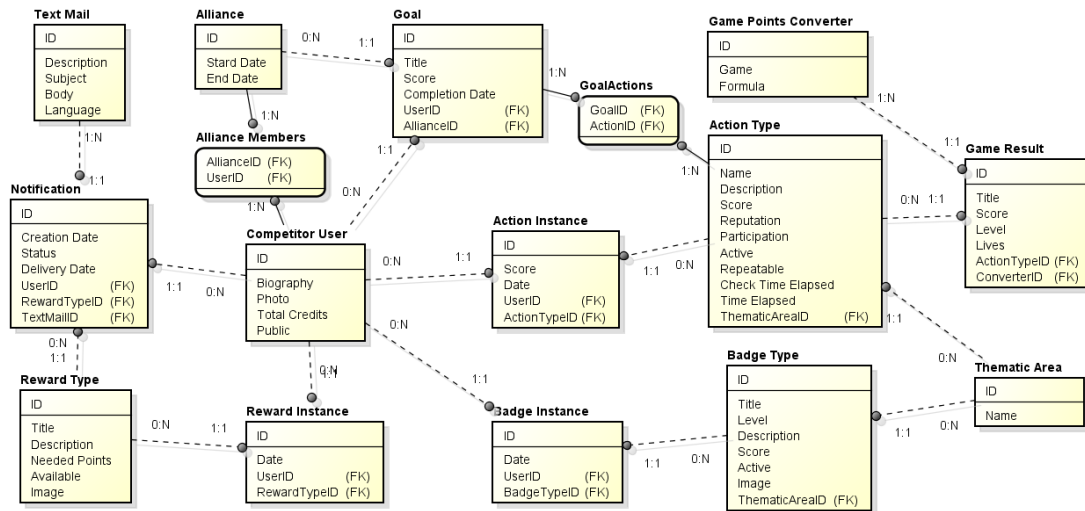


Figure 4: Gamification Relational Model.

In order to allow the Content Editor of the Gamification Engine portal to organize actions and badges according to topics, the thematic area entity has been introduced:

- **Thematic Area:** each thematic area is identified by a unique id and a name. Each action or badge belongs to a specific thematic area.

In order to allow the Content Editor of the Gamification Engine to define the rules for the actions coming from game results, which are involved in the gamified competition, the following entities have been introduced:

- **Game Result:** each game result is identified by a unique identifier, a title (e.g. New Level Reached) and optionally by a score, a level and the current available lives. Each game result is mapped to an Action Type and, according to the game results attributes (score, level, lives) the game result is converted into gamification engine credits.
- **Game Points Converter:** each conversion is identified by a unique id, the game to which the conversion rule is applied, and the customizable formula which will take the attributes as inputs (score, level, lives) and will produce gamification credits as output.

The application will provide water-saving goals to the users, to be achieved by performing different actions. Users can also form coalitions, inviting the rest of his family to join, and save water collaboratively with friends, family and neighbours. The following entities have been introduced:

- **Alliance:** each coalition among competitor users is identified by a unique id, a start date and an end date.
- **Alliance Members:** this entity stores the members of each alliance.
- **Goal:** each goal is identified by a unique id, a title, a score, and optionally the completion date. A goal can be assigned to a given user or to an alliance of users.
- **Goal Actions:** this entity stores which actions can be performed by users to achieve a particular goal.

2.3 Games Platform Data Model

Figure 5 illustrates the schema of the database supporting game data persistence [Bozzon2014, Karam2012].

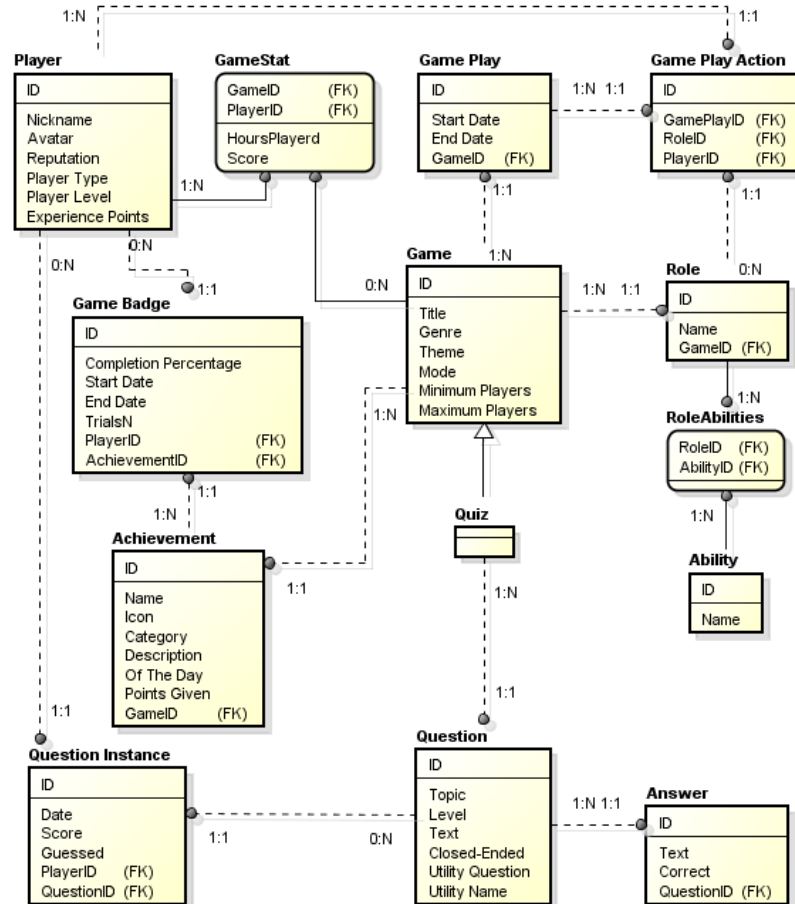


Figure 5: Games Platform Relational Model.

Game is the core entity: the *Mode* attribute represents the gameplay modes (e.g. Single Player, Multi Player, Cooperative), while the *Genre* attribute identifies its genre (e.g. Puzzle, Educational). Each game is also characterized by a *Title*, a *Theme* and the *Minimum/Maximum number of players*.

An **Achievement** has an *Icon*, which describes it in a visual way, a *Category* that specifies the task (Instructor, Grinder), an attribute *PointsGiven*, which contains the amount of points to be granted, and a Boolean attribute *OfTheDay* defining whether the achievement has to be completed on a specific day in order to obtain virtual goods, more points, or increased levels.

The **Player** entity accommodates game-specific personal and social features. *Avatar* and *Nickname* allow the user to be recognizable by using a custom image or a unique fictional name, while *Player Type*, *Player Level* and *Experience Points* convey player progress. *Reputation* in online gaming communities is fundamental and distinctive feature of any player; being able to recognize whether a player is bad mannered, prone to cheating, unpleasant to play with is of utter importance to assure a satisfying gaming experience for the user of an entertainment platform; it is usually measured as an integer number ranging from 0 to 5.

The model describes also the game-relevant statistics (**GameStats**): the proficiency and the experience of a player in a given game are represented by aggregating in a compact way

such indicators as points gathered and hours spent playing.

GameBadges represent the achievements that have been unlocked by a player. The *CompletionPercentage* field shows how much the player has already achieved in a specific task. *StartDate* and *EndDate* record the dates in which the player has started to work on the achievement's goals and the date in which he has obtained it. The *TrialsN* attribute tracks how many times the user tried to fulfill the achievement.

A **GamePlayAction** of a player, associated with a specific **Gameplay**, records the *StartDate* and *EndDate* of the gaming session and the actual actions performed by the player on that specific time frame and the *Role* defines which are the allowed actions in the game for the role associated to a player.

In order to store questions and answers required by the Drop!TheQuestion trivia game, **Question** and **Answer** entities have been provided. **QuestionInstance** keeps track of players game play information related to the specific quiz game.

3. The Smarth2O architecture specification

The Smarth2O platform is a distributed architecture for brokering data and services among a variety of heterogeneous systems and users, supporting the flows depicted in Figure 1.

3.1 Specification of the user groups

To better understand the nature of the user interfaces visible in Figure 10, Figure 6 recalls the taxonomy of the various user roles identified in the requirement analysis, as reported in *D2.1 Use cases and early requirements*.

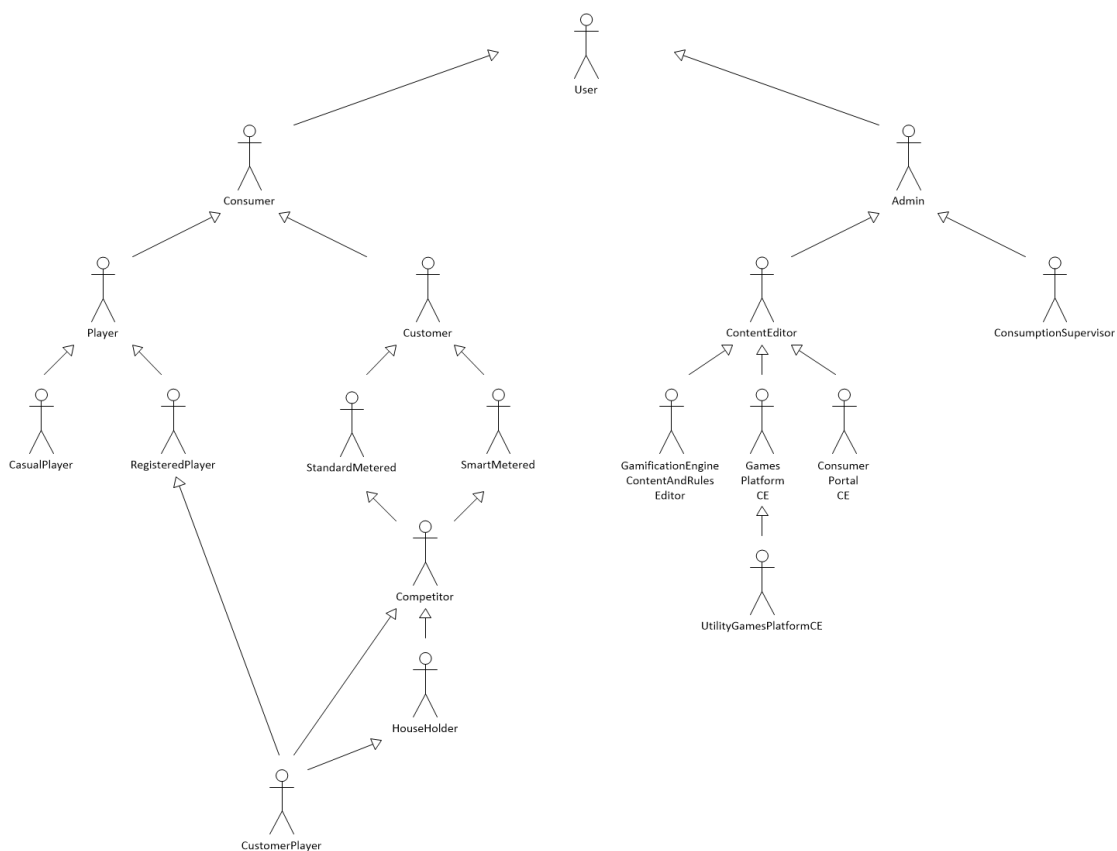


Figure 6: Taxonomy of the user groups in Smarth2O (from D2.1 Use Cases and Early Requirements).

A first distinction among user groups is between **Consumer** and **Admin**: the former is the generic user who can access the services provided by the Smarth2O platform; the latter is instead in charge of managing the services provided by the Smarth2O platform.

Consumers are partitioned into sub-groups based on which services they access.

- **Player** users are the ones who play the Games provided by the Smarth2O platform. They can be:
 - **Casual Players**: they are not registered visitors interested in playing a game.
 - **Registered Players**: they are registered to the Games platform.
- **Customers** are users registered to the Consumer Portal, who access in order to monitor their water consumption and water bill. They can be:

- **Smart metered** users: they are customers having smart meters system installed in their house. The water meter measures the customer's water consumption automatically.
- **Standard metered** users: they are customers not having smart meters system installed in their house. They need to manually input consumption data into the gamification engine.
- Inheriting properties from both Smart metered and Standard metered consumers, we find the following specialisations:
 - **Competitor** users: they are the ones who accepted to participate to the gamification mechanisms, including execution of actions, acquisition of badges and redemption of rewards. A competitor can be a:
 - **House holder** user: he/she is the responsible of a specific house. He/She can add other family members to the Gamification Engine and create collaborations with neighbours.
 - At the bottom of the hierarchy, inheriting from House holders, Competitors and from RegisteredPlayers, we find:
 - **CustomerPlayers** are users who are registered both to the Gamification Engine and the Games Platform. They have the possibility to collect points either by performing actions provided in the gamification engine or by playing the available games.

Admin users are partitioned into sub-groups based on which services they manage.

- **Content Editors** are administrators in charge of creating the content of the applications composing the smart water system.
 - **Gamification Engine Content and Rules Editors**: they are in charge of creating the content related to the gamification platform (the one used by Competitor users) such as actions, rewards and goals. They are also in charge of defining the rules to assign the suitable amount of points to each action.
 - **Consumer Portal Content Editors**: they are in charge of creating the content related to the platform used by Customer users, such as tips to improve water saving, teaching videos.
 - **Games Platform Content Editors**: they are in charge of creating the content related to the games, such as the questions provided in a quiz game related to generic water consumption topics.
 - **Utility Games Platform Content Editor**: they are a specialization of the Games Content Editor users, related to a specific utility game. For example they manage the specific questions provided in a quiz game.
- **Supervisors** are administrators in charge of monitoring and managing system data. They can be:
 - **Consumption Supervisor**: they are in charge of modelling user's consumption.
 - **Gamification Engine Supervisor**: they are in charge of profiling users, making available users clusters that can be used to suggest the most suitable actions to perform.

Figure 7, Figure 8 and Figure 9 show the essential UML class diagram that represents the user groups that implement the taxonomy of Figure 6.

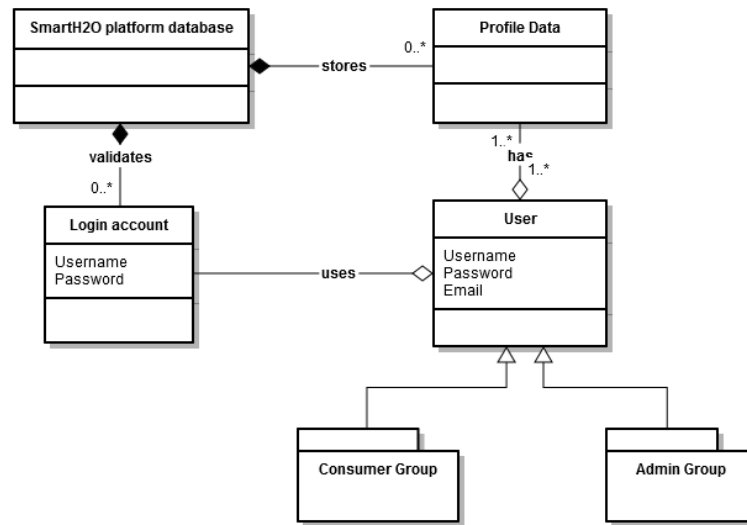


Figure 7: User groups class diagram.

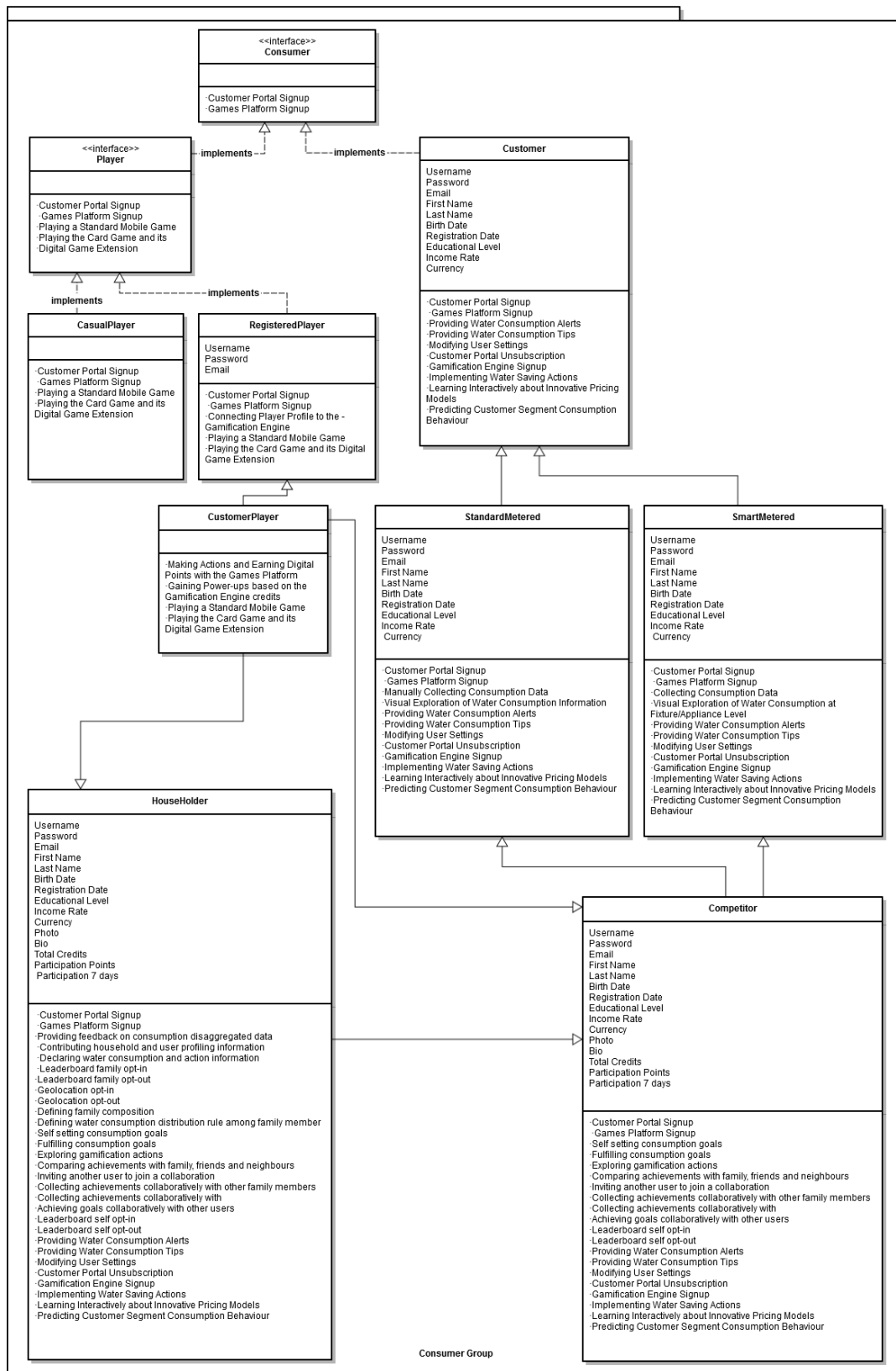


Figure 8: Consumer group package class diagram within user groups.

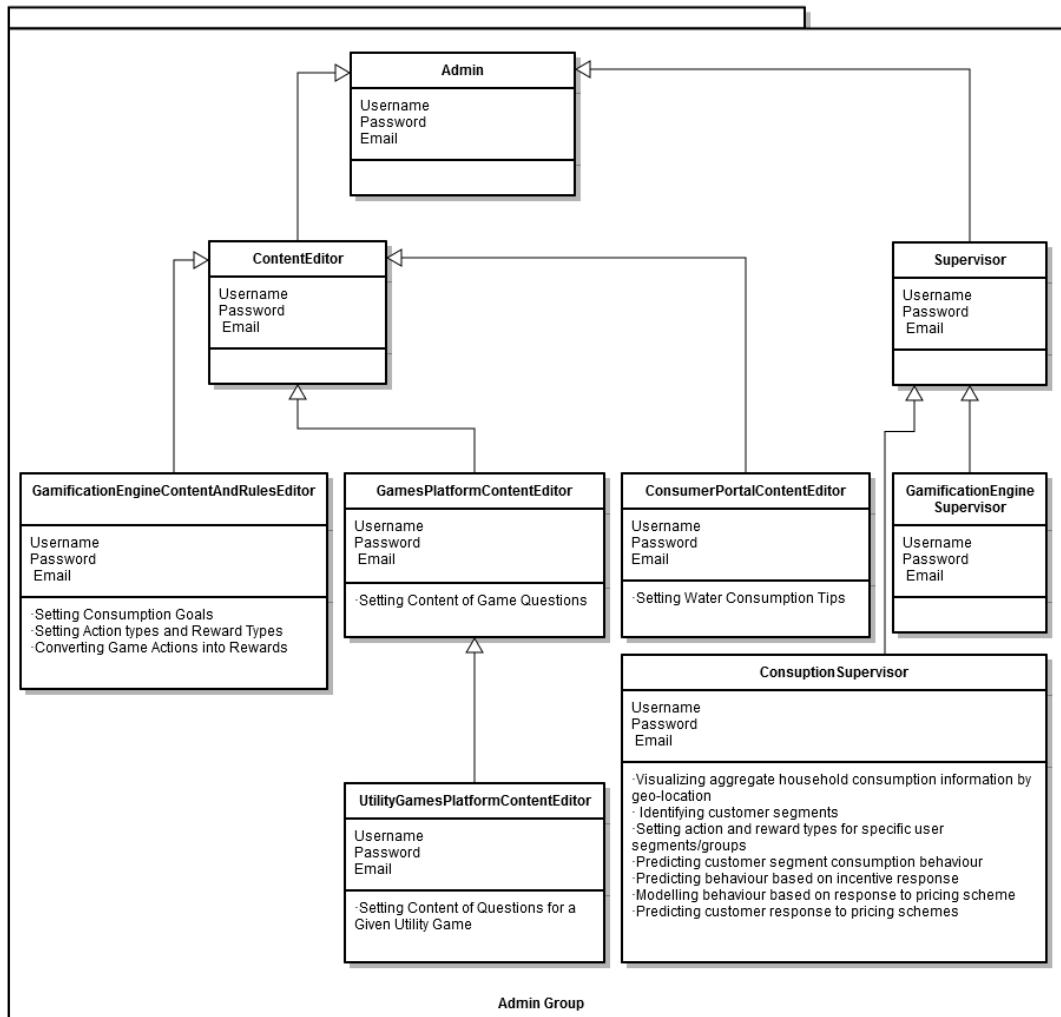


Figure 9: Admin group package class diagram within user groups.

3.2 Overview of the architecture components

Figure 10 illustrates the main components that constitute the SmartH2O platform.

The **SmartH2O Database** is the central repository of the information that is either common to all the SmartH2O components or supports the coordination and exchange of messages among them. Not all the data of SmartH2O will reside in the SmartH2O database; for example, commercial data about the water consumers maintained by the water utility will be stored in the proprietary systems of the company.

The **Enterprise Service Bus** (from now on, **ESB**) is a middleware layer that supports the loose coupling of the SmartH2O components; it permits the publication of their interfaces (Application Programming Interfaces, APIs) and the synchronous and asynchronous communication among components. The goal of the ESB is to decouple the heterogeneous components of the SmartH2O platform as much as possible, to support the extension with new services and functions and the rapid adaptation of the platform to new contexts (e.g., other utility companies with different IT standards and information systems).

The **Smart Meter Data Manager** deals with the acquisition of data streams from smart meters and with their consolidation within the SmartH2O database. It implements the data privacy and security policy of the utility company and ensures that only admissible (e.g.,

aggregated, anonymised) data is stored within the platform database.

The **Water Utility Consumer Portal** is a component, typically embedded within the proprietary portal services of the utility company, which supports the interaction between the utility customers and the SmartH2O awareness functionality. The integration is lightweight: the consumer will navigate from the standard GUI of the utility company to what she sees as a special section of the portal, where she can access the awareness tools and interfaces developed by SmartH2O.

The **Water Utility Admin Portal** is a component, integrated within the proprietary portal services of the utility company, which supports the work of the supervisor in the analysis of the water consumption data and of the outcome of the gamification rules; it also supports the work of the content editor, who administers the content (e.g., tips, articles, news, etc.) published to the customers. The portal also offers interfaces to the water utility operators to run simulations, based on the models embodied in the Models of User Behaviour component and on the algorithms implemented in the Pricing Engine and in the Agent Based Modelling component.

The **Portal Data Exchange Manager** deals with the data exchange communication that occurs “behind the scenes” among the SmartH2O platform and the third party applications already supporting the interaction with the various types of users. Such applications may comprise the “standard” customers’ portal of the utility company, or a B2E application for the utility company’s supervisors.

The **Gamification Engine** is a back-end component that embodies rules for transforming users’ actions into gamification scores and achievements. It is exploited in order to “gamify” the water consumption of the users, according to the awareness approach implemented by SmartH2O. It has an interface for the end-user, who sees the results of her water consumption actions; and administrative interfaces for the utility company’s managers and operators, who can supervise the outcome of the awareness policies and define the rules that reward the actions of water consumers.

The **Games Platform** supports the execution of all the digital games of SmartH2O, including the games that are played as part of the interaction with the Drop! board game (for a description of the current status of the social awareness applications, including the SmartH2O games, see deliverable *D4.1: First social game and implicit user information techniques*). The Games Platform must also support casual players, and thus has an independent users’ registration procedure, as well as a procedure for enrolling users that are already registered in the Utility Portal. The Games Platform exposes two kinds of interfaces: one or more digital games directed to the end users; an administrative interface, directed to the content editors of the game platform. The GUIs are served by a local database (the Games DB), which stores information that is pertinent only to the game play (e.g., the gaming history of players not registered in the Utility Portal).

The **Pricing Engine** allows water utility companies to assess the impact for various dynamic pricing algorithms on their customer behaviour. The pricing engine will use consumption data, user profile data, external input like meteorological data, water supply forecast. The Pricing Engine will model the user elasticity to different pricing schemes and it will be able to report how pricing stimuli can impact aggregate customer behaviour. The Pricing Engine also allows the Consumer to estimate the cost of its water use according to different pricing schemas, ranging from the actual tariff, to various simulated tariffs, which are evaluated in the SmartH2O project.

The **Models of User Behaviour** component contains models and algorithms for profiling the behaviour of water consumers. It contains a classification algorithm that creates user segments (classes of users with similar behaviour) on the basis of their features. It also contains a disaggregation algorithm that can attribute the end uses of the total amount of water used by a household during one day, with a certain degree of approximation. This algorithm is also used to identify the relevant features to be used in classification. Through the use of the SmartH2O platform supplemental features will be generated, such as the influence of social awareness (obtained by the Gamification component) or the sensibility to price changes (obtained by the pricing engine).

In summary, through this component, the water utility can visualize the water consumption of

each customer at a fixture/appliance level, in order to identify consumption patterns and trends, and thus identifying the most promising areas where conservation efforts may be polarized. For a description of the algorithms exploited to model the user behaviour, see deliverable *D3.2: First user behaviour models*.

The **Agent Based Modelling** component allows the water utility to simulate whole districts of users, thus extrapolating user models provided by the Models of User Behaviour component at a larger scale and also extrapolating the impact of network effects due to users' interactions, both in the physical and in the virtual world. The agent based model includes influence/mimicking mechanisms and social interaction among the consumers, and thus will be employed by the water utility to understand how some user types (leaders/influencers) can stimulate a behavioural change on other users.

The **Authentication Gateway** component centralizes user registration into the Smarth2O platform database for the users registered in the components having own user database.

Also, this component provides an unique point for authentication to all the users primarily registered at a component level, in order to allow logging in to other components by using the original set of credentials without the need to perform another registration.

The **Social Network Crawler and Data Analyser** component allows the platform, where deemed appropriate by the water utility portal, to launch social data analysis campaigns to identify relevant users and content in the area of sustainable water consumption. For example, this component supports the crawling of Twitter data in order to automatically find people and content relevant for a thematic area, such as water consumption.

The **Social Network Connector** component has a dual role with respect to the Social Network Crawler and Data Analyser; it allows the Consumer, Player, and Competitor users to post their achievements from the Smarth2O Water Utility Portal and Games Platform to the social network of their preference, in order to engage people from their social circle to the water consumption and sustainability campaigns of the water utility company

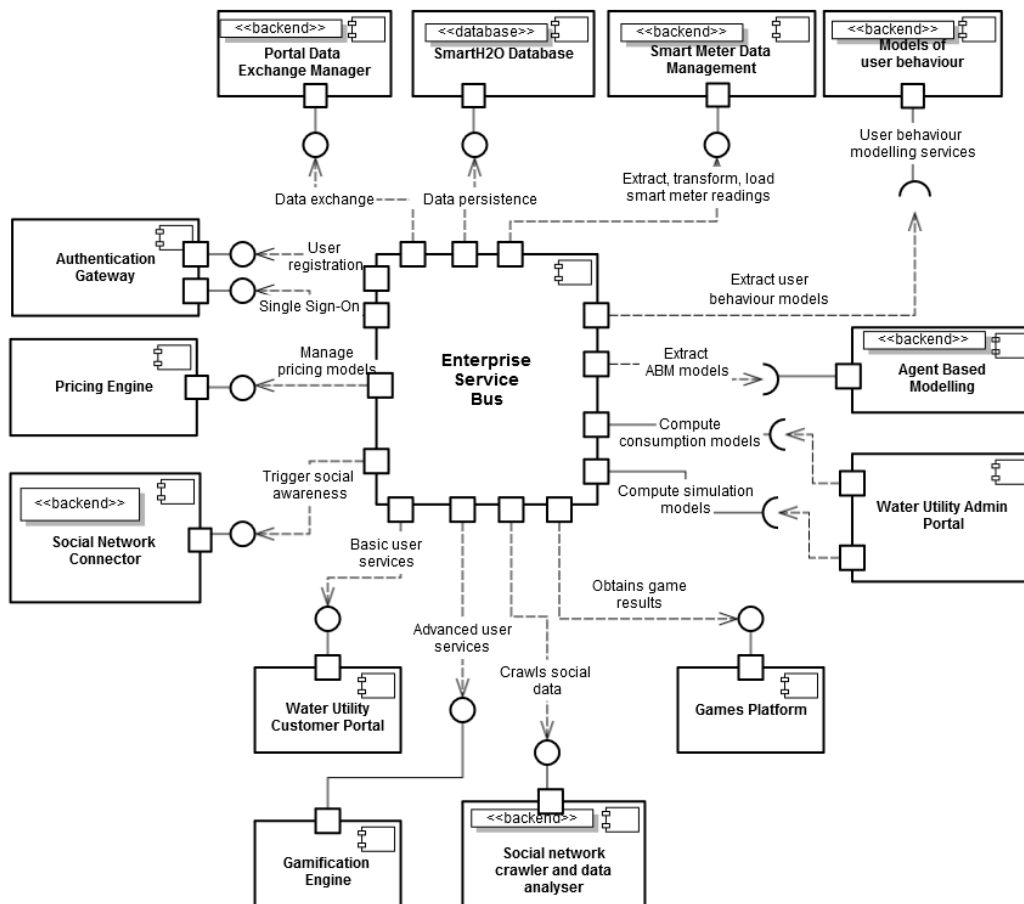


Figure 10: overview of the main components of the Smarth2O architecture.

3.3 Water Utility Customer Portal

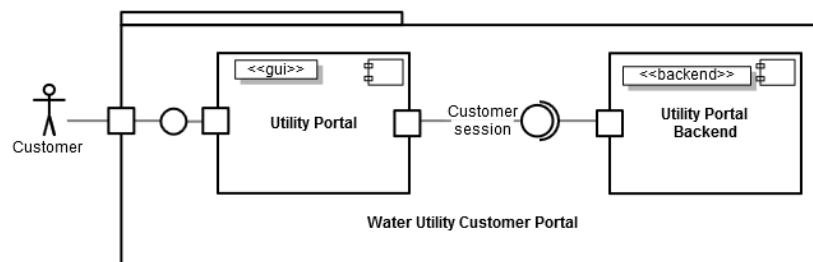


Figure 11: Water Utility Customer Portal Component diagram.

This component includes:

- the **Utility Portal** in which the user can see his household consumption, water saving tips and alerts, and water saving educational content. This UI integrates inputs from different components (e.g. the smart meter data management)

Component Name	Water Utility Consumer Portal
Leader	POLIMI
Partners	SETMOB
Goals	<i>The component manages the visualization of the water consumption status for a Customer User</i>
User use cases	8.2 <i>Use case: Collecting consumption data</i> 8.3 <i>Use case: Manually collecting consumption data</i> 8.6 <i>Use case: Utility Portal Signup</i> 8.7 <i>Use case: Visual exploration of water consumption information</i> 8.8 <i>Use case: Visual exploration of water consumption at fixture/appliance level</i> 8.9 <i>Use case: Providing water consumption alerts</i> 8.11 <i>Use case: Providing water consumption tips</i> 8.12 <i>Use case: Modifying User Settings</i> 8.13 <i>Use case: Utility Portal Unsubscription</i>
Provided Interfaces	<i>This component is a web application, which integrates inputs from external sources. It does not provide interfaces to other components.</i>
Dependencies / Required Interfaces	<i>The component requires the implementation of the following interfaces:</i> - <i>[GET]user(userid): returns all the metadata for the specified user</i> - <i>[GET]exists(userEmail): returns true if a user with the specified email exists, false otherwise</i> - <i>[GET]billByUser(userId): returns a short recap of the bill for the logged user</i> - <i>[GET]waterConsumptionStatisticsByUser(userId) returns the statistics of water consumption for the logged user</i> - <i>[GET]tips(userId): returns a list of tips for a specified user</i> - <i>[GET]waterUsageAlerts(userId): returns the list of usage alerts for the logged user</i> - <i>[GET]mediaContent(userID): videos, tutorial etc: may depends on the users</i> - <i>[POST]signup: pushes a registration to the SmarH2O DB</i> - <i>[POST]unsubscribe: remove a registration to the SmarH2O DB</i> - <i>[POST]updateUserProfile: updates user profile information like: family members, house configuration, appliances etc..</i>

3.3.1 Main Use Cases

Use Case 1.1: House Holder Signup

Use Case 1.1		House Holder Signup
Goal in Context		<i>The consumer performs a registration to the Water Utility Customer Portal, his data are saved into the Smarth2O DB.</i>
Precondition		<i>The consumer is a client of the utility that manages the portal.</i>
Success Condition	End	<i>The user becomes a Customer user, who can access the Utility Portal.</i> <i>The consumer is correctly saved into the DB.</i>
Failed Condition	End	- <i>A registration with the specified email already exists, this registration is cancelled and the user is notified.</i> - <i>The provided Utility Code does not exist or does not match the user information; this registration is cancelled and the user is notified.</i>
Primary actors		<i>Customer, UtilityPortal</i>
Secondary Actors		<i>ESB, Smarth2ODB, Authentication Gateway</i>
Steps		- <i>the Customer requests the registration UI</i> - <i>the Customer fill and send the registration form</i> - <i>Water Utility Customer Portal verifies if the email was already registered</i> - <i>Water Utility Customer Portal verifies if the Utility Code exists and is valid</i> - <i>Water Utility Customer Portal saves the new user into the Smarth2O DB</i>
Note		<i>Fields in the registration form (see 2.1 Consumer Data Model): address, residenceType, size, ownership, #occupants, #pets, gardenArea, poolVolume, age, second, public, districtID, hhEducationLevel, hhIncomeRate, hhName</i>

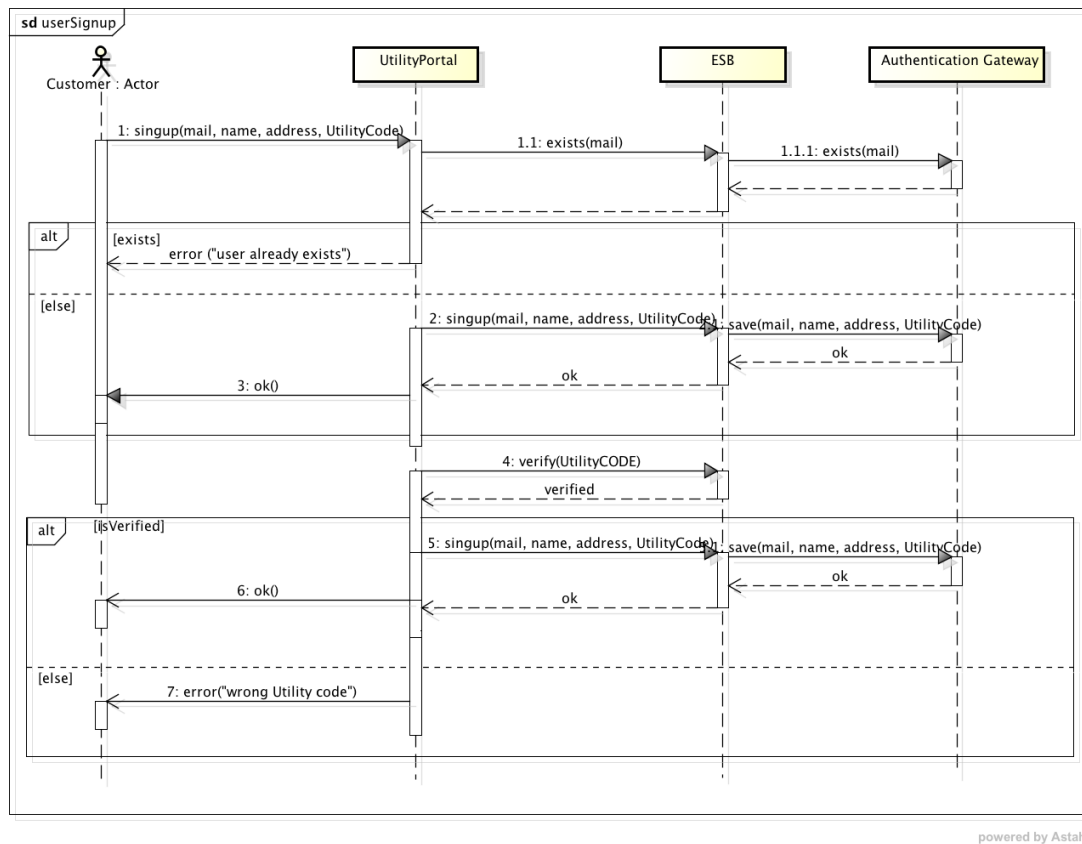
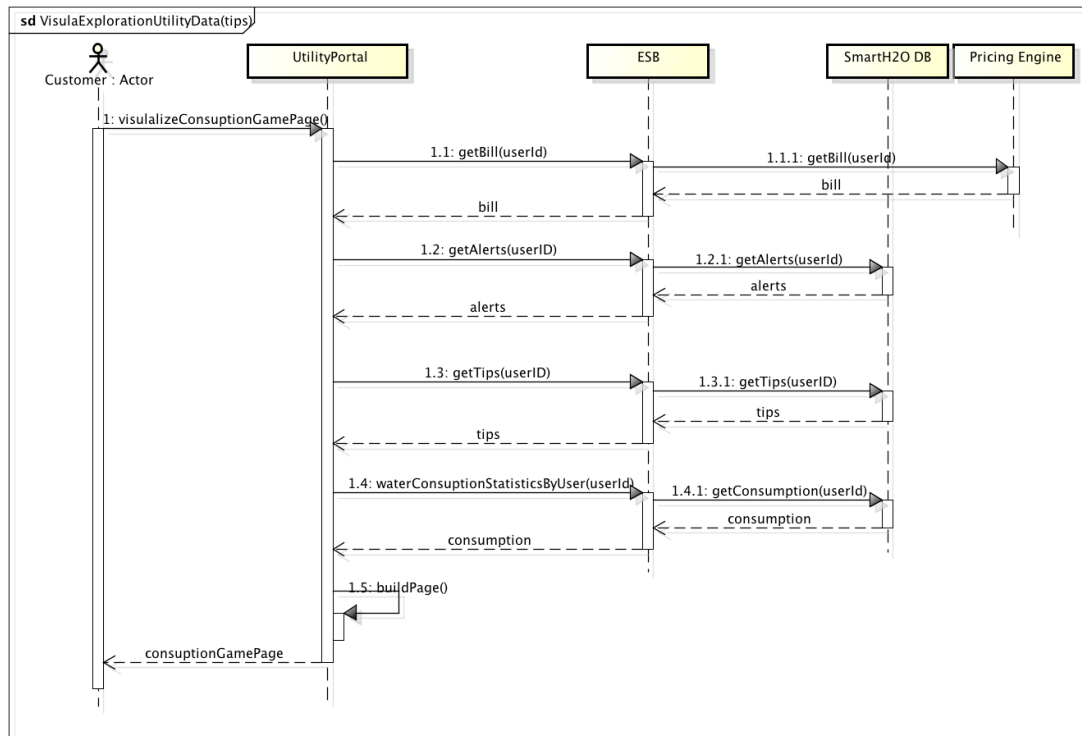


Figure 12: Sequence diagram Use Case 1.1.

Use Case 1.2: Visual Exploration Utility Data

Use Case 1.2		Visual Exploration Utility Data
Goal in Context		The consumer explores his water consumption visualizing his actions, his bills, tips, and alerts.
Precondition		The consumer is logged to the platform
Success Condition	End	The consumer is able to visualize his data
Failed Condition	End	--
Primary actors		Consumer, UtilityPortal
Secondary Actors		ESB, Smarth2ODB
Steps		- The customer requires to visualize the exploration page - The Utility portal retrieves all the data and builds the page
Note		--



powered by Astah

Figure 13: Sequence Diagram UseCase 1.2.

3.4 Gamification Engine

The Gamification Engine is a component that listens to the actions of the Competitor consumer and of the Player and transforms them into a variety of rewards, for improving activity and participation.

The need for a Gamification approach in SmartH2O has been described in D2.1 Use cases and early requirements, by detailing the possible use case and motivations beyond this choice. In particular, the definition of points, achievements and rewards allow water utility companies to challenge their users with water saving goals to be achieved each month to obtain individual water consumption and household data provisioning. The users of the platform can be stimulated to adopt water saving behaviours by offering them gamified pricing schemes and water consumption data, to make them aware of the impact of their actions and the needs of the water utility company. The **Gamification Engine** is also used as a mean for raising water consumption awareness by promoting sustainable behaviours for families, friends and neighbours.

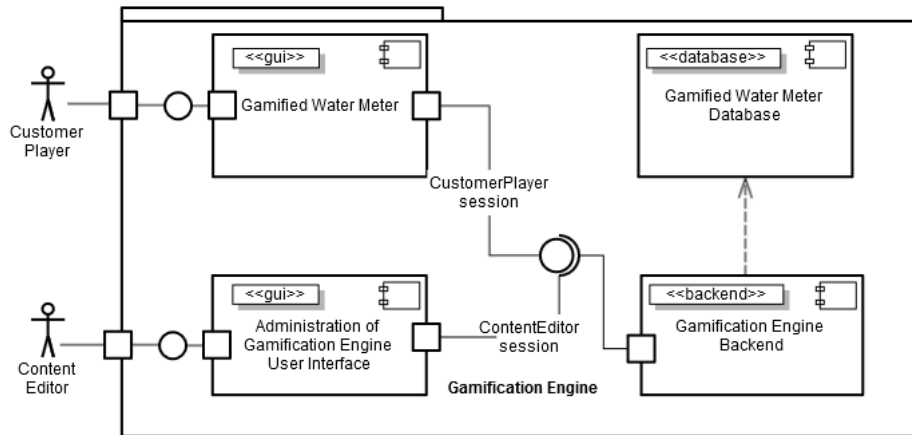


Figure 14: Gamification Engine component diagram.

As shown in Figure 14, the **Gamification Engine** is the central component that handles the communication with the main SmarH2O platform components and takes care of registering users, converting users' actions in other components into gamified actions for SmarH2O in order to compute scores, badges, achievements and other gamified features and being able to track and profile the users.

All the gamified data are stored in a **Gamification Engine DB**, in order to decouple the data from the various water utilities portals with the one managed by SmarH2O.

To extend the features of the water utility customer portal, a **Gamified Utility Portal UI**, tailored to the specific needs of the competitor users is created, in order to let the users of this group access their gamified profile, the points collected, the achievements obtained, and redeem the earned rewards.

The Gamification process has to be interactive and dynamically tailored to the rapid changes requested by the scenarios and the community at hand; for these reasons, new goals, rewards and achievement can be added in the platform by non-technical users through the **Gamification Engine Admin UI**.

With the same interface, but less privileges, admins can also monitor the status of the gamification campaign.

Component Name	<i>Gamification Engine</i>
Leader	<i>POLIMI</i>
Partners	--
Goals	<i>The component manages gaming scores and achievements and visualize the user water consumption</i>
User use cases	9.3 <i>Use case: Gamification Engine Signup</i> 9.5 <i>Use case: Self setting consumption goals</i> 9.6 <i>Use case: Fulfilling consumption goals</i> 9.7 <i>Use case: Implementing water saving actions</i> 9.8 <i>Use case: Contributing household and user profiling information</i> 9.9 <i>Use case: Declaring water consumption and action information</i> 9.10 <i>Use case: Exploring gamification actions</i> 9.12 <i>Use case: Comparing achievements with family, friends and neighbours</i> 9.13 <i>Use case: Inviting another user to join a collaboration</i> 9.14 <i>Use case: Collecting achievements collaboratively with other family members</i> 9.15 <i>Use case: Collecting achievements collaboratively with neighbours</i> 9.16 <i>Use case: Achieving goals collaboratively with other users</i> 9.17 <i>Use case: Making actions and earning digital points with the games platform</i> 9.18 <i>Use case: Converting game actions into rewards</i> 9.19 <i>Use case: Gamification Engine self opt-in</i> 9.20 <i>Use case: Gamification Engine self opt-out</i> 9.21 <i>Use case: Gamification Engine family opt-in</i> 9.22 <i>Use case: Gamification Engine family opt-out</i> 9.23 <i>Use case: Geolocation opt-in</i> 9.24 <i>Use case: Geolocation opt-out</i> 9.25 <i>Use case: Defining family composition</i> 9.26 <i>Use case: Defining water consumption distribution rule among family members</i> 9.27 <i>Use case: Learning interactively about innovative pricing models</i>
Provided Interfaces	<i>This component is a web application, which includes UIs and REST APIs.</i> <i>The APIs are:</i> - <i>GET: GetActions</i> - <i>GET: GetUserCredits</i> - <i>GET: GetUserRewards</i> - <i>POST: AssignActionsToUsers</i> - <i>POST: RedeemUserReward</i> - <i>POST: UserRegistration/userRegistration</i> - <i>POST: UserUpdate/userUpdate</i>

Dependencies / Required Interfaces	The component requires the implementation of the following interfaces: - [GET]user(userid): returns all the metadata for the specified user - [GET]exists(userEmail): returns true if a user with the specified email exists, false otherwise - [GET]billByUser(userId): returns a short recap of the bill for the logged user - [GET]waterConsumptionStatisticsByUser(userId) returns the statistics of water consumption for the logged user - [GET]tips(userId): returns a list of tips for a specified user - [GET]waterUsageAlerts(userId): returns the list of usage alerts for the logged user - [GET]mediaContent(userID): videos, tutorial etc: may depend on the users - [GET]neighbors(userID): returns neighbors location (house holders onl, same zip code) - [POST]signup: pushes a registration to the SmartH2O DB - [POST]userStatus (userId, online offline): notify to the SmartH2O DB the change of status of a user - [POST]unsubscribe: remove a registration to the SmartH2O DB - [POST] selfOptIn - [POST] selfOptOut - [POST] familyOptIn - [POST] familyOptOut - [POST] geoOptIn - [POST] geoOptOut - [POST]addUserProfile: saves user profile information like: family members, house configuration, appliances etc.. - [POST]updateUserProfile: updates user profile information like: family members, house configuration, appliances etc.. - [POST]consumptionGoal(goal, userId): saves a consumption goal defined by the user - [POST]addConsumptionAction(userId, action): pushes a consumption action (e.g "watering the garden for 15 minutes") to the SmartH2O DB
---	--

3.4.1 Main Use Cases

Use Case 2.1: Gamification Engine Signup

Use Case 2.1	Gamification Engine Signup	
Goal in Context	The consumer performs a registration to the platform, his data are saved into the SmartH2O DB.	
Precondition	The consumer is logged with his customer account.	
Success Condition	End	The consumer becomes a Competitor customer and can exploit the gamification extension of the customer portal.
Failed Condition	End	The consumer does not become a Competitor user, and cannot exploit the gamification extension of the customer portal.

Primary actors	<i>Consumer, GamificationEngine</i>
Secondary Actors	<i>ESB, Smarth2ODB</i>
Steps	- the user requests the registration UI - the user fill and send the registration form - GamificationEngine adds the user to the competitor list - GamificationEngine forward the registration to the Smarth2ODB
Note	<i>Fields in the registration form: photo (see Figure 4)</i>

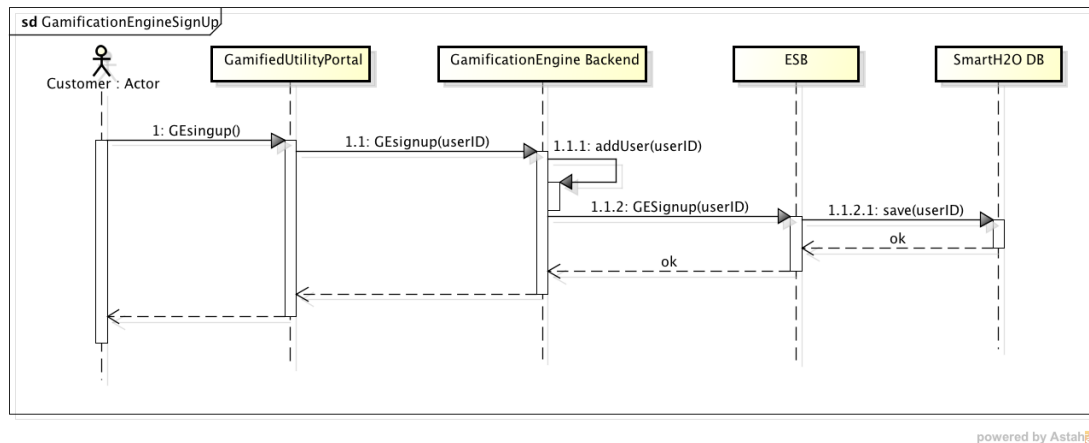


Figure 15: Sequence Diagram Use Case 2.1.

Use Case 2.2: Visual Exploration Gamified Utility Data

Use Case 2.2	Visual Exploration Gamified Utility Data	
Goal in Context	<i>The competitor explores his water consumption visualizing his actions, his bills, his badges, the leaderboard etc.</i>	
Precondition	<i>The competitor is logged to the platform</i>	
Success Condition	End	<i>The competitor is explores his gamified data</i>
Failed Condition	End	--
Primary actors	<i>Competitor, GamificationEngine</i>	
Secondary Actors	<i>ESB, Smarth2ODB</i>	

Steps	- The competitor requests to visualize his gamified water consumption page - The GE Backend collects inputs from ESB (bill, alerts, consumption) and from its internal DB (points, actions, leaderboard) - The GE Backend builds the page
Note	- This use case is specified for a competitor consumer, if the user is a simple consumer then the visualization will not include points and leaderboards.

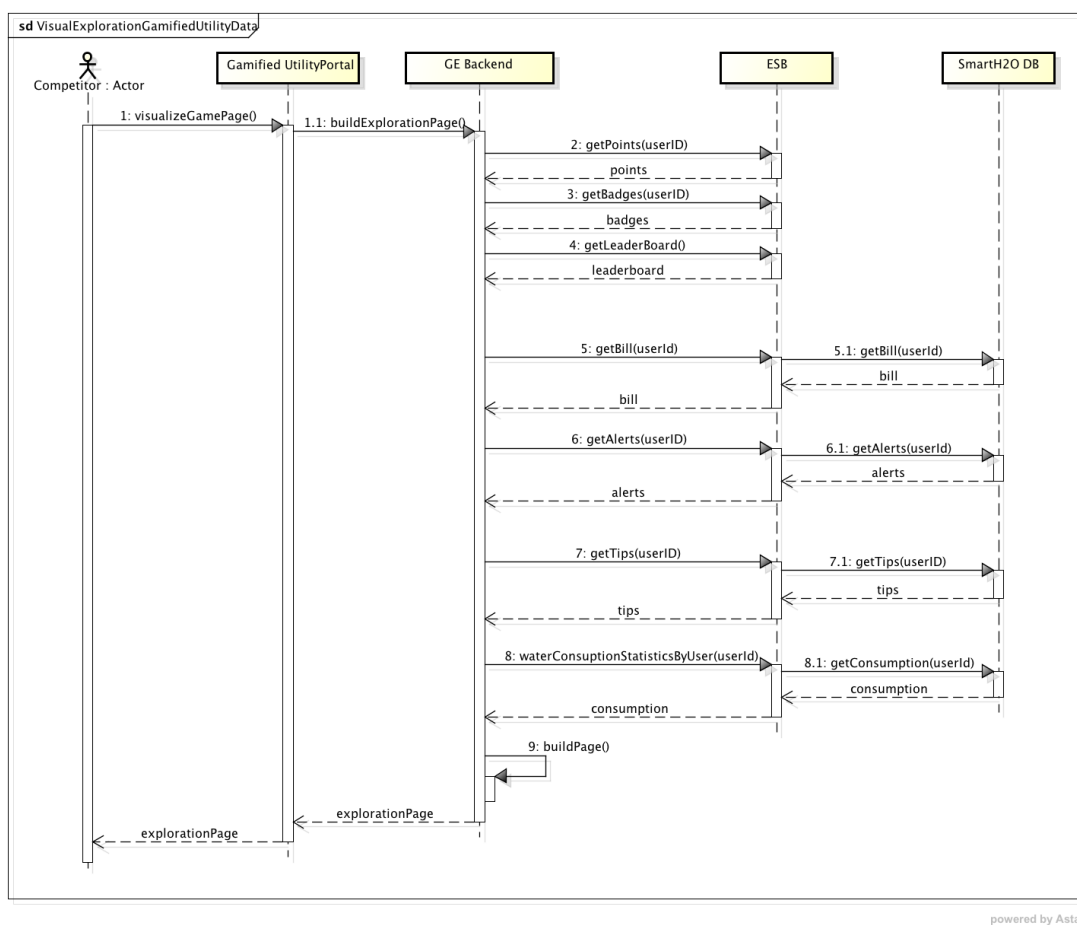
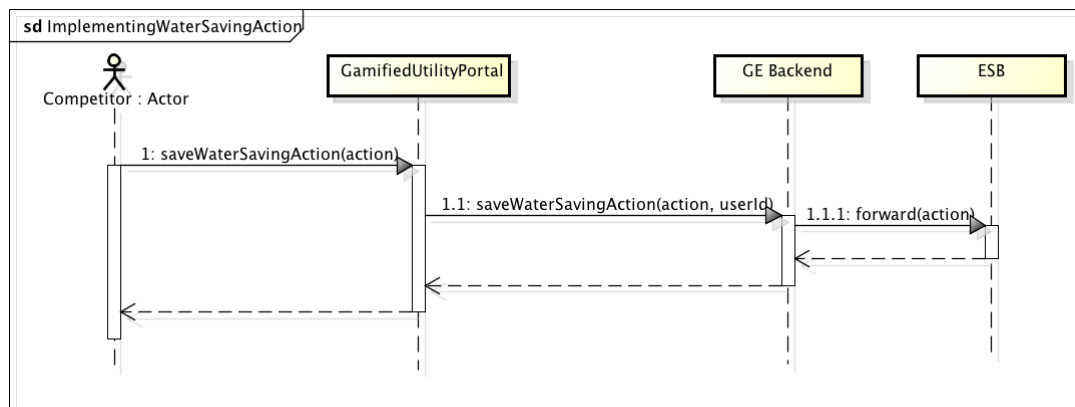


Figure 16: Sequence diagram Use Case 2.2.

Use Case 2.3: Implementing Water Saving Action

Use Case 2.3	Implementing Water Saving Action	
Goal in Context	The competitor submits a water saving action to the platform	
Precondition	The competitor is logged to the platform	
Success Condition	End	The water saving action is correctly saved into the SmartH2O DB

Failed Condition	End	--
Primary actors	<i>Competitor, GamificationEngine</i>	
Secondary Actors	<i>ESB, Smarth2ODB</i>	
Steps	- The competitor submits a water saving action to the Gamification Engine - The Gamification Engine add the points to the total points of the user - The Gamification Engine notifies the ESB that the competitor has submitted a water saving action	
Note	- The Gamification Engine will forward to the ESB only water saving actions, all the other actions (e.g. add a comment, read a tip) will be saved only locally to the GE DB.	



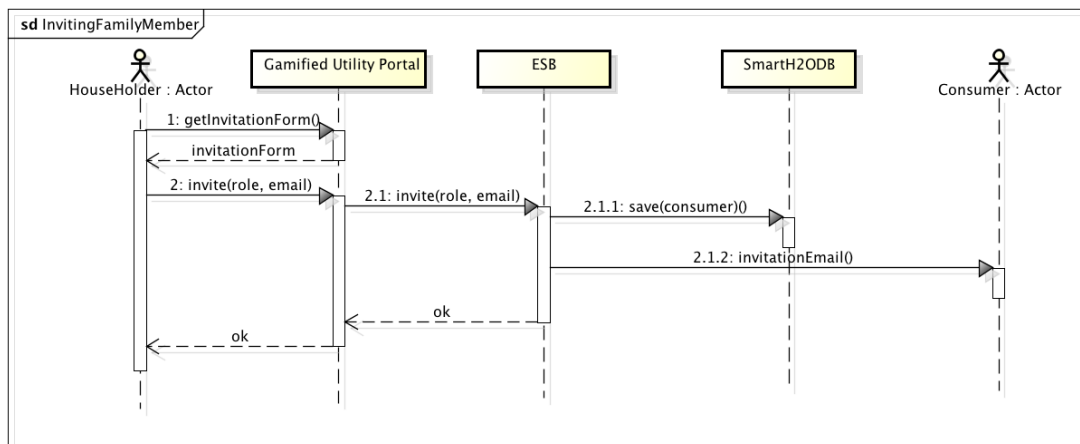
powered by Astah

Figure 17: Sequence Diagram Use Case 2.3.

Use Case 2.4: Inviting Family Member

Use Case 2.4	<i>Inviting Family Member</i>	
Goal in Context	<i>The HouseHolder invites another member of the family to join the platform</i>	
Precondition	<i>The HouseHolder is logged to the platform</i>	
Success Condition	End	<i>The platform sends an invitation to the family member</i>
Failed Condition	End	<i>The specified email does not exists, the HouseHolder is notified</i>
Primary actors	<i>HouseHolder, GamificationEngine</i>	
Secondary Actors	<i>ESB, Smarth2ODB</i>	

Steps	- The HouseHolder requests the invitation page - The HouseHolder fills the registration page specifying the other member role and email - The GamificationEngine sends a notification to the ESB with the new role and email - The ESB saves the new member in the Smarth2ODB and sends an invitation via email to the new member
Note	



powered by Astah

Figure 18: Sequence diagram Use Case 2.4.

Use Case 2.5: Setting action types

Use Case 2.5	<i>Setting action types</i>	
Goal in Context	<i>The GE Content and Rules Editor configures a new type of action</i>	
Precondition	<i>The GE Content and Rules Editor is logged to the platform, a Gamified application exists.</i>	
Success Condition	End	<i>A new action type or reward type is inserted into the system</i>
Failed Condition	End	<i>No action type or reward type is inserted into the system</i>
Primary actors	<i>GE Content and Rules Editor, GamificationEngine</i>	
Secondary Actors		
Steps	- The GE Content and Rules Editor submits a configuration for a new action (including: name, score, participation, reputation, repeatable, timeElapsed, active) - The Gamification Engine saves the action	

Note

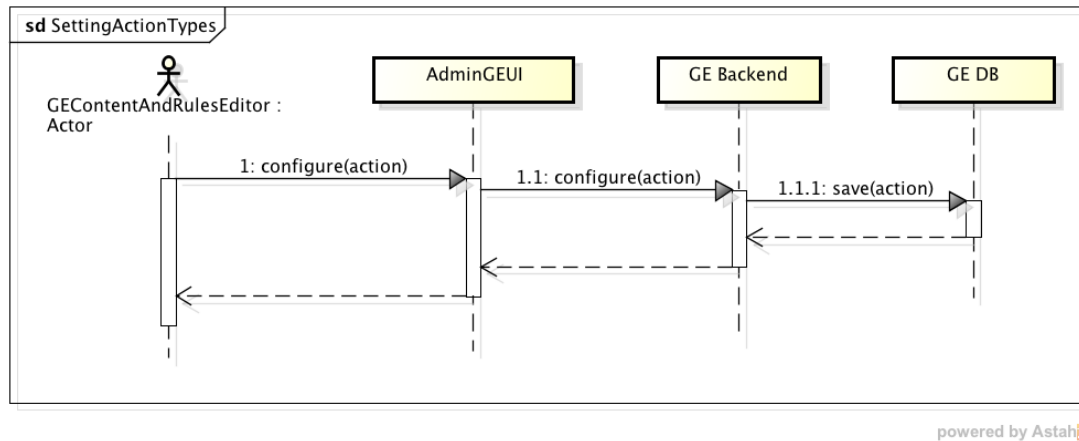


Figure 19: Sequence Diagram Use Case 2.5.

3.4.2 Component Interfaces Specification

Retrieve the list of available Actions

In order to get available actions for a specific gamified application the Get Action web service is available. The service requires the name of the gamified application as a mandatory parameter.

The endpoint is:

{webappUrl}/UserActivityCreditWebServiceREST/GetActions/getActions.do

A sample of JSON response is:

```
{
  "actions": [
    {
      "gamifiedApplication": "Energy Portal",
      "actionName": "Do energy saver quiz",
      "actionID": 4
    },
    {
      "gamifiedApplication": "Energy Portal",
      "actionName": "Login",
      "actionID": 1
    }
  ]
}
```

Get the user credits

In order to **get the user credits** obtained by interacting with the game, it is required to specify the user email as parameter of the REST web service.

Endpoint

```
{webappUrl}/UserActivityCreditWebServiceREST/GetUserCredits/getUserCredits.do?userEmail= xxx@yyy.com
```

Response

```
{
  "userCredits": {
    "userEmail": "xxx @yyy.com",
    "totalCredit": 3400,
    "creditsSpent": 0,
    "creditsAvailable": 3400
  }
}
```

Get the rewards

To **get the rewards** that can be redeemed by the user, it is also required to specify the user email as a parameter of the web service.

Endpoint

```
{webappUrl}/UserActivityCreditWebServiceREST/GetUserRewards/
getUserRewards.do?userEmail= xxx@yyy.com
```

Response

```
{
  "rewards": [
    {
      "rewardName": "CouponDiscount",
      "rewardID": 1,
      "neededPoints": 1000,
      "userEmail": "xxx@yyy.com"
    }
  ]
}
```

Post a new rewards

To **post a new reward** that can be redeemed by the user, it is also required to specify the user email as a parameter of the web service.

In order to register a new reward in the gamification platform, the **Post a new reward** web service is available. The request is a JSON array with the following parameters:

- **RewardName** [MANDATORY]
- **neededPoints** [MANDATORY]

Endpoint

```
{webappUrl}/UserActivityCreditWebServiceREST/PostUserRewards/postUserRewards.do
```

Assign Actions To User

In order to register the user action in the gamification platform, the **Assign Actions To User** web service is available. The request is a JSON array with the following parameters:

- **email**: the email of the user to assign the action [MANDATORY]
- **time**: the timestamp of the request in Unix Timestamp format [MANDATORY]
- **area**: the name of the gamified application [MANDATORY]
- **name**: the name of the action [MANDATORY]
- **description**: the description of the action [MANDATORY]
- **tag**: additional parameter for managing non-repeatable action [NOT MANDATORY]
- **link**: additional parameter for managing non-repeatable action [NOT MANDATORY]
- **executor**: additional parameter for managing non-repeatable action [NOT MANDATORY]
- **objectkey**: additional parameter for managing non-repeatable action [NOT MANDATORY]

Input

```
[
  {
    "email": "xxx@yyy.com",
    "time": 1407307785347,
    "area": "Energy Portal",
    "name": "Login",
    "description": "Login",
    "tag": " ",
    "link": " ",
    "executor": " "
  }
]
```

Endpoint

{webappUrl}/UserActivityCreditWebServiceREST/AssignActionsToUsers/assignActionsToUsers.do

Redeem User Reward

In order to register the user reward in the gamification platform the **Redeem User Reward** we service is available. The request is a JSON array with the following parameters:

- **idReward**: the id of the reward to redeem [MANDATORY]
- **userEmail**: the email of the user that redeems the reward [MANDATORY]

Input

```
{"idReward":1,"userEmail":" xxx@yyy.com "}
```

Endpoint

{webappUrl}/UserActivityCreditWebServiceREST/RedeemUserReward/redeemUserReward.do

User Registration

To push user registration data about a new user in the Gamification Engine the **User Registration** web service will be used.

The request is a JSON array with the following parameters:

- **birthdate**: the birthdate of the user in UNIX timestamp format
- **username**
- **password**
- **email**
- **firstname**
- **lastname**
- **city**
- **country**
- **publicprofile**: boolean value to indicate if the user is active or not in the community
- **internal**: boolean value to indicate if the user is an internal user of the community
- **isocode**: language isocode (used to manage international community)
- **geoarea**
- **photname**: the name of the photo of the user
- **photocode**: the photo of the user in Base64 format

Input

```
[{"birthdate":1407276000000,"username":"markross","password":"markross","email":"mark.ross@e     e.com","firstname":"Mark","lastname":"Ross","city":"London","country":"United Kingdom","publicprofile":true,"internal":false,"isocode":"en","geoarea":"Europe"}]
```

Endpoint

```
{webappUrl}/UserRegistrationWebServiceREST/UserRegistration/userRegistration.do
```

User update

To push the update user data to the gamification platform the **User update web service will be used.**

The JSON array for the request is composed by the following parameters:

- **birthdate**: the birthdate of the user in UNIX timestamp format
- **username**
- **password**
- **email [MANDATORY]**
- **firstname**
- **lastname**
- **city**
- **country**
- **publicprofile**: boolean value to indicate if the user is active or not in the community
- **internal**: boolean value to indicate if the user is an internal user of the community
- **isocode**: language isocode (used to manage international community)
- **geoarea**
- **photname**: the name of the photo of the user
- **photocode**: the photo of the user in Base64 format

Input

```
[{"birthdate":1407276000000,"username":"markross","password":"markross","email":"mark.ross@e     e.com","firstname":"Mark","lastname":"Ross","city":"London","country":"United Kingdom","publicprofile":true,"internal":false,"isocode":"en","geoarea":"Europe"}]
```

Endpoint

```
{webappUrl}/UserRegistrationWebServiceREST/UserUpdate/userUpdate.do
```

3.5 Games Platform

As explained in *D4.1, First social game and implicit user information techniques*, the introduction of a platform for handling Games in the SmarH2O project derives from the need of being able to attract the interests of users which are not strictly linked with water utilities while being able to run even as a standalone component. Different game design approaches and instantiations are being investigated; nonetheless, they should all rely on an infrastructure that provides features and data collection common to all of them.

This is the role of the **Games Platform**, a central component illustrated in Figure 20, which handles the communication with the main SmarH2O platform and takes care of managing users' registration, profiling them, orchestrating and keeping track of game instances and the gameplay sessions of the players. The points and the achievement collected in the digital games should be transferred towards the Gamification Engine, to create a tied link between the two approaches for engaging the users. For these reasons, a dedicated database takes care of storing the users and their details, gameplay session's data, point and achievements obtained within the games platform.

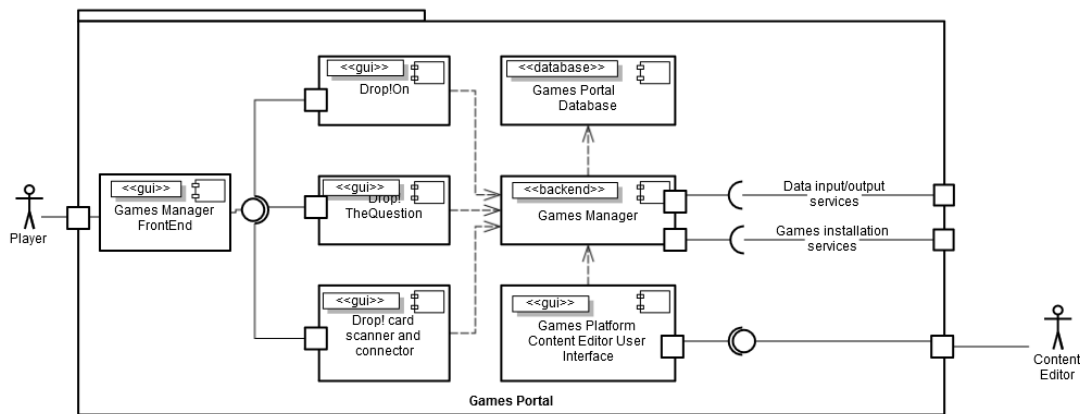


Figure 20: The Game Platform component diagram.

At the time of writing, the component should support two different game instances: a Quiz Game, called **Drop!TheQuestion**, that is the digital extension of the **Drop!** board game and is used to collect behavioural data from the users; a single player game, called **DropOn!** that is used to promote sustainable behaviours and as an educational game. Both of them are described in D4.1. For the definition of new content for the proposed games (e.g. badges and achievements for the single player mode), the games platform offers also a Content editor UI that allows non-technical users to enrich the content offered by the applications without the need of modifying the underlying architecture or create new builds.

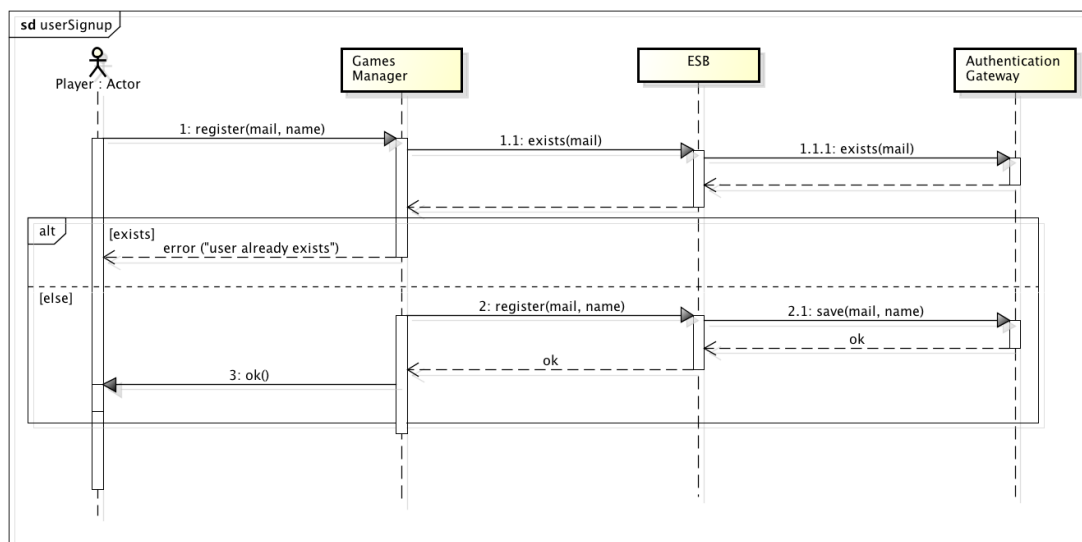
Component Name	Games Platform
Leader	MOONSUB
Partners	POLIMI
Goal	<i>The component manages the integration with all the games and it talks with the GamificationEngine to save actions and points</i>
User use cases	11.1 Use case: Games Platform signup 11.2 Use case: Playing a standard mobile game 11.3 Use case: Playing the card game and its digital game extension 11.4 Use case: Gaining power-ups based on the Gamification Engine credits 11.5 Use case: Connecting player profile to the Gamification Engine 11.6 Use case: Setting content of game questions 11.7 Use case: Setting content of questions for a given utility game
Provided Interfaces	<i>This component is a mobile application written using the framework Unity.</i>
Dependencies / Required Interfaces	<i>The component requires the implementation of the following interfaces:</i> - [GET]: GetUserRewards - [POST]signup: pushes a registration to the SmartH2O DB - [POST]: RedeemUserReward - [POST]userStatus (userID, online offline): notify to the SmartH2O DB the change of status of a user - [POST]unsubscribe: removes a registration from the SmartH2O DB - [POST]addUserProfile: saves user profile information like: family members, house configuration, appliances etc.. - [POST]updateUserProfile: updates user profile information like: family members, house configuration, appliances etc.. - [POST]connectGEProfile: connect the gamer profile to the associated GE profile - [POST]AssignActionsToUsers(userID, action): forwards an action to the GamificationEngine

3.5.1 Main Use Cases

Use Case 3.1: Application Registration

Use Case 3.1	Application Registration
Goal in Context	<i>The player registers the installed application into the Games Portal</i>
Precondition	<i>The player installed the application on a device</i>
Success Condition	End <i>The player is correctly registered into the Games Portal</i>

Failed Condition	End	<i>The email already exists, the player is notified</i>
Primary actors	<i>Player, GamesManager</i>	
Secondary Actors	<i>ESB, Authentication Gateway</i>	
Steps	- The player asks to register to the platform specifying email and name - The Games Manager verifies if the email has been used on another registration invoking the Authentication Gateway - If the email does not exists then the Games Manager invoke the Authentication Gateway to save the new user - Otherwise it notifies the error to the player	
Note		



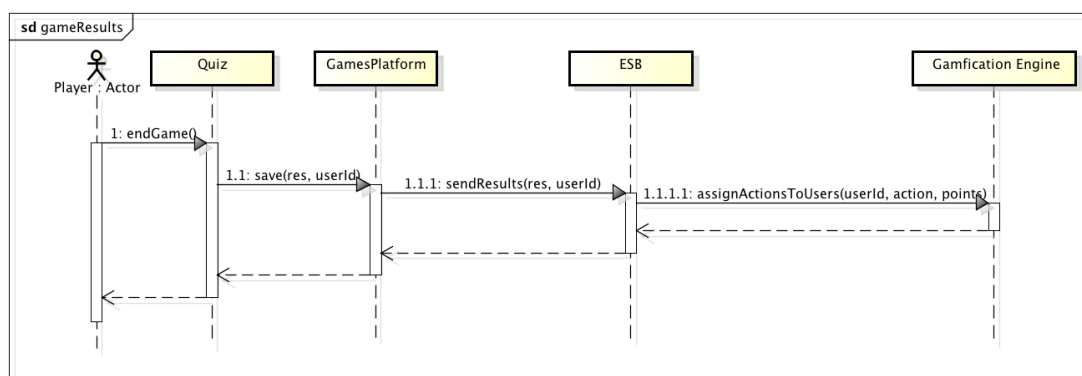
powered by Astah

Figure 21: Sequence Diagram Use Case 3.1.

Use Case 3.2: Game Results

Use Case 3.2	Game Results	
Goal in Context	<i>The player ends a game and the platform forward the results to the GamificationEngine</i>	
Precondition	<i>The player is logged to the platform, the player played one of the games in the platform</i>	
Success Condition	End	<i>The gamer gains points on the Gamification Engine</i>
Failed Condition	End	--

Primary actors	<i>Player, GamesPlatform</i>
Secondary Actors	<i>ESB, GamificationEngine</i>
Steps	- <i>The gamer ends a game</i> - <i>The game calculates the results and send it to the ESB</i> - <i>The ESB forwards the result to the GamificationEngine</i>
Note	



powered by Astah

Figure 22: Sequence Diagram Use Case 3.2

Use Case 3.3: Submit a new Question

Use Case 3.3	Submit a new Question
Goal in Context	<i>Add a new Question to the set of questions used in the Drop quiz game</i>
Precondition	--
Success Condition	End <i>The new question is saved and available for next games</i>
Failed Condition	End --
Primary actors	<i>Games Platform CE, GamesPlatform</i>
Secondary Actors	--
Steps	- <i>The Games Platform CE accesses the Games Platform CE UI and submits a new Question and a set of Answers (including the correct one)</i> - <i>The Games Platform saves the question into the DB</i>
Note	<i>The list of fields for Question and Answer are specified in 2.3</i>

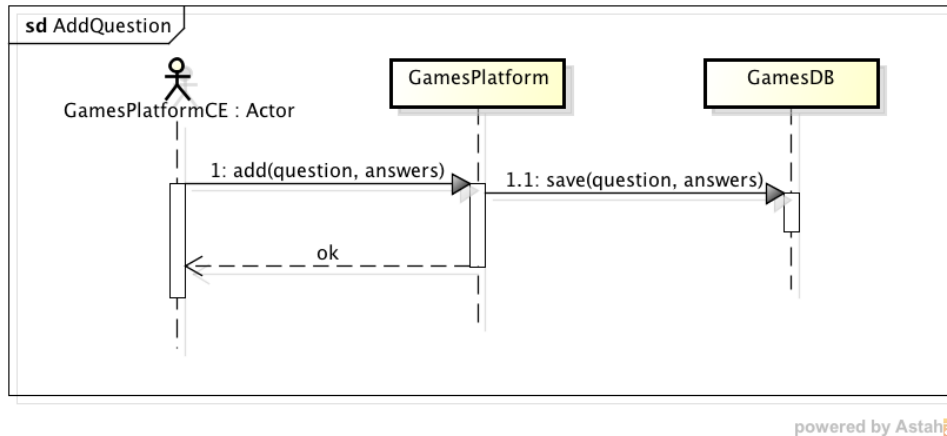


Figure 23: Sequence Diagram Use Case 3.3

3.6 Enterprise Service Bus (ESB)

This component is a service oriented middleware layer that supports the loose coupling of the Smarth2O components; it permits the publication of their interfaces (Application Programming Interfaces, APIs) and the synchronous and asynchronous communication among components. The goal of the ESB is to decouple the heterogeneous components of the Smarth2O platform as much as possible, to support the extension with new services and functions and the rapid adaptation of the platform to new contexts (e.g., other utility companies with different IT standards and information systems).

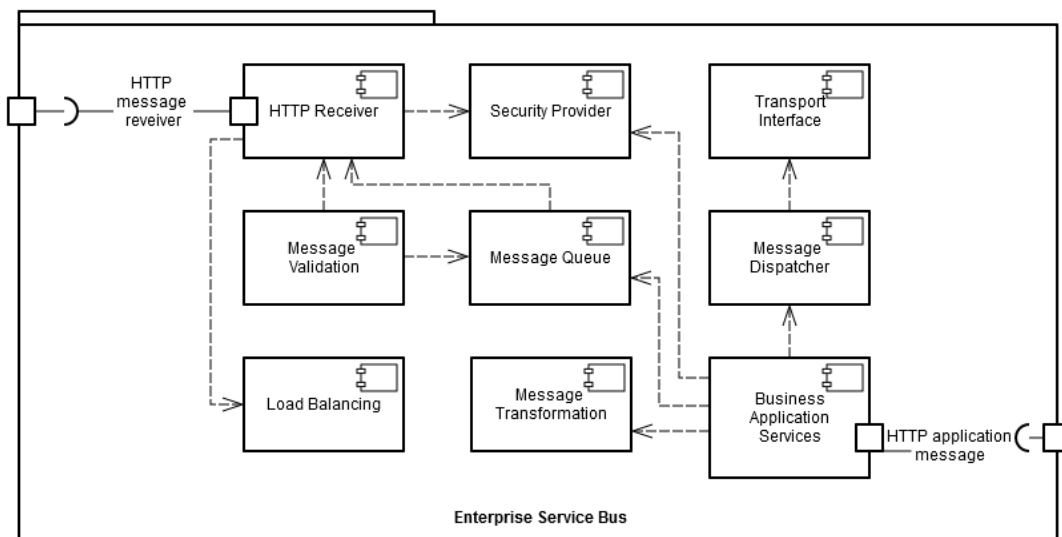


Figure 22: Enterprise Service Bus component diagram.

Component Name	ESB
Leader	SETMOB
Partners	POLIMI, SUPSI
Goal	Provide loose coupling of the SmartH2O Components
User use cases	The use cases of ESB are the summary of Use Cases that involves component interaction. See summary table below.
Provided Interfaces	Interfaces provided by ESB are the summary of SmartH2O required interfaces that need to interact with other components. See summary table below.
Dependencies / Required Interfaces	Interfaces required by ESB are the summary of SmartH2O provided interfaces that service requests from other components. See summary table below.

Table 1 summarizes Use Cases where the ESB component is involved. Provided interfaces and interfaces to be implemented (Required interfaces) are detailed for each use case.

This summary will allow identification of common requirements among components and will be the basis in defining the minimum required set of interfaces to be provided and implemented interfaces.

Table 1: Summary of Use Cases pertaining to ESB.

Use Case	Client component	Provided interfaces	Service component	Required interfaces
userSignup	UtilityPortal	exists(mail)	Authentication Gateway	exists(mail)
	UtilityPortal	signup(mail, name, address, UtilityCode)	Authentication Gateway	save(mail, name, address, UtilityCode)
	UtilityPortal	verify(UtilityCode)		
visualExplorationUtilityData	UtilityPortal	getBill(userId)	Pricing Engine	getBill(userId)
	UtilityPortal	getAlerts(userId)	SmartH2ODB	getAlerts(userId)
	UtilityPortal	getTips(userId)	SmartH2ODB	getTips(userId)
	UtilityPortal	waterConsumptionStatisticsByUserId(userId)	SmartH2ODB	getConsumption(userId)
	Utility Portal	getNeighborhood(userId)	SmartH2ODB	getNeighborhood(userId)
Gamification Engine SignUp	GE BackEnd	GESignUp(userId)	SmartH2ODB	save(userId)
Visual Exploration Gamified Utility Data	GE BackEnd	getPoints(userId)		
	GE BackEnd	getBadges(userId)		

	GE BackEnd	getLeaderBoard(userId)		
	GE BackEnd	getBill(userId)	Pricing Engine	getBill(userId)
	GE BackEnd	getAlerts(userId)	SmarrH2ODB	getAlerts(userId)
	GE BackEnd	getTips(userId)	SmarrH2ODB	getTips(userId)
	GE BackEnd	waterConsumptionStatisticsByUserId(userId)	SmarrH2ODB	getConsumption(userId)
Implementing Water Saving Action	GE BackEnd	forward(userId)		
Inviting Family Member	UtilityPortal	invite(role, email)	SmarrH2ODB	save(consumer)
			Consumer	InvitationEmail()
User Registration	Games Manager	exists(email)	Authentication Gateway	exists(email)
	Games Manager	register(email, name)	Authentication Gateway	save(email, name)
Game Results	Games Platform	sendResults(res, userId)	Gamification Engine	assignActionsToUsers(userId, actions, points)
Transfer of incoming user authentication	Portal Exchange Manager	authenticateUser(userId)	Authentication Gateway	authenticateUser(userId)
Transfer of outgoing authentication	Authentication Gateway	authenticateOutgoingUser(userId)	PortalExchangeManager	authenticateUser(userId)
Transfer of user profile – incoming			PortalExchangeManager	getUserProfile(userId)
Transfer of user profile – outgoing	PortalExchangeManager	getUserProfile	SmarrH2ODB	getUserProfile(userId)
Visualizing aggregate household consumption information by geolocation	AdminWaterUtility	getWaterConsumption(geolocation)	SmarrH2ODB	getWaterConsumption(geolocation)
Identifying customer segments	AdminWaterUtility	saveSegment(attributes)	SmarrH2ODB	saveSegment(attributes)
Setting actions and	AdminWaterUtility	setActionRewardForSegment	SmarrH2O	getUsersForSegment

rewards
types for
specific user
segments/groups

			Smarth2O	setActionsRewardsForUsers
Predicting customer segment consumption behavior on past information	AdminWaterUtility	getConsumptionSimulationPastInfo	ABM	getConsumptionSimulationPastInfo
Predicting customer segment consumption behavior based on incentive response	AdminWaterUtility	getConsumptionSimulationIncentive	ABM	getConsumptionSimulationIncentive
Predicting customer response to pricing scheme	AdminWaterUtility	getConsumptionSimulationPricing	ABM	getConsumptionSimulationPricing

3.6.1 Component Interfaces Specification

--

3.7 Smart Meter Data Manager

The Smart Meter Data Management component (from now on, SMDM) has the role to acquire, process and consolidate water consumption data automatically provided by smart meters. The sources of the water consumption metering data are the hardware systems and the software platforms of the water utility. The data is provided by FTP/SFTP as bulk files (CSV, XML) containing periodical measurements on hourly, daily, weekly, monthly, etc. bases.

The increased frequency of the water counter acquisition and processing is a key factor in delivering accurate data for assisting on correct decision making. As a consequence, the SMDM component has to provide a scalable method for processing and consolidating increasing amounts of consumption data flowing from an increasing number of water counters. In this sense, the critical business requirement for the SMDM component is to implement a scalable architecture beyond traditional RDBMSs. This is accomplished by implementing specific BigData processing techniques such as parallel processing, incremental MapReduce computations, virtualization of the storage for using cloud computation in order to achieve data consolidation while obtaining a reduced response time.

Component Name	Smart Meter Data Manager
Leader	SETMOB
Partners	--
Goal	Collecting consumption data via smart meters
User use cases	8.2 Collecting consumption data
Provided Interfaces	<i>This component is a batch process performing an extract, transform, load (ETL) suite of jobs for pulling the water consumption data from data files and consolidating in the Smarth2O platform database.</i>
Dependencies / Required Interfaces	<i>The component requires the implementation of the following jobs:</i> - <i>FileValidation: checks if the file fulfils the label and size related criteria</i> - <i>StructureValidation: checks if the data file has the structure previously agreed with the water utility</i> - <i>DataProcessing: launching the extraction process, preparing the record set for persistence, loading the record set to the database</i>

3.7.1 Main Use Cases

Use Case 5.2: Collecting consumption data

Use Case 5.2	Collecting consumption data
Goal in Context	<i>Collecting consumption data via smart meters</i>
Precondition	<i>Water consumption of customer household is metered (smart meters) for the reference interval of time</i>
Success Condition	End <i>The system stores in the Smarth2O platform database the consumption data for a reference interval of time</i>
Failed Condition	End <i>The system is not able to store in the Smarth2O platform database the consumption data for a reference interval of time despite the process has been correctly initiated by the water utility</i>
Primary actors	Smart Meter Data Manager
Secondary Actors	<i>SmartMetered user, smart meter</i>

Steps	- The smart water meter measures the customer's water consumption. The consumption data is collected by the water utility automatically (smart meter). The water utility creates a file with water consumption data in a pre-agreed format - The water consumption data is transmitted to the Smarth2O platform where it is received, validated, processed and stored - The result of processing the collected consumption data by the Smarth2O platform is saved in a Log file that can be visualised online by the water utility or by the platform administrators
Note	<i>This use case is triggered by the water utility uploading to the Smarth2O server via SFTP the file containing the consumption data for a reference interval of time</i>

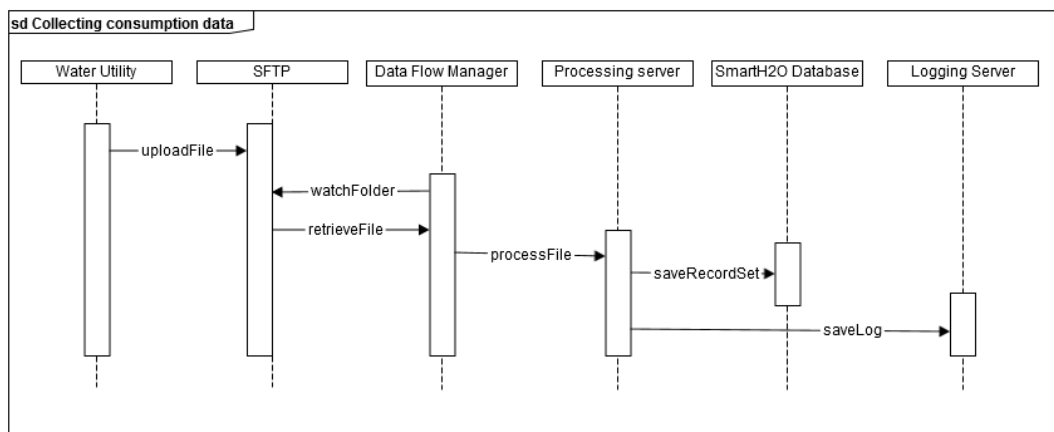


Figure 24: Sequence Diagram Use Case 5.2.

3.8 Portal Data Exchange Manager

This component deals with the data exchange communication that occurs “behind the scenes” between the Smarth2O platform and the third party applications already supporting the interaction with the various types of users.

There are several use cases of data exchange implemented by this component:

- Transfer of user authentication. Single Sign-On, both for Consumer Users and Admin Users. Partner portals must be first registered for trust relationship.
- Transfer of user profile. This flow transfers user profile from a source portal to a destination portal. As an internal rule, each portal might transfer user profile only if user previously agreed on such a transfer

To ensure a smooth and complete transfer of user profile, it is recommended that each Water utility portal implement a standard of user profile. Else, a mapping of user attributes between different portals must be implemented by this component.

Component Name	<i>Portal Data Exchange Manager</i>
-----------------------	--

Leader	SETMOB
Partners	--
Goal	Manages transfer of information between Smarth2O and similar portals of Water Utility
User use cases	- 6.1 Transfer of incoming user authentication - 6.2 Transfer of outgoing user authentication - 6.3 Transfer of user profile incoming - 6.4 Transfer of user profile outgoing
Provided Interfaces	- [GET] authenticateIncomingUser(portalId, partnerPortalKey, userId). Authenticate incoming user against Smarth2O platform and provide authentication token to partner portal - [GET] getUserProfile(portalId,partnerPortalKey,authenticationKey, userId) Returns Smarth2O user profile
Dependencies / Required Interfaces	We assume that the Partner portal will implement following interface: - [GET] authenticateIncomingUser(portalId,smarth2OKey, userId). Expected to retrieve authentication token from partner portal - [GET] getUserProfile(portalId,smarth2OKey, authenticationKey,userId). Returns Partner portal user profile

3.8.1 Main Use Cases

Use Case 6.1: Transfer of incoming user authentication

Use Case 6.1	Transfer of incoming user authentication	
Goal in Context	Provide Single Sign-On service for incoming users from registered portals	
Precondition	Partner portal was previously registered in Smarth2O	
Success Condition	End	If existing and active user in Smarth2O, incoming user is authenticated If user not exist in Smarth2O, incoming user is not authenticated
Failed Condition	End	Existing active user is not authenticated
Primary actors	PartnerPortal, Portal Exchange Manager	
Secondary Actors	ESB, AuthenticationGateway	
Steps	- Partner portal send an authentication request for its user - Partner Portal Exchange Manager check user against AuthenticationGateway - If User exists and is active, authentication key for user is returned. - If User does exist or is not active, an notice message is returned	

Note

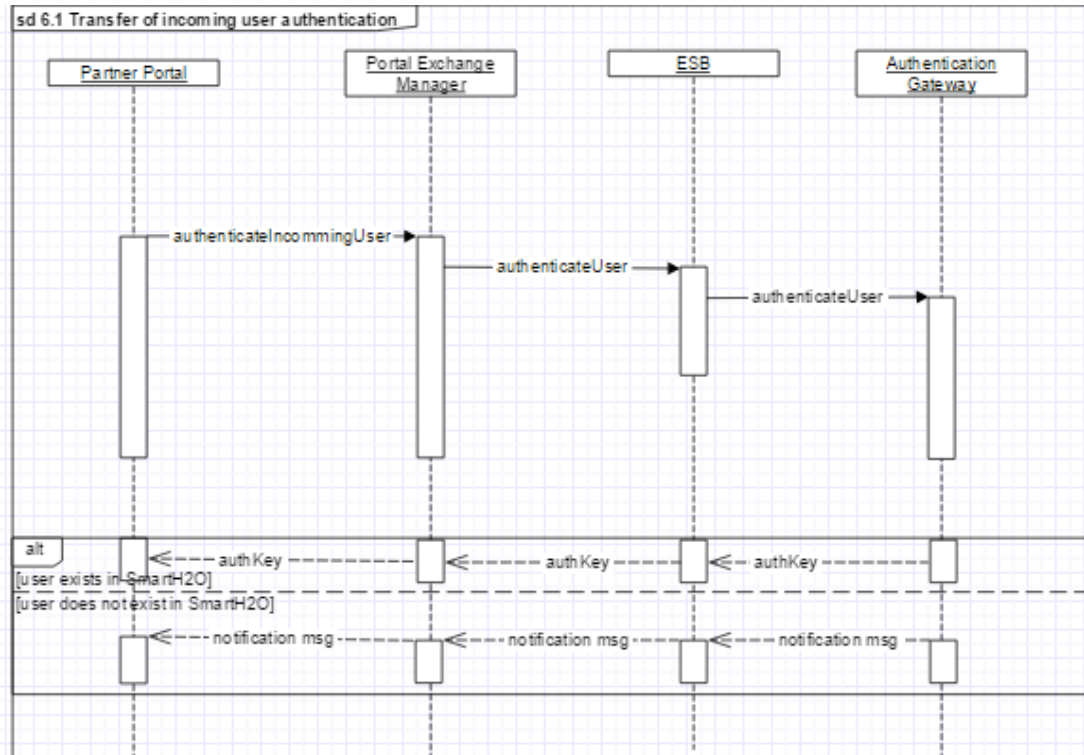


Figure 25: Sequence Diagram Use Case 6.1.

Use Case 6.2: Transfer of outgoing user authentication

Use Case 6.2		Transfer of outgoing user authentication
Goal in Context		Provide Single Sign-On service for outgoing users of SmartH2O
Precondition		Partner portal was previously registered in SmartH2O
Success Condition	End	If existing and active user in PartnerPortal, outgoing user is authenticated If user does not exist in PartnerPortal, outgoing user is not authenticated
Failed Condition	End	Existing active user is not authenticated in PartnerPortal
Primary actors		Portal Exchange Manager
Secondary Actors		ESB, AuthenticationGateway

Steps	- <i>AuthenticationGateway sends an authentication request a user against a PartnerPortal</i> - <i>Partner Portal Exchange Manager verifies user against PartnerPortal</i> - <i>If User exists and is active in PartnerPortal, authentication key for user is returned.</i> - <i>If User does exist or is not active in PartnerPortal, an notice message is returned</i>
Note	

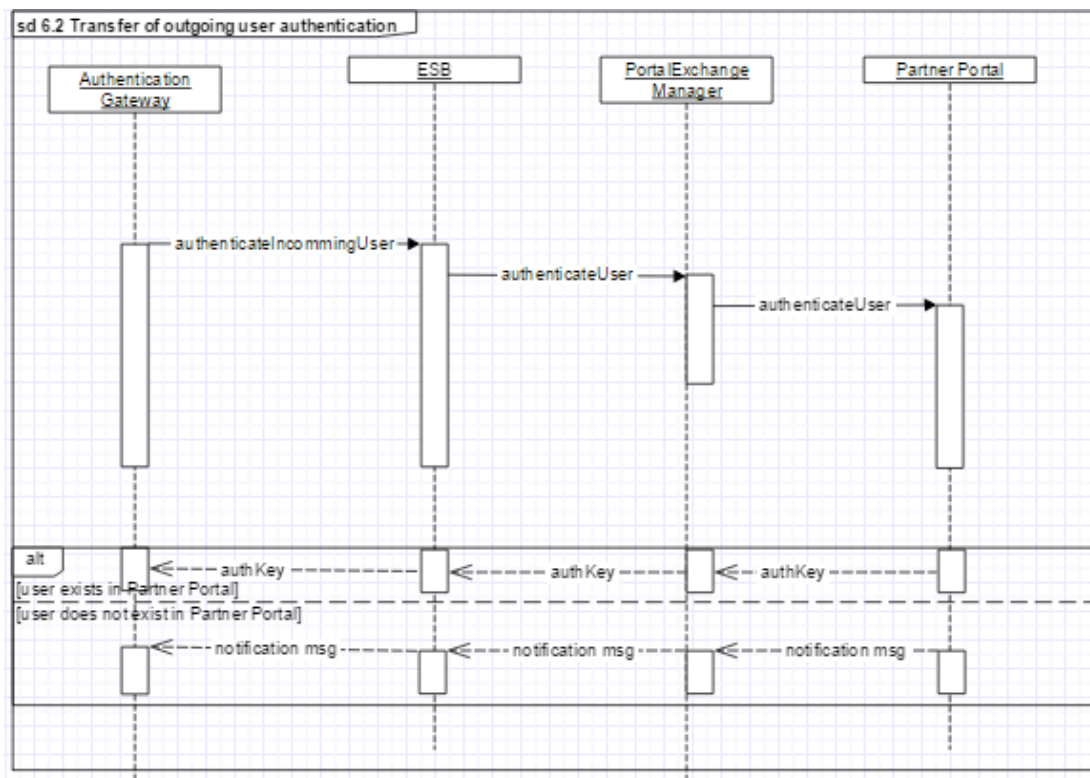


Figure 26: Sequence Diagram Use Case 6.2.

Use Case 6.3: Transfer of user profile – incoming

Use Case 6.3	<i>Transfer of user profile - incoming</i>	
Goal in Context	<i>Transfer of user profile from partner portal to Smarth2O DB</i>	
Precondition	<i>Partner portal was previously registered in Smarth2O.</i> <i>Mapping of user profiles was previously configured in Smarth2O</i>	
Success Condition	End	<i>Incoming user profile is saved to database</i>
Failed Condition	End	<i>Incoming user profile is not saved to database</i>

Primary actors	<i>PartnerPortal, Portal Exchange Manager</i>
Secondary Actors	<i>ESB, Smarth2ODB</i>
Steps	- <i>Partner Exchange Manager sends request for user profile to partner portal</i> - <i>Partner Portal sends user profile</i> - <i>Partner Exchange Manager saves user profile in Smarth2O database</i>
Note	

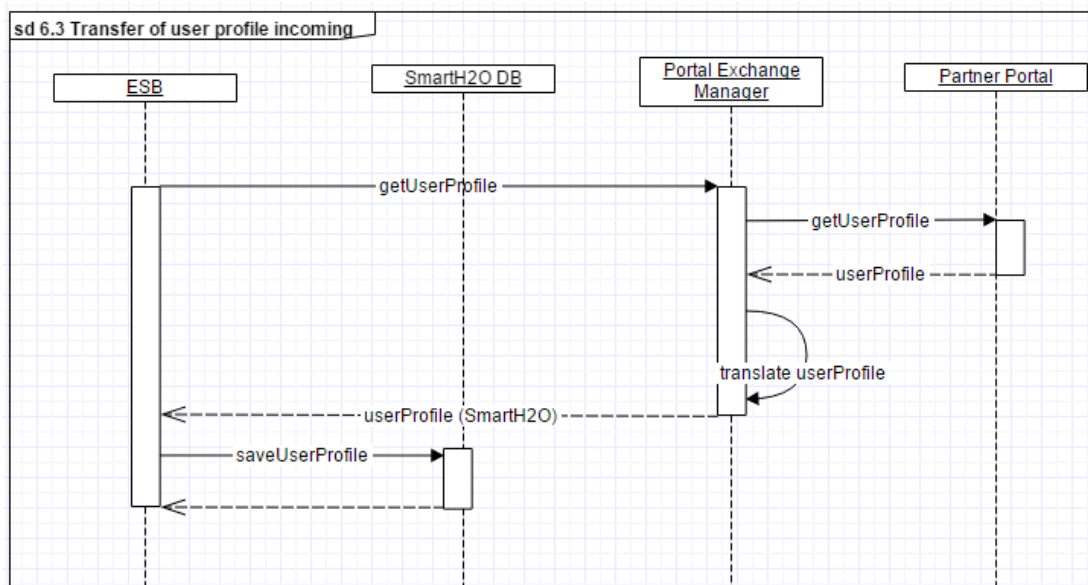


Figure 27: Sequence Diagram Use Case 6.3.

Use Case 6.4: Transfer of user profile – outgoing

Use Case 6.4	<i>Transfer of user profile - outgoing</i>	
Goal in Context	<i>Transfer of user profile from Smarth2O to partner portal</i>	
Precondition	<i>Partner portal was previously registered in Smarth2O</i>	
Success Condition	End	<i>Outgoing user profile is saved to partner portal database</i>
Failed Condition	End	<i>Outgoing user profile is not saved to database</i>
Primary actors	<i>PartnerPortal, Portal Exchange Manager</i>	
Secondary Actors	<i>ESB, Smarth2O DB</i>	

Steps	- <i>PartnerPortal sends a profile transfer request to Portal Exchange Manager</i> - <i>Portal Exchange Manager retrieves user profile from SmarH2O</i> - <i>Portal Exchange Manager sends user profile to Partner Portal</i> - <i>Partner Portal saves user profile to its database</i>
Note	

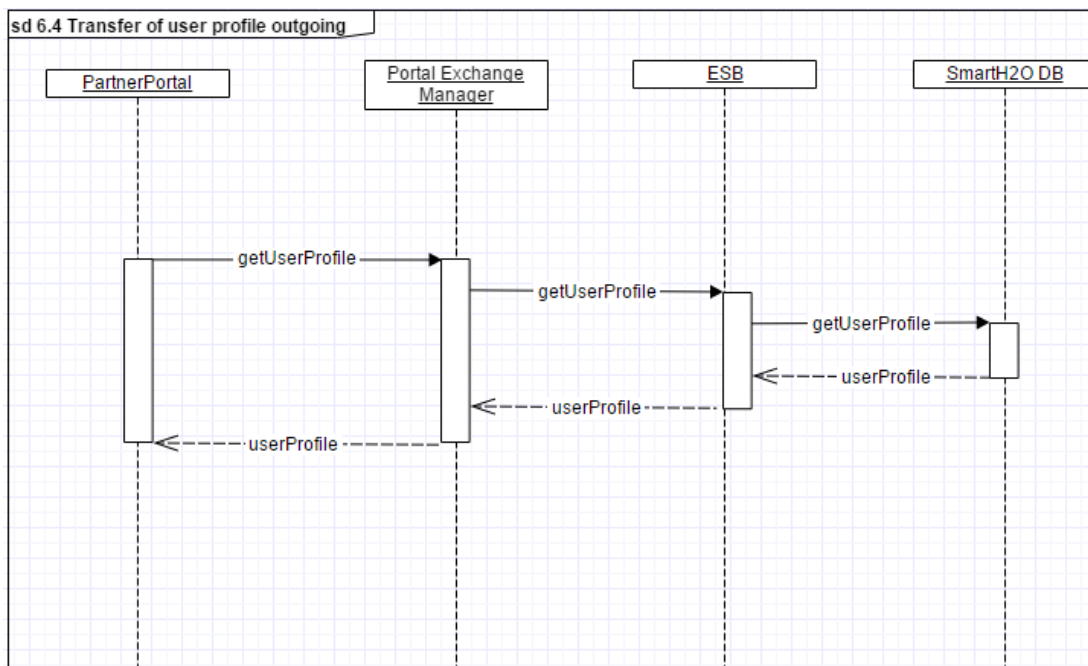


Figure 28: Sequence Diagram Use Case 6.4.

3.8.2 Component Interfaces Specification

[GET] authenticateIncomingUser(portalId, partnerPortalKey, userId).

Authenticate incoming user against SmarH2O platform and provide authentication token to partner portal.

Parameters:

- *portalId*. ID of portal that requests authentication. Portal must be enlisted as Partner Portal in SmarH2O DB
- *partnerPortalKey*. Key / token. Identification key / token that ensures identity of client application. This key / token must exist in Partner Portal record in SmarH2O DB and must be provided to Partner Portal for requests against SmarH2O components
- *userId*. Id of user that requests authentication

The endpoint is:

{webappUrl}/PortalExchangeManagerWebServiceREST/authenticateIncomingUser/
authenticateIncomingUser.do

[GET] *getUserProfile(portalId,partnerPortalKey,authenticationKey, userId)* Returns Smarth2O user profile

A sample of JSON response for a successful authentication is:

```
{
  "authResult":
  {
    "authKey": "FFD321324.m432dssadeqwiopi432",
    "authenticated": "Yes"
  }
}
```

A sample JSON response for a failed authentication is:

```
{
  "authResult":
  {
    "authenticated": "No",
    "reason": "user does not exist"
  }
}
```

[GET] *getUserProfile(portalId,partnerPortalKey,authenticationKey, userId)*

Returns Smarth2O user profile

Parameters:

- *portalId*. ID of portal that requests authentication. Portal must be enlisted as Partner Portal in Smarth2O DB
- *partnerPortalKey*. Key / token. Identification key / token that ensures identity of client.application This key / token must be exist in Partner Portal record in Smarth2ODB and must be provided to Partner Portal for requests against Smarth2O components
- *userId*. Id of user whose profile need to be retrieved

The endpoint is:

`{webappUri}/PortalExchangeManagetWebServiceREST/getUserProfile/getUserProfile.do`

A sample of JSON response for a successful retrieval of user profile is:

```
{
  "userId":1220211102,
  "sourceSystem":"Smarth2O",
  "result":
  {
    "status":"Ok",
    "reason":"success"
  },
  "profileData":
  {
```

```

    "name": "John",
    "surname": "Doe",
    "dateofbirth": "1982-09-11",
    "gender": "male",
    ...
  }
}

```

A sample of JSON response for a failed retrieval of user profile is:

```

{
  "userId": 1220211102,
  "sourceSystem": "SmarrH2O",
  "result": {
    "status": "nOk",
    "reason": "error message"
  }
}

```

3.9 Water Utility Admin Portal

Figure 29 shows the essential component diagram of the Water Utility Admin Portal.

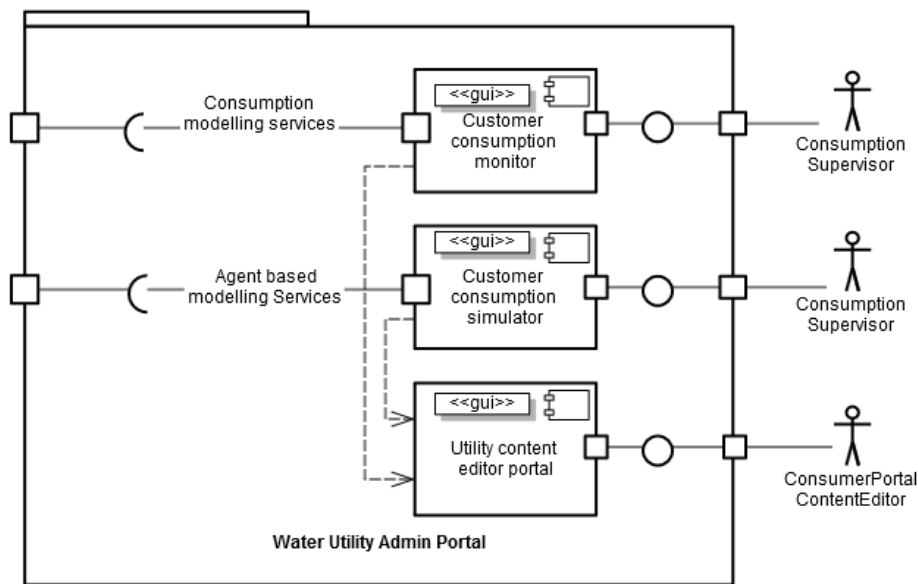


Figure 29: Water Utility Admin Portal Component Diagram.

The portal offers services to the Consumption Supervisor and the Consumer Portal Content Editor, both administrative users of the water utility company. It is organised in three main sub-components: a Consumption Monitor, which allows the Consumption Supervisor to check the consumption data of specific consumers; the Customer Consumption simulator, which allows the Consumption Supervisor to run simulation based on specific simulated incentive campaigns and on the services offered by the Agent-Based Modelling component; and the Utility content editor Portal, which permits the Content Editor to provide content for feeding the Consumer Portal.

Component Name	Water Utility Admin Portal
Leader	POLIMI
Partners	SETMOB
Goals	The component manages the monitoring and simulation of aggregate water consumption
User cases	<i>use</i> 7.1 Visualizing aggregate household consumption information by geo-location 7.2 Identifying customer segments 7.3 Setting action and reward types for specific user segments/groups 7.4 Predicting customer segment consumption behavior 7.5 Predicting behavior on incentive response 7.6 Predicting customer response to pricing scheme 7.7 Create content (news, tips for water consumption optimization,...)
Provided Interfaces	This component is a web application, which integrates inputs from external sources.
Dependencies / Required Interfaces	The component requires the implementation of the following interfaces: - [GET]getGeoLocations(): retrieves all geolocation divisions of Water Utility - [GET]getUserAttributes(): retrieves all user attributes available for consumers of Water Utility - [GET]getSegments(): retrieves all consumer segments, already defined for a Water Utility - [GET]getWaterConsumption(geoLocations, segments, period): retrieves actual water consumption data for specified geolocations, segments, for a specified period for a Water Utility. - [GET]getWaterConsumptionSimulationPastInfo(geoLocations, segments, period): retrieves simulation of user consumption based on past consumption info. - [GET]getWaterConsumptionSimulationIncentive(geoLocations, segments, period, incentive): retrieves simulation of user consumption based on incentive/reward. - [GET]getWaterConsumptionSimulationPricing(geoLocations, segments, period, pricingScheme): retrieves simulation of user consumption based on pricing scheme. - [GET]getActionRewards(segment): retrieves existing game actions and associated rewards for specified consumer segment - [GET]getSimulationIncentiveRewards(): retrieves existing incentive/rewards that were implemented by consumption simulation engines (ABM) - [GET]getSimulationPricingSchemes(): retrieves a list existing pricing schemes that were implemented by the consumption simulation engines (ABM) - [POST]saveSegment(userAttributes): creates or updates a segment based on a collection of user attributes for Water Utility - [POST]saveActionRewards(segments, actions, reward): creates game actions and associated reward for specified consumer segments

3.9.1 Main Use Cases

Use Case 7.1	<i>Visualizing aggregate household consumption information by geo-location</i>
Goal in Context	<i>Identifying e.g. leakage or consumption metrics of customers to take appropriate action (e.g. adjust pumping, fix leaks, adapt pricing schemes)</i>
Precondition	<i>Customer households are metered; Household consumption data is available</i>
Success End Condition	<i>Household consumption data is visualized.</i>
Failed End Condition	<i>Household consumption data is not visualized.</i>
Primary actors	<i>ConsumptionSupervisor</i>
Secondary Actors	<i>ESB, SmartH2ODB</i>
Steps	- <i>User selects specific geographic region / district / household</i> - <i>The system visualizes the aggregated metered water consumption data of a specific geographic region / district / household</i> - <i>The user can interact with the visualization by choosing different zoom levels of the information (hourly, daily, monthly)</i> - <i>The user can compare average to a set of available other aggregate averages (e.g. municipality, district)</i> - <i>If data available, the user can compare the consumption with amount of pumped water to identify potential leakages</i>
Note	

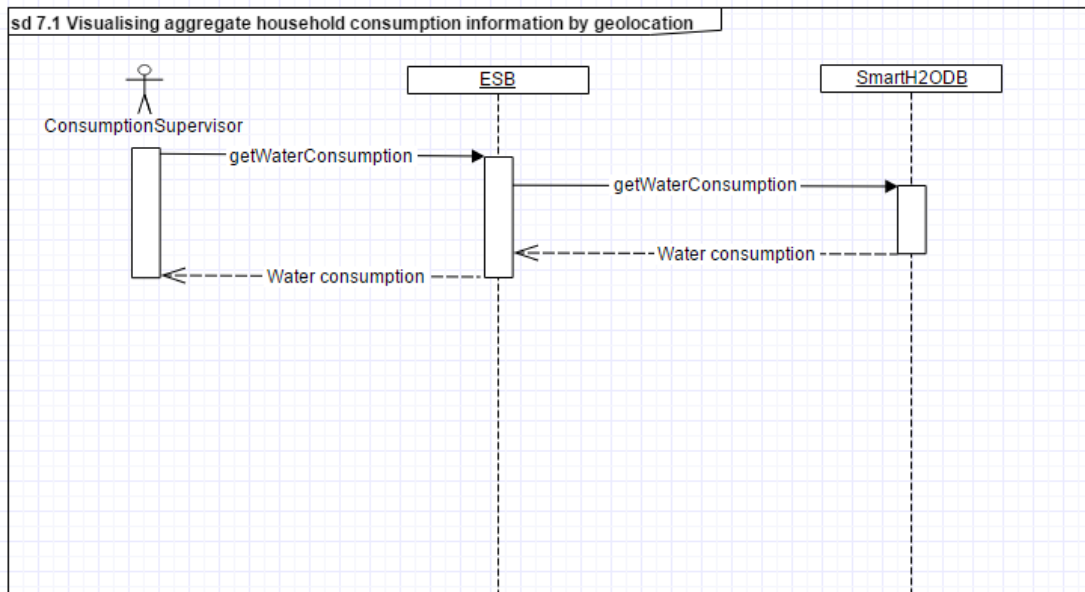


Figure 30: Sequence Diagram Use Case 7.1.

Use Case 7.2	Identifying customer segments
Goal in Context	<i>Identifying specific customer segments to provide targeted incentives and personalized feedback for water saving.</i>
Precondition	<i>Customer households have provided household and consumption information. Households were metered.</i>
Success End Condition	<i>The identified customer segments are validated based on observational data.</i>
Failed End Condition	<i>The observational data does not confirm the identified customer segments. The identified customer segments are not fine-grained enough.</i>
Primary actors	<i>Consumption Supervisor</i>
Secondary Actors	<i>ESB, SmartH2ODB</i>
Steps	- <i>System displays the behavioural customer and household attributes.</i> - <i>Based on this information, the user can identify and save a specific customer segment by selecting a subset of attributes.</i> - <i>System saves customer segments.</i>
Note	

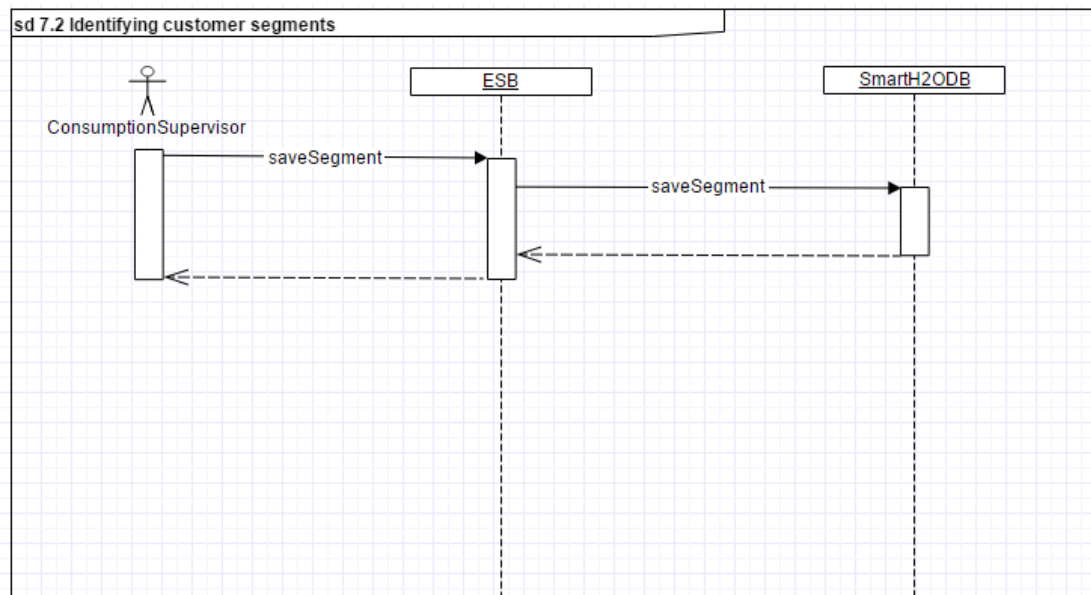


Figure 31: Sequence diagram Use Case 7.2

Use Case 7.3		<i>Setting actions and reward types for specific user segments/groups</i>
Goal in Context		<i>Provide personalized actions and rewards based on specific customer segments.</i>
Precondition		<i>User profiles and segments are available in the system.</i>
Success Condition	End	<i>Different actions and rewards are linked to each user segment.</i>
Failed Condition	End	<i>The system is not able to propose different actions and rewards for each user segment.</i>
Primary actors		<i>Consumption Supervisor</i>
Secondary Actors		<i>ESB, SmartH2ODB, Gamification Engine</i>
Steps		<i>User selects segment</i> <i>User defines reward/incentive</i> <i>User selects actions to be performed to get the reward/incentive</i> <i>User saves the actions and rewards for specified segment</i>
Note		

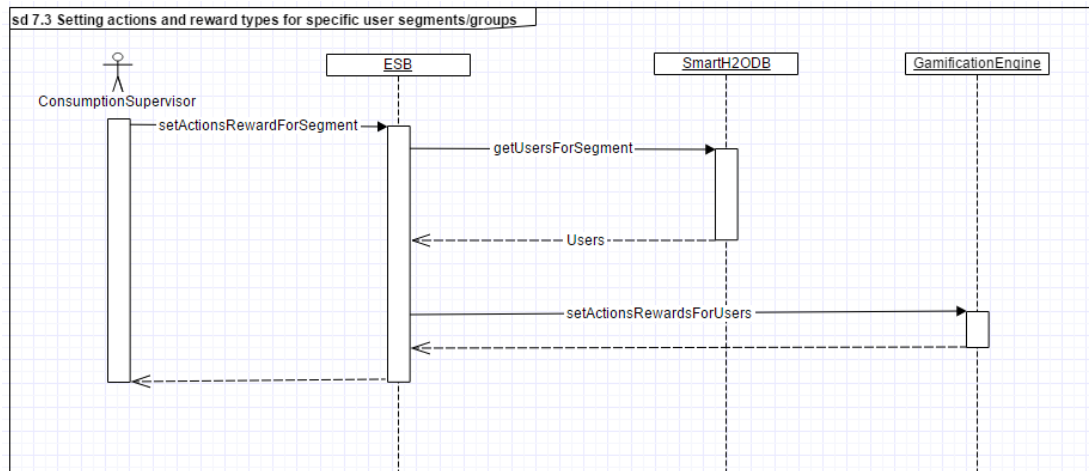


Figure 32: Sequence diagram Use Case 7.3.

Use Case 7.4	Predicting customer segment consumption behavior on past information
Goal in Context	<i>Water consumption prediction by customer segment based on past information and customer segment defined from information available from the app.</i>
Precondition	<i>Customer households have provided household and consumption information. Households were metered.</i>
Success Condition	End <i>The model is able to accurately predict customer segment consumption behaviour.</i>
Failed Condition	End <i>The model is not able to accurately predict customer segment consumption behaviour.</i>
Primary actors	<i>Consumption Supervisor</i>
Secondary Actors	<i>ESB, ABM, SmartH2ODB</i>
Steps	<i>Users selects simulation type= based on past information</i> <i>User selects segment, and future simulation period</i> <i>Based on behaviour model, system predicts future consumption behaviour based on past information</i>
Note	

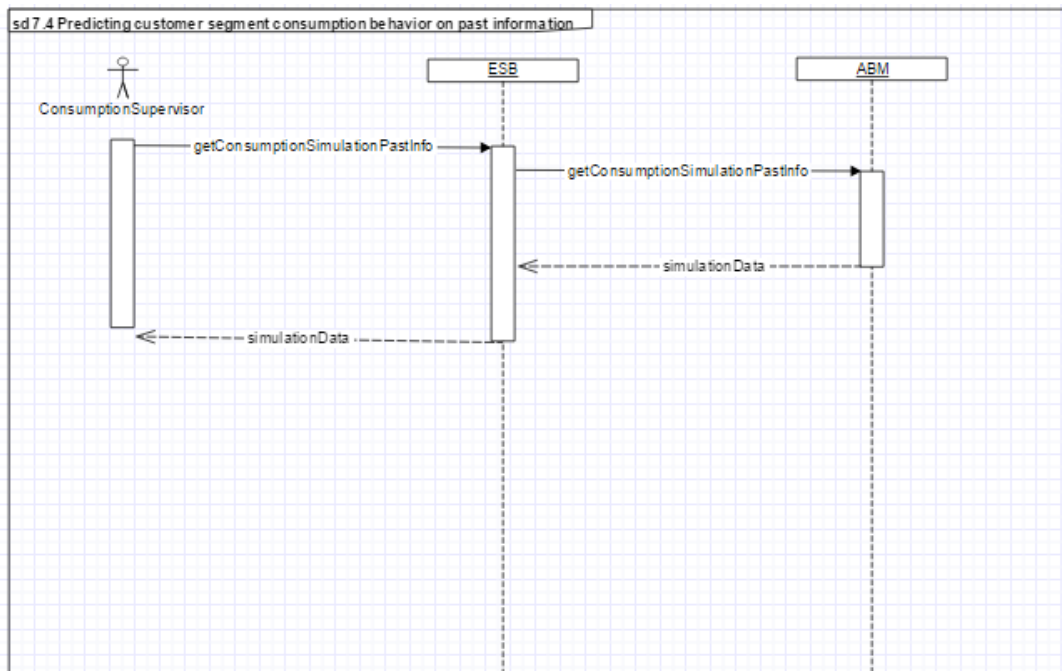


Figure 33: Sequence diagram Use Case 7.4.

Use Case 7.5		Predicting customer segment consumption behavior on incentive response
Goal in Context		<i>Prediction of customer water consumption response to specific reward / incentive scheme based on the estimated model of customer response to rewards/incentive schemes.</i>
Precondition		<i>Customer households have provided household and consumption information. Households were metered. Relevant and feasible rewards have been identified. Customer response model has been estimated.</i>
Success Condition	End	<i>The model is able to accurately predict customer consumption response to reward/incentive scheme.</i>
Failed Condition	End	<i>The model is not able to accurately predict customer consumption response to reward/incentive scheme.</i>
Primary actors		<i>Consumption Supervisor</i>
Secondary Actors		<i>ESB, Gamification Engine, ABM</i>
Steps		<i>Users selects simulation type = based on reward/incentive</i> <i>User selects segment, and future simulation period</i> <i>User selects reward/incentive used for simulation</i> <i>Based on behaviour model, system predicts future consumption behaviour based on reward/incentive response</i>
Note		

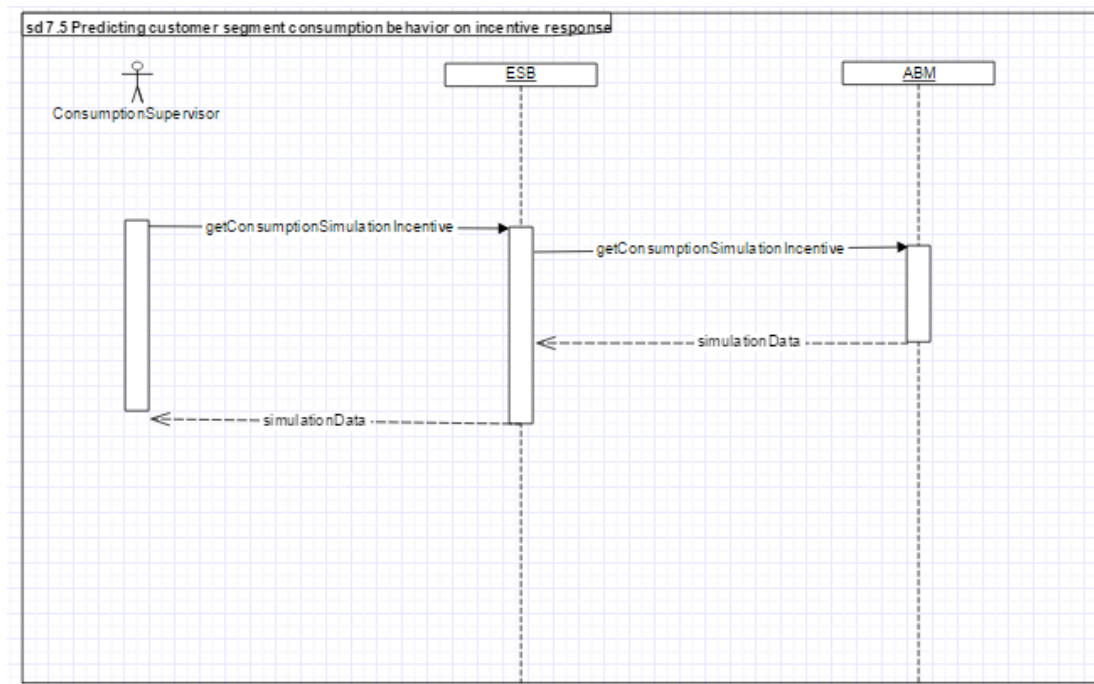


Figure 34: Sequence diagram Use Case 7.5.

Use Case 7.6	Predicting customer response to pricing scheme
Goal in Context	<i>Prediction of customer water consumption response to specific pricing schemes such as blocking rates.</i>
Precondition	<i>Customer households have provided household and consumption information. Households were metered.</i>
Success End Condition	<i>The model is able to accurately predict customer consumption response to pricing scheme.</i>
Failed End Condition	<i>The model is not able to accurately predict customer consumption response to pricing scheme.</i>
Primary actors	<i>Consumption Supervisor</i>
Secondary Actors	<i>ESB, Smarth2ODB, ABM</i>
Steps	<i>Users selects simulation type = based on pricing schemes</i> <i>User selects segment and future simulation period</i> <i>User selects pricing scheme used for simulation</i> <i>Based on behaviour model, system predicts future consumption behaviour based on pricing scheme response</i>
Note	

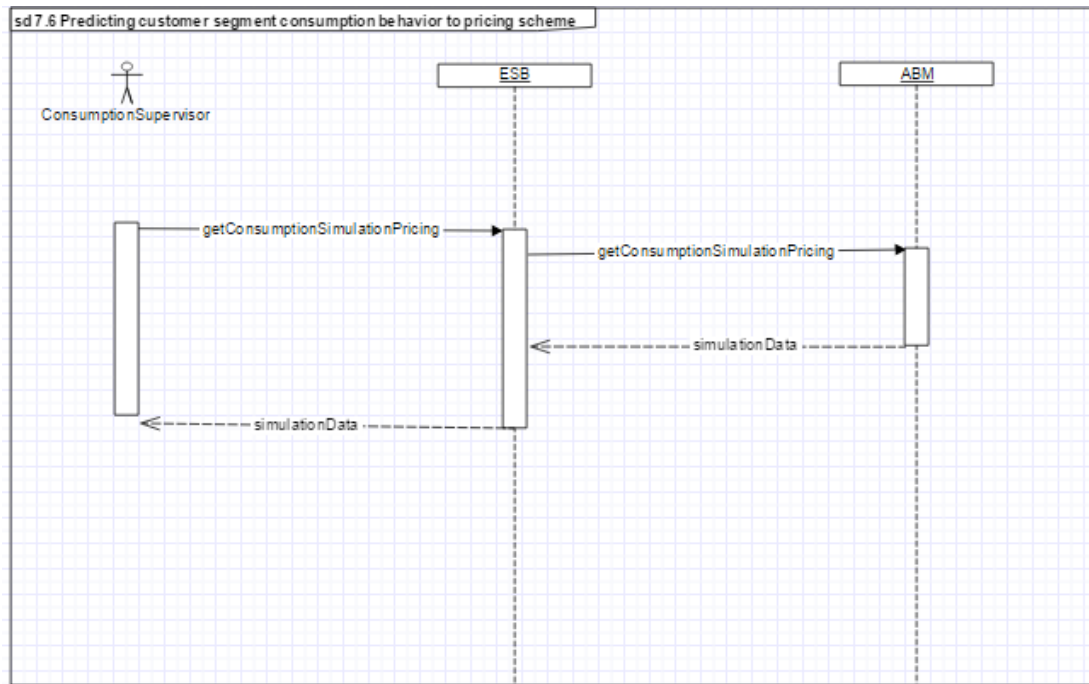


Figure 35: Sequence diagram UseCase 7.6.

3.10 Authentication Gateway

This component role is to centralize and consolidate the user registration data within the Smarth2O platform database, also for the users registered in the external components, which may use their own local user databases. The involved components are:

- Water Utility Customer Portal
- Gamification Engine
- Games Portal
- Water Utility Admin Portal

Also, this component provides a unique point for authentication to all the users primarily registered at a component level, in order to login to other components by using the original set of credentials without the need to perform another registration.

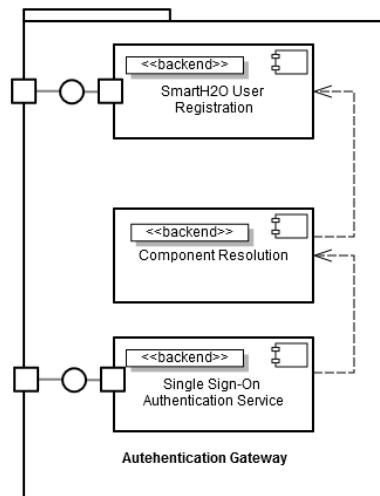


Figure 36: Authentication Gateway component diagram.

Component Name	Authentication Gateway
Leader	SETMOB
Partners	POLIMI
Goals	Consolidates the user registration data within the SmartH2O platform database. Provides a unique point for authentication to all the users primarily registered at a component level, in order to login to other components.
User use cases	- Consolidates the user registration in the SmartH2O platform database - User authentication through Single Sign-On
Provided Interfaces	- [POST]signUpToSmartH2ODB(userAttributes): creates a user registration to the SmartH2O DB - [POST]unsubscribeFromSmartH2ODB: remove a registration from the SmartH2O DB - [POST]updateUserProfileToSmartH2ODB: updates user profile information like: family members, house configuration, appliances etc.. to the SmartH2O DB
Dependencies / Required Interfaces	The component requires the implementation of the following interfaces: - [POST]delegateAuthentication: authenticates the user who is passing the credentials - [POST]delegationSignOff: signs off the user authenticated through Single Sign On

3.10.1 Main Use Cases

Use Case 8.1: Consolidates the user registration in the SmartH2O platform database

Use Case 8.1	Consolidates the user registration in the Smarth2O platform database	
Goal in Context	<i>Register in the central database the user that has been previously registered in a component databases</i>	
Precondition	<i>The user has performed registration in a component database</i>	
Success Condition	End	<i>The user is registered in the Smarth2O central database</i>
Failed Condition	End	<i>The component is not able to register the user in the Smarth2O central database</i>
Primary actors	Smarth2O User Registration	
Secondary Actors	ESB, Smarth2ODB	
Steps	- The user performs registration in any component managing its own user database - The component calls the Smarth2O User Registration - The user is registered in the Smarth2O database	
Note	<i>This use case is triggered by any of the components performing their own user registration:</i> - Water Utility Customer Portal - Gamification Engine - Games Portal - Water Utility Admin Portal	

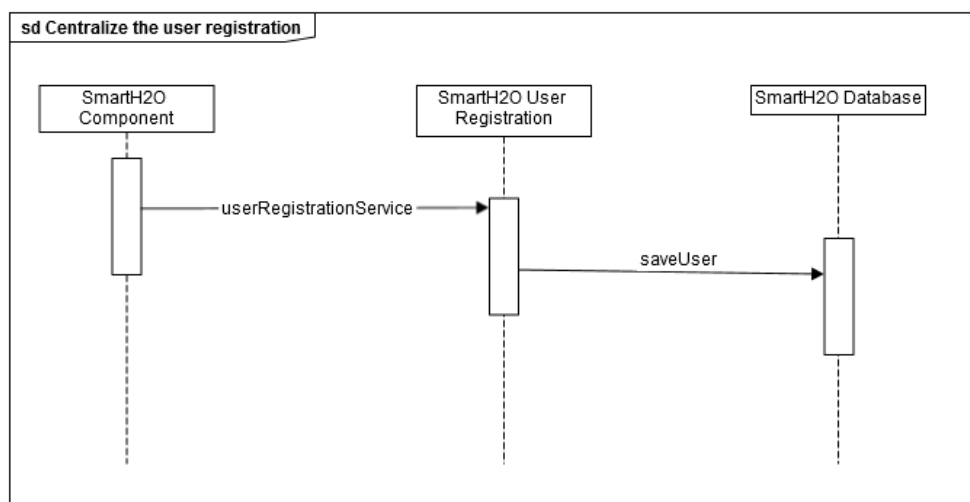


Figure 37: Sequence diagram for Use Case 8.1.

Use Case 8.2: User authentication through Single Sign-On

Use Case 8.2	User authentication through Single Sign-On
---------------------	---

Goal in Context	Allow an user authenticated in a component to access another component where he has not directly performed registered through a Single Sign-on mechanism
Precondition	<i>The is registered in the Smarth2O platform database</i>
Success Condition	End <i>The user access a component where he has not directly performed registration</i>
Failed Condition	End <i>The user is not able to access a component where he has not directly performed registration</i>
Primary actors	User, Single Sign-On Authentication Service
Secondary Actors	ESB, Water Utility Customer Portal, Gamification Engine, Games Portal, Water Utility Admin Portal
Steps	- <i>The user logs in to a component where he previously performed direct registration</i> - <i>The user request access to another component where he did not previously accomplished a direct registration</i> - <i>The user access the other component</i>
Note	

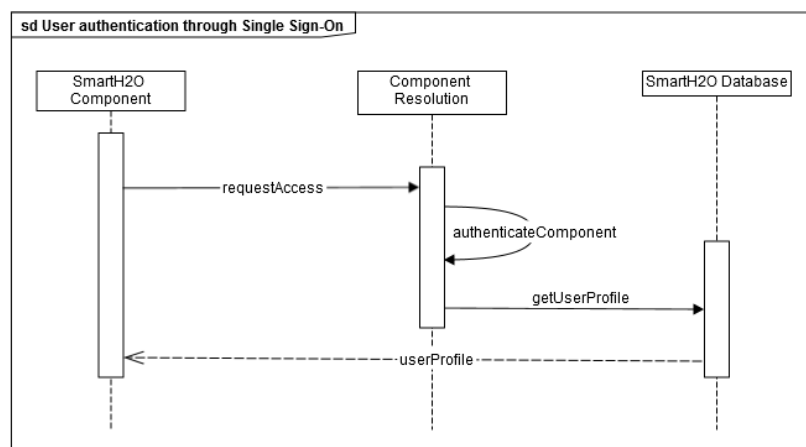


Figure 38: Sequence Diagram for Use Case 8.2.

3.11 Pricing Engine, Agent Based Modelling and Models of User Behaviour

This section overviews the current understanding of the role and internal structure of the Pricing Engine, Agent Based Modelling and Models of User Behaviour. The present specification will be further detailed and refined as part of the technical progress of the work in WP3 User Modelling, WP4 Saving water by social awareness, WP5 Saving water by dynamic pricing.

The **Pricing Engine** offers services to both the Customer and the Consumption Supervisor, who access it through dedicated user-interfaces. The Pricing Engine Backend manages the dialogue of the two above mentioned user groups with the price modelling and simulation

functionality offered by the Pricing Modelling Services. The resulting coarse organization of the component is overviewed in Figure 39.

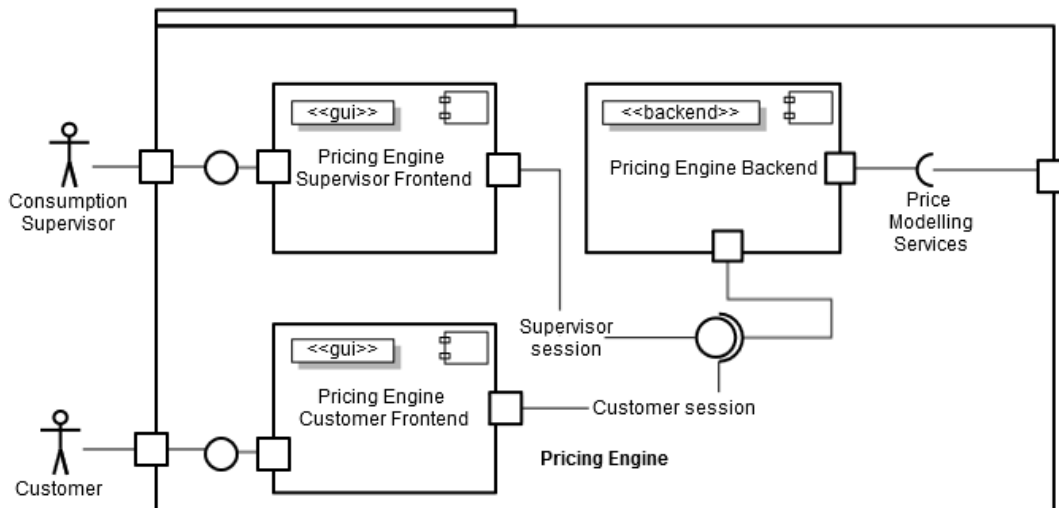


Figure 39: Pricing Engine Component Diagram.

The **Agent Based Modelling** component allows the water utility to simulate whole districts of users, thus extrapolating user models at a larger scale. The agent based model includes influence/mimicking mechanisms and social interaction among the consumers, and thus will be employed by the water utility to understand how some user types (leaders/influencers) can stimulate a behavioural change on other users.

It exploits the **Models of User Behaviour** component embodies models describing water consumption behaviour of a single customer or a class of consumers. Its internal coarse organization is shown in Figure 40.

Through this component, the water utility can visualize the water consumption of each customer at a fixture/appliance level, in order to identify consumption patterns and trends, and thus identifying the most promising areas where conservation efforts may be polarized. Furthermore, based on their behaviour, consumers are clustered into different classes, and the water utility can foresee the consumer behaviour in front of exogenous variables (climate), social awareness campaigns, social pressure, water restrictions, etc. For a description of the algorithms exploited to model the user behaviour, see deliverable D3.2: First user behaviour models.

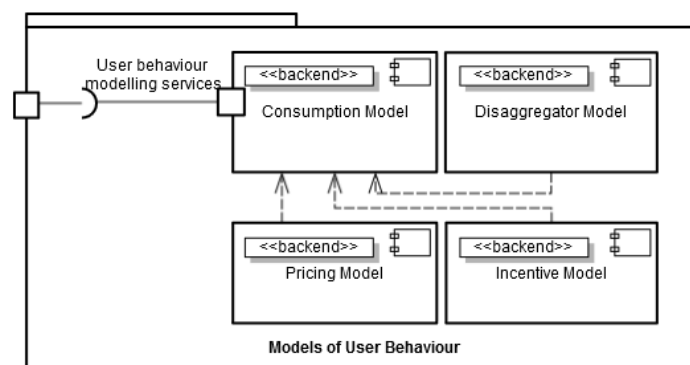


Figure 40: Models of User Behaviour Component Diagram.

4. Deployment Architecture

As detailed in the component architecture diagram, Smarth2O Platform is a heterogeneous collection of components with different functionality that interact to each other to deliver the proposed objectives. Each component by itself deliver a certain functionality but it also requires general platform services like authentication, communication or data services. Components also need data or computation from other component of the platform.

The major technical criterion for the selection of the platform architecture was to have a Service Oriented Architecture (SOA). This requirement will ensure loose coupling and isolation of logic of components. Being a current industry standard, SOA architecture will also allow easy integration of Smart H2O platform with other existing platforms. SOA architecture can also provide scalability, security and availability of the platform.

SOA architecture imposed that interaction between components and interaction between components and the platform must be realized via an integration middleware component. The most suitable integration component for SOA architecture was an Enterprise Service Bus (ESB) component. This architecture pattern will also allow easy but solid integration of Water Utility existing systems with Smarth2O Platform. For example an existing CRM system of Water Utility can be easily plugged in to provide segmentation information of customers.

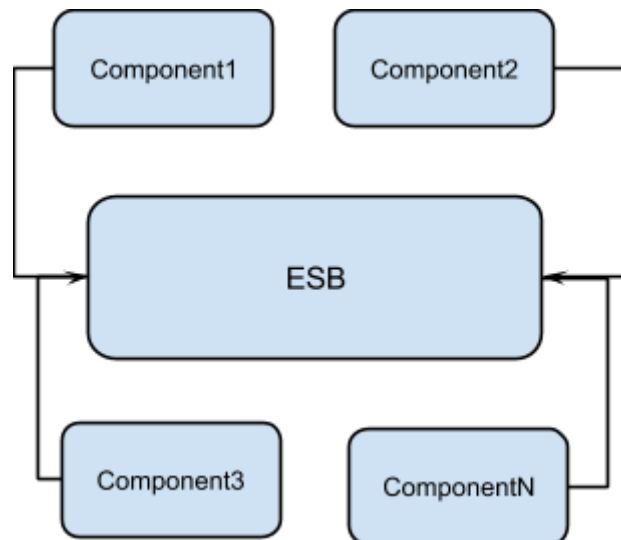


Figure 41- General integration platform architecture

SOA architecture will be implemented using open and standardized technologies and frameworks like REST, HTTP(S), J2EE (Java Enterprise Edition), JBoss Fuse. Each component will be also internally implemented by open source frameworks and languages, industry proven technologies but in the context of the Smarth2O platform they will expose a common interface of Web Services so that they can be integrated in the platform with the rest of the components.

Most components contain an internal multi-tiered architecture relying on web-based J2EE framework. That means that the client application will typically run in a web browser.

Technical Implementation details of each component can be found in ANNEX2 - List of technologies, frameworks and languages used.

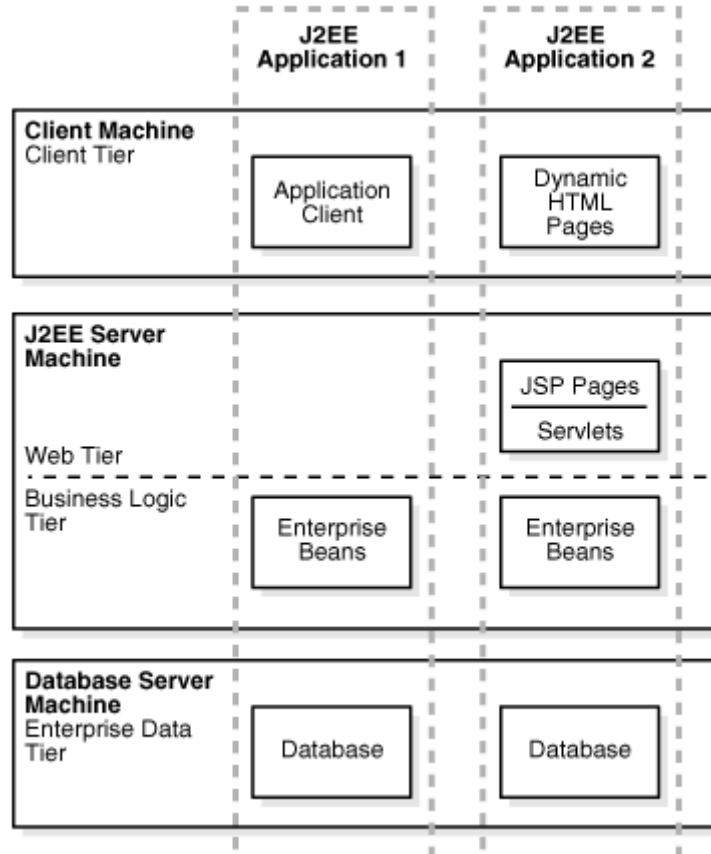


Figure 42 - Component level multi-tiered architecture (from docs.oracle.com)

The choice of multi-tiered architecture will provide:

- scalability of Business Logic and Data tiers
- no client-side maintenance
- separation of development roles. User Interface development tasks are separated from Business Logic development tasks. Development task separation allow specialization and scalability of development resources

In the context of Water Utility oriented applications this choice of architecture and technologies will deliver, among many other, these main benefits:

- **interoperability.** Easy and rapid adaptation of SmartH2O platform to customer specifics. Each Water Utility provider is supposed to operate more management systems that can be integrated in the Platform to provide enhanced Customer and Water Usage data. Also they can be fed by Smart H2O with behavioural data and consumption scenarios.
- **scalability.** Water Utility companies may handle huge amounts of data. Distribution of computing, data access and storage and presentation operations allow efficient resources allocation in such quantity that the Platform delivers required performance level.

4.1 Development Environment Deployment

The hardware infrastructure of the development and testing server is powered by a Dell PowerEdge 6850 with 4 Intel Xeon dual core CPUs at 3.2 GHz, 32GB of DDR2 ECC RAM and 1.5 TB of RAID storage. These resources are split by VMware ESXi 5.5 into 4 virtual machines:

- CI - 4 vCPUs, 8GB RAM, 500GB storage
- SM - 8 vCPUs, 6GB RAM, 256GB storage
- SM1 - 4 vCPUs, 10GB RAM, 260GB storage
- SM2 - 2 vCPUs, 4GB RAM, 254GB storage
- AP - 4 vCPUs, 5GB RAM, 5- gb storage

The virtual machines have separate roles, as follows:

CI - continuous integration using Jenkins, development using WebRatio using VNC access, SonarQube for code analysis, JBoss FUSE ESB, MySQL server with testing and production data, Sonatype Nexus for repository management, Tomcat 6 for testing

All SM VMs come together to create a small Hadoop cluster that can process data in a distributed, scalable and high availability fashion.

SM - Ambari for monitoring of the Hadoop system, YARN App Timeline Server, Ganglia Server, MapReduce2 History Server, Nagios Server, YARN Resource Manager, HDFS SNameNode, ZooKeeper Server, HDFS DataNode, Ganglia Monitoring, YARN NodeManager

SM1 - HDFS NameNode, Oozie Server, ZooKeeper Server, Ganglia Monitoring, YARN NodeManager, FTP Server for receiving XML files from SES, SMDMC Manager for managing the processing flow of the data received from the water utility.

SM2 - HDFS DataNode, ZooKeeper Server, Ganglia Monitoring, YARN NodeManager

As for clients, all SM VMs have installed: HDFS Client, MapReduce2 Client, Oozie Client, Pig, Sqoop, Tez Client, YARN Client, ZooKeeper Client.

AP1 - CentOS 7, Java 1.7, Tomcat 7, Jboss Fuse, ESB, Customer Portal, Gamification Engine, Business Dashboard

PM - power management, APC PowerChute appliance.

Ambari is the central piece, it monitors and starts the hadoop services. Oozie is used as the job scheduler for the data processing, it distributes the load and keeps the flow in check, providing feedback if and how a processing job fails. The Oozie jobs are started by the SMDMC component that uses Apache Camel to create, start and manage the Oozie job.

Figure 43 illustrates the software artifacts and how the VMs perform together.

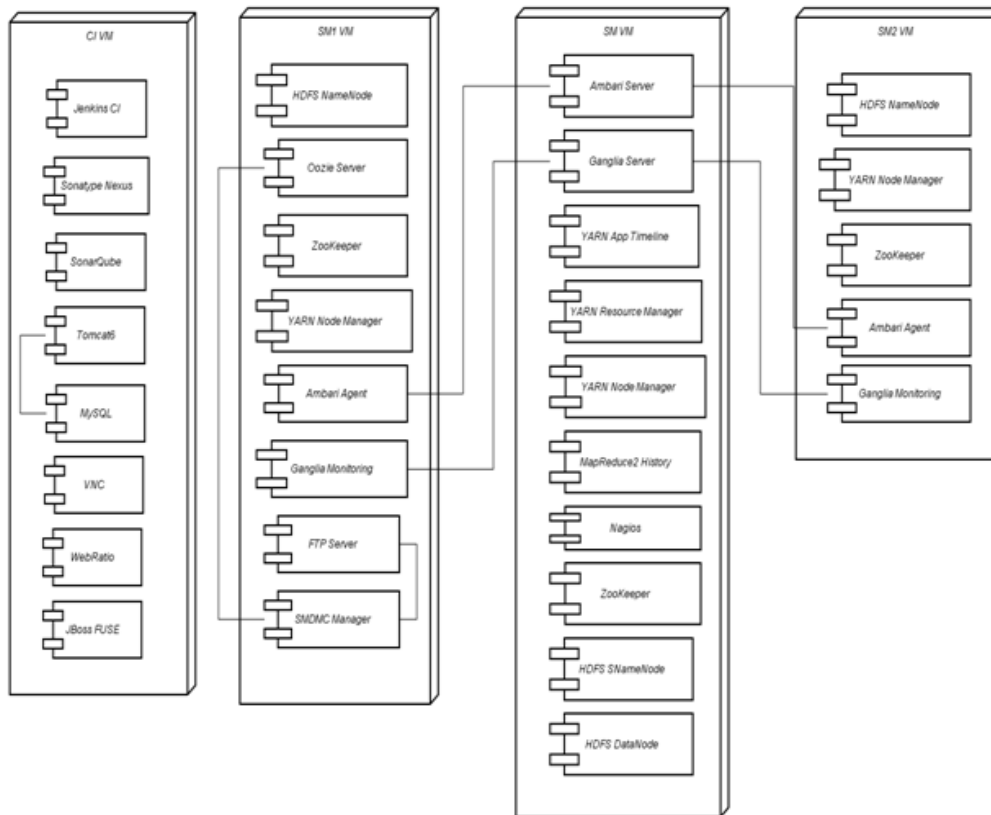


Figure 43 - SmarthH2O development server deployment diagram

4.2 Production Environment Deployment

The Production Environment will have a different virtualization layer than the Development Environment. Instead of VMWare virtualization layer, it will employ OpenStack (www.openstack.org) virtualization environment which is specialized in Cloud Deployments. OpenStack can manage large pools of hardware resources and provide high-availability, scalability, storage and computing for cloud infrastructures.

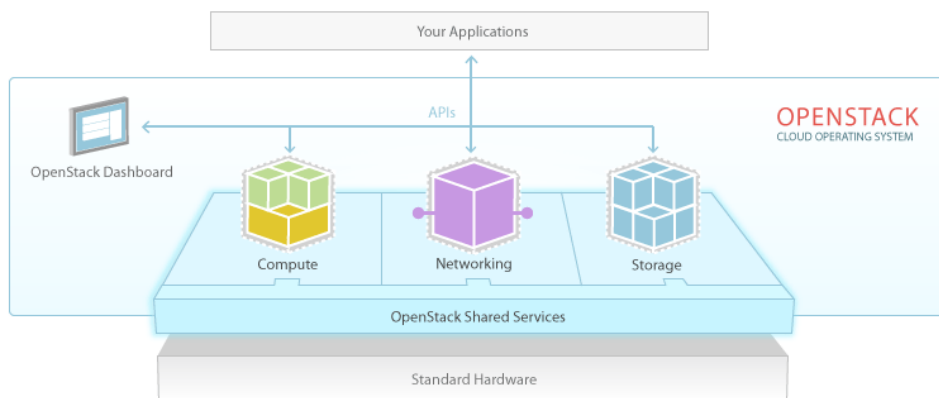


Figure 44 - OpenStack architecture

The main components of OpenStack framework with respect to the picture above are:

- **Compute**
 - Cloud fabric controller (Nova) - Compute architecture designed to scale horizontally on standard both virtual and bare-metal hardware
- Storage**
 - Object Storage (Swift) - scalable redundant file and object level distributed storage
 - Block Storage (Cinder) - provide persistent block-level distributed storage
- Networking**
 - Neutron - system for managing networks and IP addresses
- Dashboard**
 - Horizon - graphical user interface to access, provision and automate cloud-based servers
- Services**
 - Identity Service (Keystone) - repository of users mapped to services they can access
 - Image Service (Glance) - provides discovery, registration, and delivery services for disk and server images

Every server in Production Environment will run over the OpenStack cloud infrastructure layer. The exact number of virtual servers deployed in the Production Environment will highly depend on the exact context of Water Utility technical and user requirements. The most important thing is that the Smarth2O both software and deployment architecture is scalable and can provide the Client flexibility to cover any real-life scenario.

The proposed architecture serves the requirements for a medium scale Production environment (approx. 10,000 water utility customers):

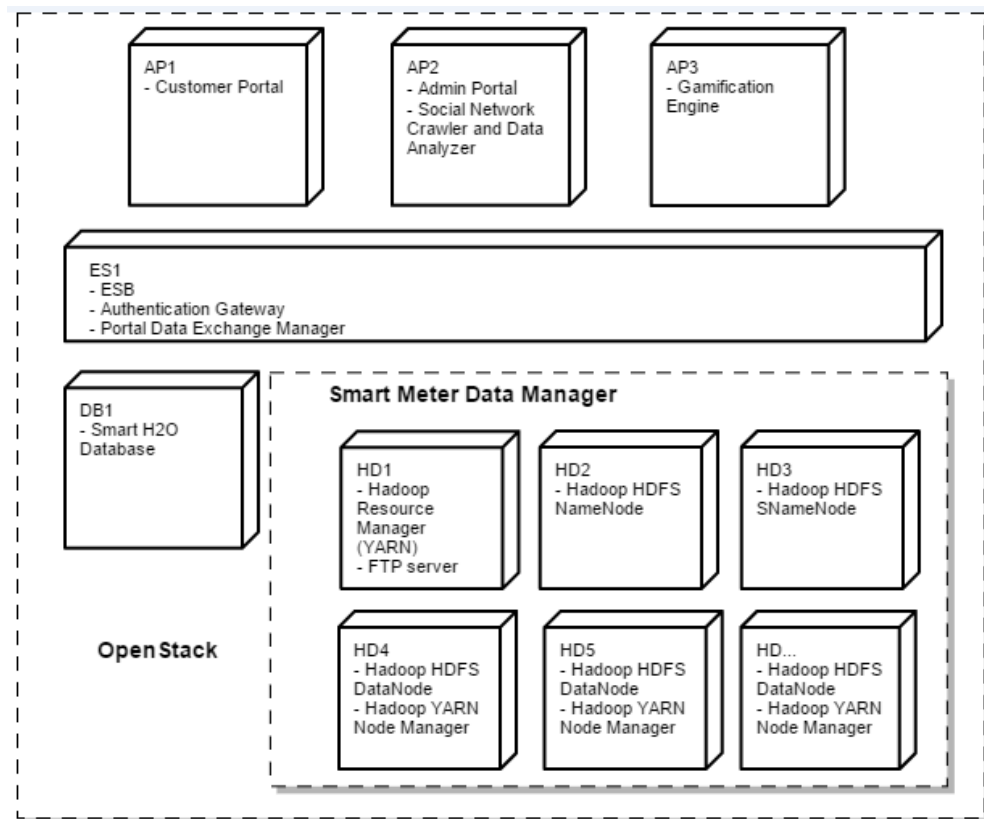


Figure 45 - Production deployment architecture

The mapping of Smarth2O platform on deployment architecture and technical details of each deployed server are presented in Table 2 below.

Table 2 – Smarth2O platform technology mapping.

SRV	SmartH2O component	Hardware infrastructure (virtual)	Software infrastructure
AP1	Customer Portal	4 vCPUs, 10GM RAM, 50GB storage	Linux Java 7 Apache WebServer Tomcat J2EE Server
AP2	Admin Portal Social Network Crawler and Data Analyzer	4 vCPUs, 10GM RAM, 50GB storage	Linux Java 7 Apache WebServer Tomcat J2EE Server
AP3	Gamification Engine	2 vCPUs, 5GM RAM, 50GB storage	Linux Java 7 Apache WebServer Tomcat J2EE Server
ES1	ESB Authentication Gateway Portal Data Exchange Manager	4 vCPUs, 10GM RAM, 50GB storage	Linux Java 7 Apache WebServer Tomcat J2EE Server JBoss Fuse

DB1	SmartH2O Database	4 vCPUs, 12GM RAM, 1TB storage	Linux MySQL Database Server
HD1	SMDM - Hadoop Resource Manager FTP Server	4 vCPUs, 16GM RAM, 500GB storage	CentOS Java7 Ambari YARN Resource Manager YARN App Timeline Server Ganglia Server ZooKeeper Server
HD2	SMDM - Hadoop HDFS NameNode	4 vCPUs, 16GM RAM, 500GB storage	Linux Java 7 HDFS NameNode
HD3	SMDM - Hadoop HDFS SNameNode	2 vCPUs, 8GM RAM, 500GB storage	Linux Java HDFS SNameNode
HD4	SMDM - Hadoop HDFS DataNode SMDM - Hadoop YARN Node Manager	2 vCPUs, 8GM RAM, 500GB storage	Linux Java 7 HDFS DataNode YARN Node Manager
HD5	SMDM - Hadoop HDFS DataNode SMDM - Hadoop YARN Node Manager	2 vCPUs, 8GM RAM, 500GB storage	Linux Java 7 HDFS DataNode YARN Node Manager
HD...	SMDM - Hadoop HDFS DataNode SMDM - Hadoop YARN Node Manager	2 vCPUs, 8GM RAM, 500GB storage	Linux Java 7 HDFS DataNode YARN Node Manager

5. Conclusions and outlook

This deliverable is the first specification of the architecture and components of the SmartH2O platform. It contains the knowledge matured at month 9 of the project.

The overall architecture has been specified in terms of the major components, data stores, user interfaces, and system interfaces, as well as the principal control and data flows among them.

Several components have been examined at depth, which permitted us to identify clearly the supported use cases, and the required interfaces and message orchestrations.

Other components are under scrutiny, as they depend on the further refinement of the use cases in the demonstrators and on the progress of the scientific work in the more technical components, such as the pricing engine and user behaviour model components, which are due later in the project.

The specification of the architecture will be refined as the project progresses, and the result of this refinement will be incorporated in the documents accompanying the deliverables representing the various releases of the SmartH2O platform: *Platform implementation and Integration* D6.3 at month 12, D6.4 at month 24, and D6.5 at month 36.

6. References

- [Bozzon2014] Alessandro Bozzon, Piero Fraternali, Luca Galli, Roula Karam: Modeling CrowdSourcing Scenarios in Socially-Enabled Human Computation Applications. J. Data Semantics 3(3): 169-188 (2014)
- [Bramb2014] Marco Brambilla, Piero Fraternali, Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML, The MK/OMG Press, December 2014.
- [Karam2012] Roula Karam, Piero Fraternali, Alessandro Bozzon, Luca Galli: Modeling End-Users as Contributors in Human Computation Applications. MEDI 2012: 3-15

7. ANNEX 1 – List of Languages, Frameworks and Technologies

Smarth2O components will use various technologies, frameworks and development languages. The choice of technology for each component passed several criteria:

- Ability to deliver required functionality
- Open Source
- Open Standard oriented
- Documentation
- Proven record in real-life exploitation
- Commitment for future enhancements and bug-fixing
- Alignment with the entire technical solution of the platform

Smart H2O platform will be constructed using several types of software building blocks:

- New components or sub-components, developed as part of the current project
- Frameworks and technologies: ex. J2EE, Apache Camel Route
- Application Servers: ex. Apache JBoss, JBoss Fuse, Apache Hadoop
- Database Servers: ex. MySQL
- Communication Servers: ex. FTP, SMTP
- Management and Administration components. ex. MySQLWorkbench,

Each type of building block has different types of software characteristics. For example, a newly developed building block can be described by programming language used for development, frameworks and libraries used. An Application Server building block can be described by required operating systems.

For each Smarth2O component we detail below its most important software characteristics:

Smarth2O Database

- Operating system: Linux distribution
- MySQL database server
- Administration: Workbench client for MySQL administration and manual query execution

ESB – Enterprise Service Bus

- Operating system: Linux distribution
- Language: Java
- Application Server: JBoss Fuse
- Communication protocols: HTTP(S), REST Web Services
- Frameworks and technologies:
 - o Apache Camel Route used to define messaging routes and orchestration between platform component
 - o J2EE used to develop processing components

Smart Meter Data Manager

Back-end, batch oriented component that processes metered data and feeds the Smarth2O database with usage data.

- Operating system: Linux distribution
- Languages:
 - o Java
 - o Pig
 - o SQL
- Application Servers:
 - o Apache JBoss
 - o Apache Hadoop cluster
- Frameworks and technologies:

- o J2EE
- o REST WebServices
- o MapReduce
- Communication servers:
 - o FTP used to transfer metered usage files
- DBMS: MySQL relational database to store household and metered data
- Other storage: HDFS to store raw metered data files, intermediate processed metered data files

Customer Portal

Web application that manages Customer's interaction with consumption data, games actions and rewards.

- Operating system: Linux distribution
- Languages:
 - o Client side: IFML used for Model Driven Development , JavaScript, JQuery
 - o Server side: IFML used for Model Driven Development , Java
- Frameworks and technologies: J2EE, REST Web Services

Admin Portal

Web application that manages Utility Admin interaction with Customers, consumption data, games actions and rewards.

- Operating system: Linux distribution
- Languages:
 - o Client side: IFML used for Model Driven Development , JavaScript, JQuery
 - o Server side: IFML used for Model Driven Development , Java
- Frameworks and technologies: J2EE, REST Web Services

Gamification Engine

Back-end component that manages definition of games actions and rewards and processes users' actions

- Operating system: Linux distribution
- Languages:
 - o Client side: IFML used for Model Driven Development , JavaScript, JQuery
 - o Server side: IFML used for Model Driven Development , Java
- Frameworks and technologies: J2EE, REST Web Services

Games Platform

Mobile game trivia pattern to increase awareness regarding water consumption. Game result can be converted to game points that can be used by Gamification Engine.

- Operating system: Android, iOS
- Languages:
 - o Client side: C#, Unity 3D Player
 - o Server side: JavaScript (Node.js)
- Frameworks and technologies: Unity 3D Game Engine

Models of User Behaviour

Application that disaggregates user water consumption during the day and classifies users in segments.

This component does not have a software deliverable but a set of rules that will be implemented in Agent Based Modeling component

Agent Based Modeling

This component models user consumption behavior at a larger scale and estimates the impact of network effect due to users interactions.

- Operating system: Linux, MacOSX, Windows

- Languages:
 - o Client side: Java, AnyLogic Simulation Platform
 - o Server side: N/A
- Frameworks and technologies: Eclipse Framework

Authentication Gateway

- Operating system: Linux
- Languages:
 - o Server side: Java
- Frameworks and technologies: OAUTH2

Social Network Crawler and Data Analyser

- Operating system: Ubuntu
- Database: MongoDB
- Languages:
 - o Client side: Java
- Frameworks and technologies: Twitter Stream API

Social Network Connector

- Operating system: Ubuntu
- Languages:
 - o Client side: Java
- Frameworks and technologies: Social Network REST API

8. ANNEX 2 – User Interaction Flows

The User Interaction details for Customer Portal and Admin Portal components are detailed in the following sections:

- Customer Portal Basic Version
- Customer Portal Advanced Version
- Admin Portal

The following sections describe, using the standardized modelling language IFML [Bramb2014] described next, the user interaction and backend processing of these components.

8.1 IFML in a Nutshell

IFML supports the platform-independent description of graphical user interfaces for applications deployed or accessed on such systems as desktop computers, laptops, PDAs, mobile phones, and tablets. The main focus is on the structure and behavior of the application as perceived by the end user. The modeling language also incorporates references to the data and business logic that influence the user's experience. This is achieved respectively by referencing the domain model objects that provide the content displayed in the interface and the actions that can be triggered by interacting with the interface. This section introduces the essential features of IFML: its scope, the design rules behind it, its main modeling elements, and its role in the development process. The Section concludes with an initial example of the language.

8.1.1 Scope and perspectives

To better understand the aim and scope of IFML it may be useful to refer to the well-known Model-View-Controller (MVC) software architecture of an interactive application,¹ shown in Figure 46. MVC distinguishes the application internal status and business logic (Model), their representation in the user interface (View) and the rules governing the response to the user's interaction (Controller).

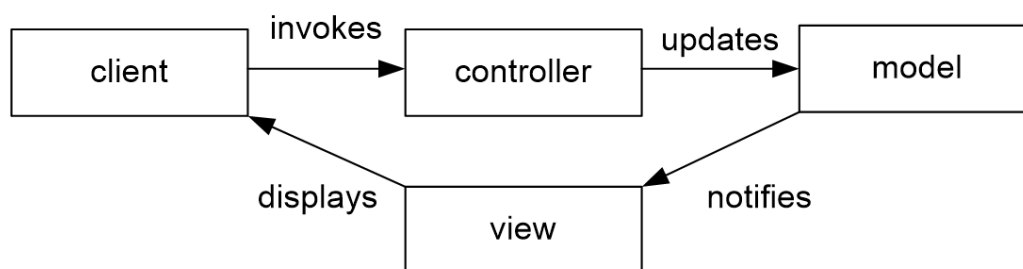


Figure 46- the Model-View-Controller architecture of an interactive application

IFML mainly describes the view, i.e., the content of the front end and the user interaction mechanisms available in the interface. More precisely, IFML covers various aspects of the user interface:

- The view structure: it expresses the general organization of the interface, in terms of ViewContainers, along with their nesting relationships, visibility, and

¹See, for example, <http://en.wikipedia.org/wiki/Model-view-controller>.

reachability.

- The view content: it specifies what ViewContainers actually contain, in terms of ViewComponents, i.e., elements for content display and data entry. ViewComponents that display content are further characterized by a ContentBinding, which expresses where the published content comes from.
- The events: they are the occurrences that affect the state of the user interface, which can be produced by the user's interaction, by the application itself, or by an external system.
- The event transitions: they specify the consequences of an event on the user interface, which can be the change of the ViewContainer, the update of the content on display, the triggering of an action, or a mix of these effects.
- The parameter binding: it clarifies the input-output dependencies between ViewComponents, view containers, and actions.

For the sake of conciseness, IFML condenses all these perspectives within one diagram type only, called **Interaction Flow Diagram**, as opposed to other modeling languages, such as UML, which rely on multiple diagrams for conveying the various facets of an application.

Besides describing the view part of the application, an IFML Interaction Flow Diagram also provides the hooks to connect it with the model and controller parts:

- With respect to the controller, IFML represents the effects of the user's interactions; it defines the events produced in the view and the course of action taken by the controller in response to them, such as triggering a business component and updating the view.
- With respect to the model, IFML describes the data binding between the interface elements and the objects that embody the state of the application, as well the actions that are triggered by the user's interactions.

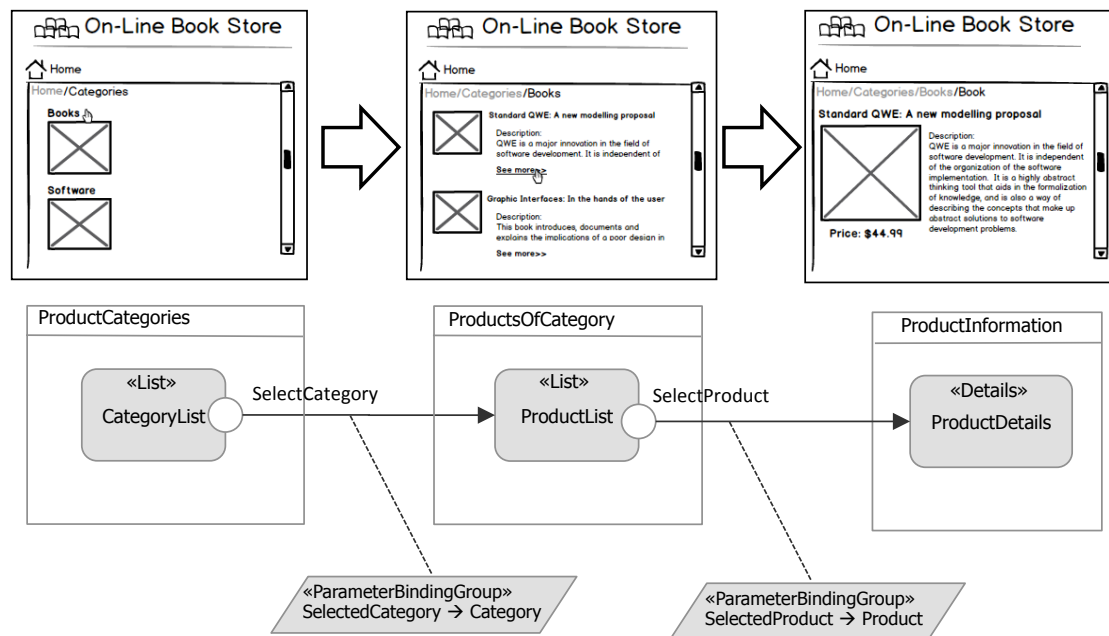


Figure 47- example of interface and its IFML specification

Figure 47 shows as an initial example the IFML model of a simple interface: the view structure consists of three ViewContainers ("ProductCategories", "ProductOfCategory", and "ProductInformation"), which reflect the top-level organization of the GUI in three distinct pages. The model shows the content of each ViewContainer: for example, the "ProductCategories" ViewContainer comprises one ViewComponent called "CategoryList". This notation represents the content of the respective page in the GUI, i.e., a list of product categories. Events are represented in IFML as circles: the "SelectCategory" event specifies that the "CategoryList" component is interactive. In fact, in the GUI the user can select one of

the categories to access the list its products. The effect of the “SelectCategory” event is represented by the arrow emanating from it (called InteractionFlow in IFML): it specifies that the triggering of the event causes the display of the “ProductOfCategory” ViewContainer and the rendering of its “ProductList” ViewComponent, i.e., the list of products *of the selected category*. The input-output dependency between the “CategoryList” and the “ProductList” ViewComponents is represented as a parameter binding (the IFML ParameterBindingGroup element in Figure 47): the value of the “SelectedCategory” parameter, which denotes the object selected by the user in the “CategoryList” ViewComponent, is associated with the value of the input parameter “Category”, which is requested for the computation of the “ProductList” ViewComponent.

8.1.2 Overview of IFML main concepts

An IFML diagram consists of one or more top-level *ViewContainers*, i.e., interface elements that comprise components for displaying content and supporting interactions.

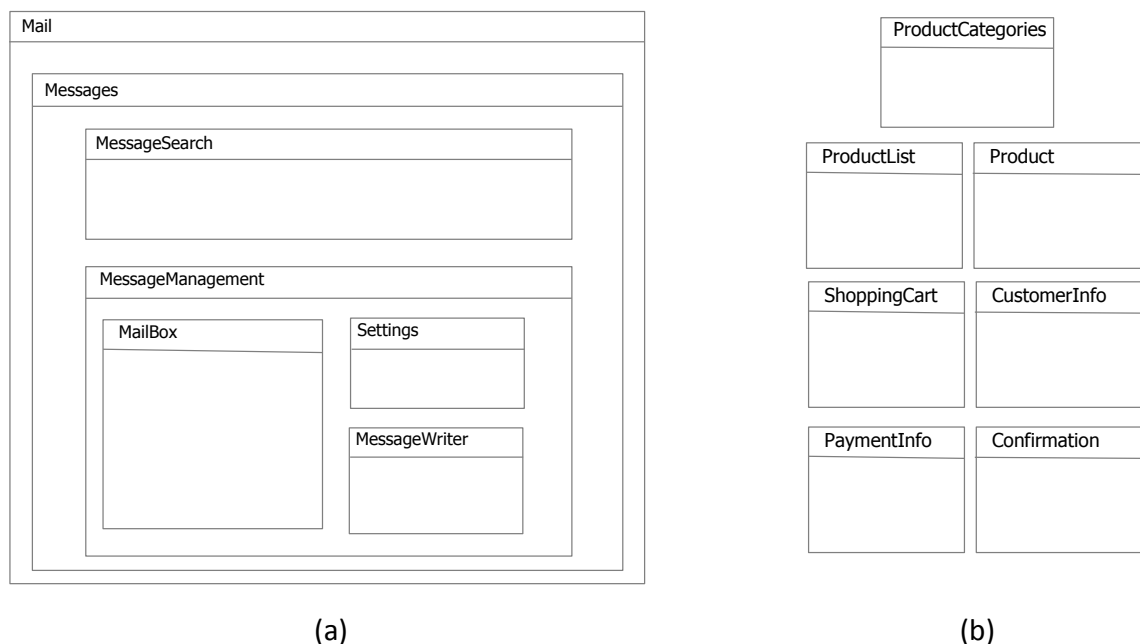


Figure 48: example of different top-level interface structures

Figure 48 contrasts two different organizations of the GUI: a mail desktop or rich internet application (a) consists of a top-level container with embedded sub-containers at different levels; an e-commerce web site (b) organizes the user interface into different independent view containers corresponding to page templates.

Each view container can be internally structured in a *hierarchy of sub-containers*. For example, in a desktop or rich internet application, the main window can contain multiple tabbed frames, which in turn may contain several nested panes. The child view containers nested within a parent view container can be displayed simultaneously (e.g., an object pane and a property pane) or in *mutual exclusion* (e.g., two alternative tabs). In case of mutually exclusive (XOR) containers one could be the *default container*, displayed by default when the parent container is accessed. The meaning of a container can be specified more precisely, by adding a stereotype to the general-purpose construct. For instance, a ViewContainer can be tagged as «window», as in the case of the “Mail” ViewContainer in Figure 49, to hint at the nature of its expected implementation.



Figure 49: example of mutually exclusive sub-containers

In Figure 49, the “Mail” top-level container comprises two sub-containers, displayed alternatively: one for messages and one for contacts. When the top level container is accessed, by default the interface displays the “Messages” ViewContainer.

A ViewContainer can contain *ViewComponents*, which denote the publication of content (e.g., a list of objects) or the input of data (e.g., entry forms).

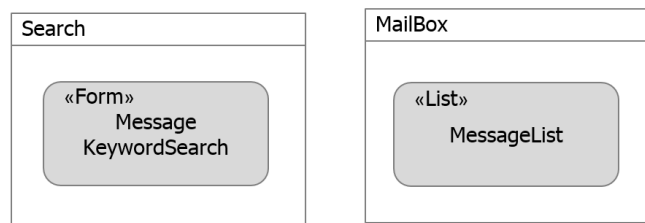


Figure 50 - example of ViewComponents within view containers

Figure 50 shows the notation for embedding ViewComponents within ViewContainers: the “Search” ViewContainer comprises a “MessageKeywordSearch” ViewComponent that represents a form for searching; the “MailBox” ViewContainer comprises a “MessageList” ViewComponent that denotes a list of objects.

A ViewComponent can have *input and output parameters*. For example, a ViewComponent that shows the details of an object has an input parameter corresponding to the identifier of the object to display; a data entry form exposes as output parameters the values submitted by the user; and a list of items exports as output parameter the item selected by the user.

A ViewContainer and a ViewComponent can be associated with *events*, to express that they support the user’s interaction. For example, a ViewComponent can represent: a list associated with an event for selecting one or more items, a form associated with an event for input submission, or an image gallery associated with an event for scrolling though the gallery. IFML events are mapped to *interactors*² in the implemented application. The way in which such interactors are rendered depends on the specific platform for which the application is deployed and is not captured by IFML, but is rather delegated to transformation rules from Platform-Independent Model (PIM) to Platform-Specific Model (PSM). For example, the scrolling of an image gallery may be implemented as a link in an HTML

² By interactor we mean any interface widget that supports the user’s interaction, such as a button, a link, a check box, and so on.

application and as a swipe gesture handler in a mobile phone application.

The effect of an event is represented by an *interaction flow*, which connects the event to the ViewContainer or ViewComponent affected by the event. For example, in an HTML web application the event produced by the selection of one item from a list may cause the display of a new page with the details of the selected object. This effect is represented by an interaction flow connecting the event associated with the list component in a top-level ViewContainer (the web page) with the ViewComponent representing the object detail, positioned in a different ViewContainer (the target web page). The interaction flow expresses a change of state of the user interface: the occurrence of the event causes a transition from a source to a target web page.

For example, in Figure 51 the “MailBoxList” ViewComponent shows the list of available mailboxes and is associated with the “MailBoxSelection” event, whereby the user can open the “MailBox” ViewContainer and access the messages of the mailbox selected in the “MessageList” ViewComponent .

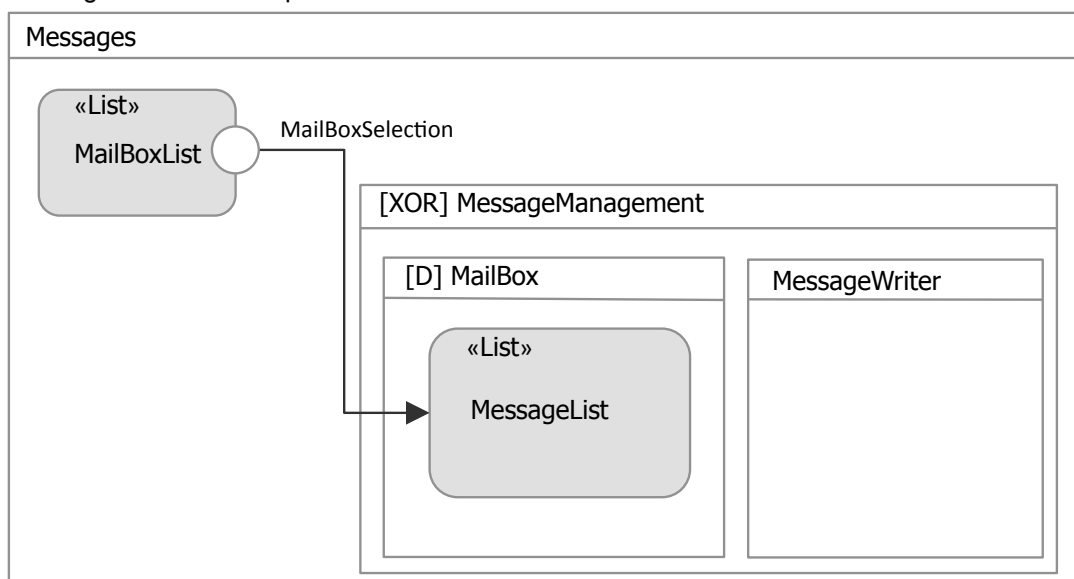


Figure 51 - Example of interaction flow between ViewComponents

An event can also cause the triggering of an action, which is executed prior to updating the state of the user interface; the effect of an event firing an action is represented by an interaction flow connecting the event to an action symbol, consisting of a hexagon. For example, in a mail management application the user can select several messages from a list to delete them; the selection event triggers a delete action, after which the ViewContainer with the updated list is displayed again. The result of action execution is represented by an interaction flow that connects the action to the ViewContainer or ViewComponent affected by it.

In Figure 52, The “Message toolbar” ViewContainer is associated with the events for deleting, archiving and reporting mail messages. Such events are connected by a flow to an action symbol (a labelled hexagonal icon), which represents the business operation. The outgoing flow of the action points to the ViewContainer displayed after the action is executed; if the outgoing flow of an action is omitted, this means that the same ViewContainer wherefrom the action has been activated remains in view (as illustrated for the “Archive” and “Report” actions in Figure 52).

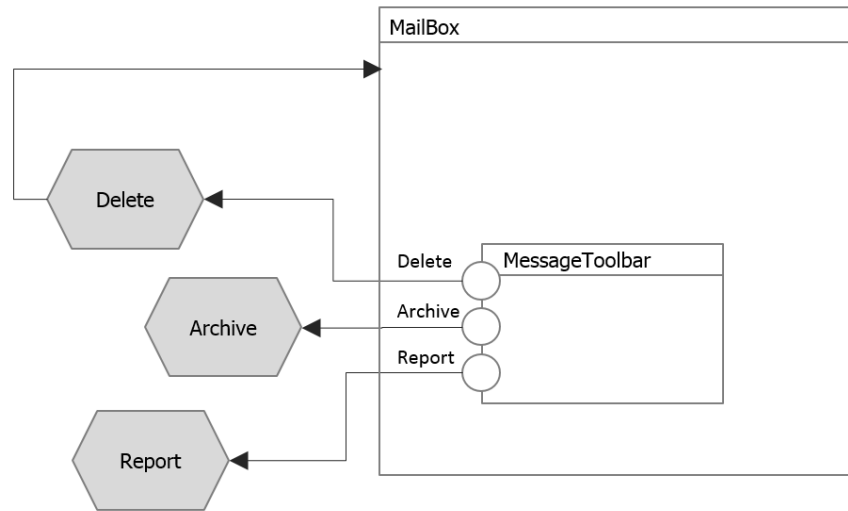


Figure 52: Example of events triggering business actions

The model of Figure 52 does not express the objects on which the business actions operate. Such an *input-output dependency* between view elements (ViewContainers and ViewComponents) or between view elements and actions requires the specification of *parameter bindings* associated with interaction flows. More specifically, two kinds of interaction flows can host parameter bindings: *navigation flows*, which represent navigation between view elements, and *data flows*, which express data transfer only, not produced by the user's interaction. Parameter binding rules are represented by annotations attached to navigation and data flows, as shown in Figure 53.

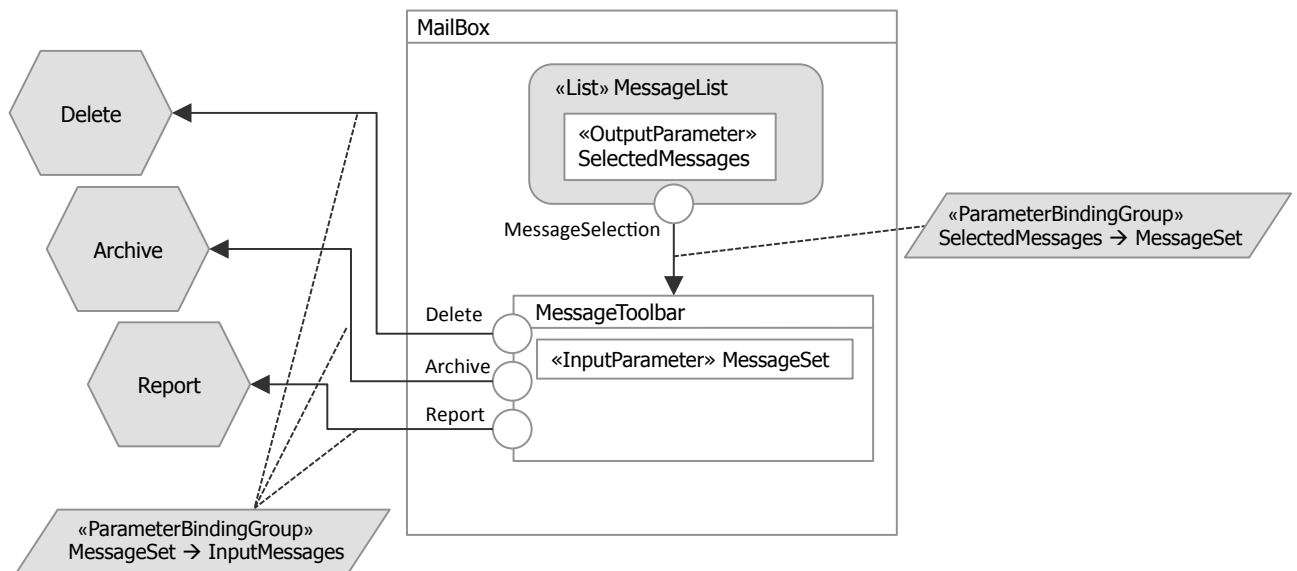


Figure 53: example of parameter bindings used for expressing input-output dependencies

In Figure 53, the “MessageToolbar” ViewContainer has an input parameter “MessageSet”; its value is set to the messages selected from the “MessageList” ViewComponent, when the user produces the “MessageSelection” event. Another parameter binding rule is associated with the Delete, Archive and Report events: the value of the “MessageSet” parameter is bound to the “InputMessages” parameter of the triggered action.

8.1.3 Role of IFML in the development process

The development of interactive applications is typically managed with agile approaches, which traverse several cycles of “problem discovery” / “design refinement” / “implementation”. Each iteration of the development process generates a prototype or a partial version of the system. Such an incremental lifecycle is particularly appropriate for modern Web and mobile applications, which must be deployed quickly and change frequently during their lifetime to adapt to the user’s requirements. Figure 54 schematizes a possible development process and positions IFML within the flow of activities.

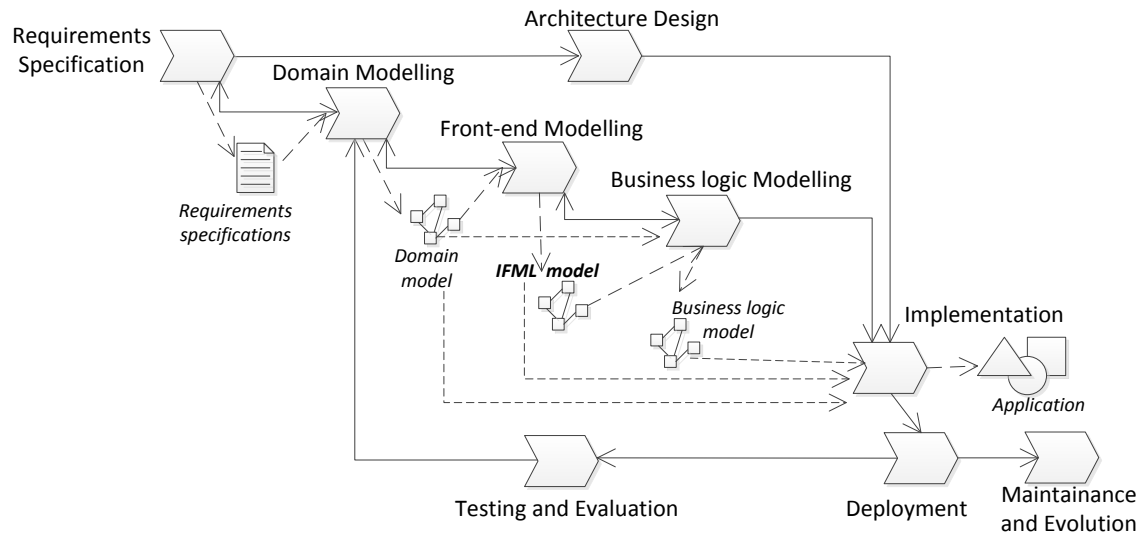


Figure 54 - role of IFML in the development process of an interactive application

8.1.4 A complete example

As a conclusion to this brief introduction of IFML and before the details of the complete IFML specifications of the SmarH2O platform front-end, we present a simple, yet complete, example. The application is an online store, where the user can browse products, such as books, music and software, and add products to his shopping cart, as shown by the UML use case diagram of Figure 55

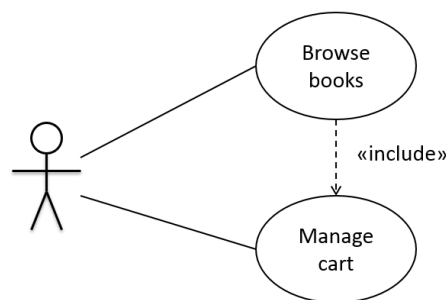


Figure 55 - use cases of the Bookstore application

The application has a Web front-end; in the “Browse books” use case, the user accesses a home page, which contains a list of product categories. Clicking on a product category, such as Books, leads to a page displaying the summary data about all the items of that category; clicking on a “See more” associated with one item’s summary opens a page where the full details of the selected object are presented. Figure 56 shows the mock-ups of the application front-end supporting the “Browse books” use case.

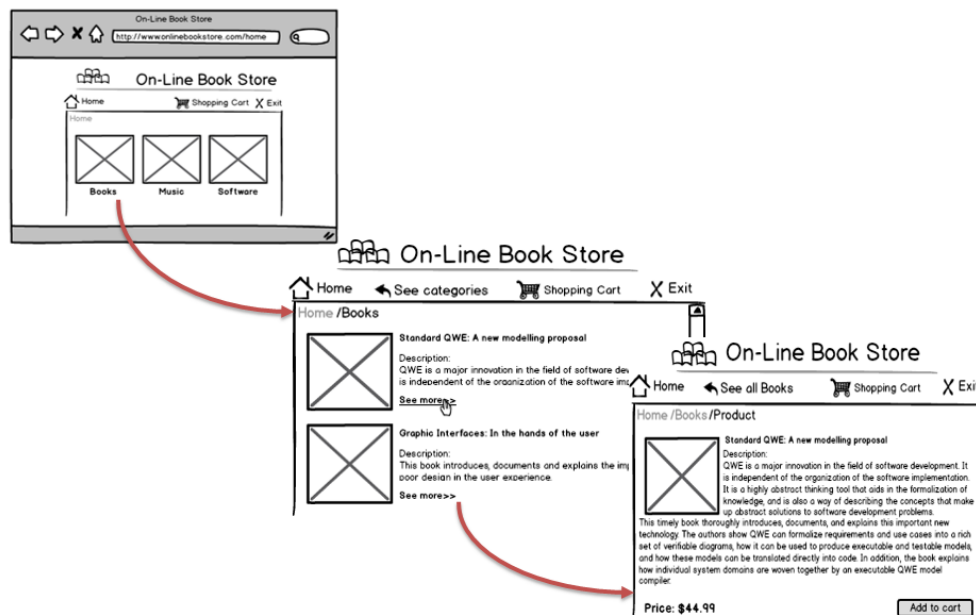


Figure 56 - mockup of the user interface supporting the “Browse books” use case

When looking at the details of an item, the user can press the “Add to cart” button to insert it in his trolley; a modal window appears, where the user can insert the quantity of goods he wants to purchase; after submitting the desired quantity, a confirmation pop-up window is presented to acknowledge the addition of the product to the cart. Figure 57 shows the mock-ups of the interface supporting the “Manage cart” use case.

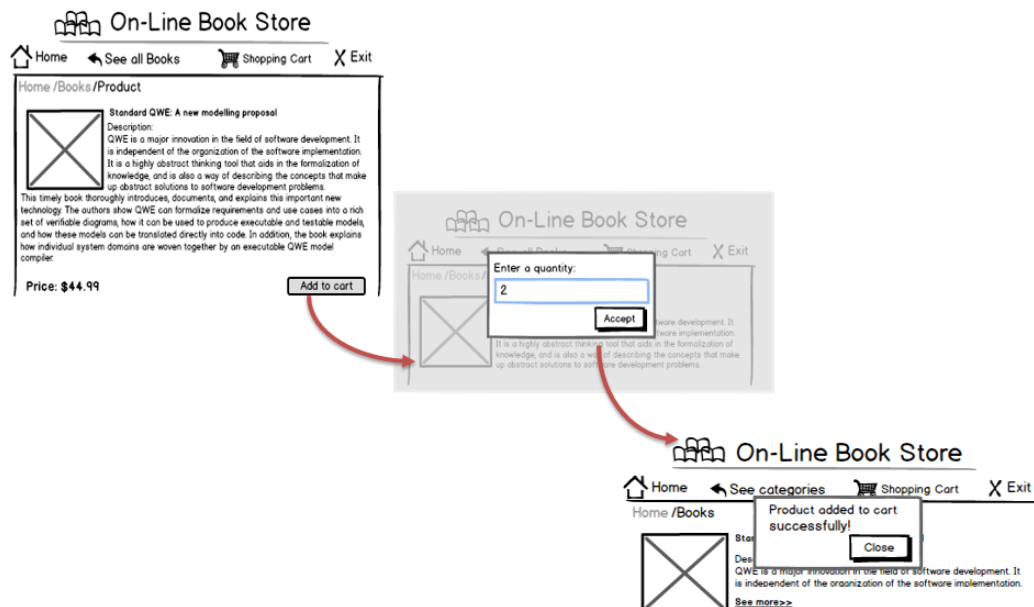


Figure 57 - mockup of the user interface supporting the “Manage cart” use case

The IFML model of the Bookstore application contains the five ViewContainers shown in Figure 58.



Figure 58 - IFML ViewContainers of the Bookstore application

The ViewContainers are annotated with stereotypes (such as H, for “Home”, L for “Landmark” and “Modal” and “Modeless”), which further specify their properties. The ViewContainers definition is refined by specifying the ViewComponents they comprise, as illustrated in Figure 59.

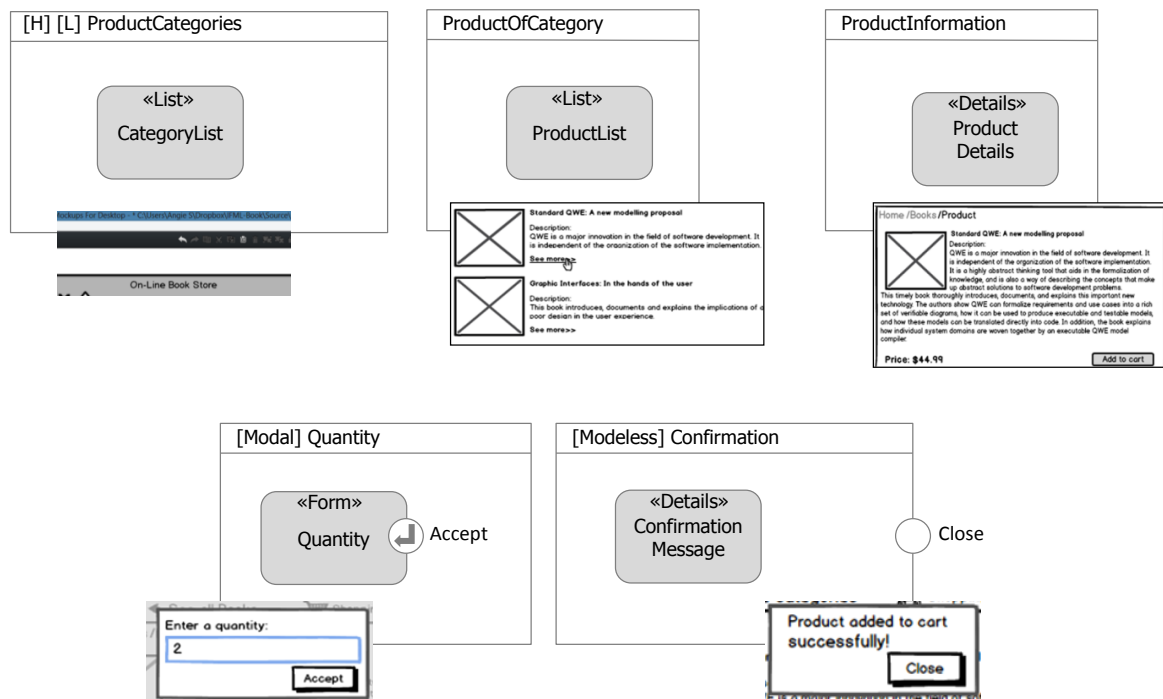


Figure 59 - ViewComponents embedded in IFML ViewContainers, with their mock-up rendition

Interactivity is represented by adding the relevant events and specifying the interaction flows they trigger, completed with the parameter binding between the source and the target components of the interaction flows. The model of Figure 60 shows that the “CategoryList”

ViewComponent supports an interactive event “SelectCategory”, whereby the user can choose a category from the index; as a result, the “ProductOfCategory” page is displayed, and the “ProductList” ViewComponent shows the items corresponding to the chosen category. The input-output dependency between the “CategoryList” and the “ProductList” ViewComponents is represented by the parameter binding group, which associates the “SelectedCategory” output parameter of the source component with the “Category” input parameter of the target component. The same modeling pattern is used to express the interaction for selecting a product from the “ProductList” component and then accessing its data in the “ProductDetails” component.

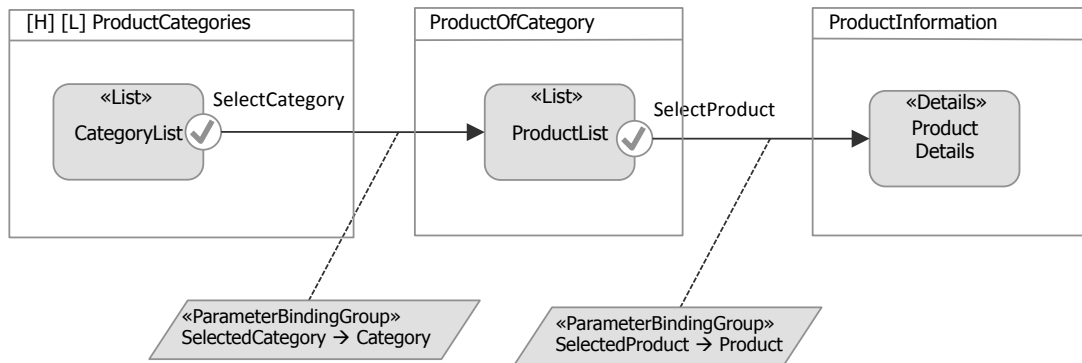


Figure 60 - IFML events and interaction flows of the “Browse Products” use case

Some event may trigger the execution of a piece of business logic; as an example Figure 57 and Figure 61 show the activation of an action for inserting items in the shopping cart: after the user presses the “Add to cart” button associated with the “ProductDetails” component, a modal window appears asking for the quantity of items desired. The quantity submission event triggers the execution of the “Add to cart” action. The quantity value from the Form ViewComponent and the “DisplayedProduct” parameter from the “ProductDetails” ViewComponent are submitted as input parameters to the “Add to cart” action. Once the action is completed, a confirmation window is displayed.

Notice that the binding of the “Quantity” output parameter is associated with an *interaction flow*, which denotes the effect of a submit event that requires the user’s interaction; conversely, the binding of the “DisplayedProduct” parameter is associated with a *data flow*, which merely expresses an input-output dependency, automatically performed by the system and not triggered by a user’s interaction.

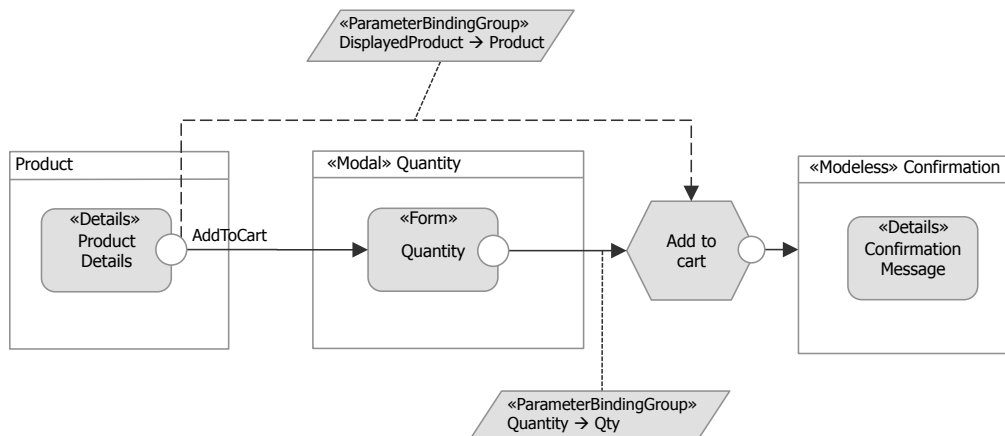


Figure 61 - IFML events and interaction flows of the “Browse Products” use case

8.2 IFML specification of the Customer Portal Basic Version

8.2.1 *Project*

Locale Summary

 English

 French

 Italian

 Spanish

RunningProfiles Summary

 Running Profiles

Property Summary

- ajaxDebug
- app-code

SMTPServer Summary

 smtp server

XsdProvider Summary

 CommunityWS

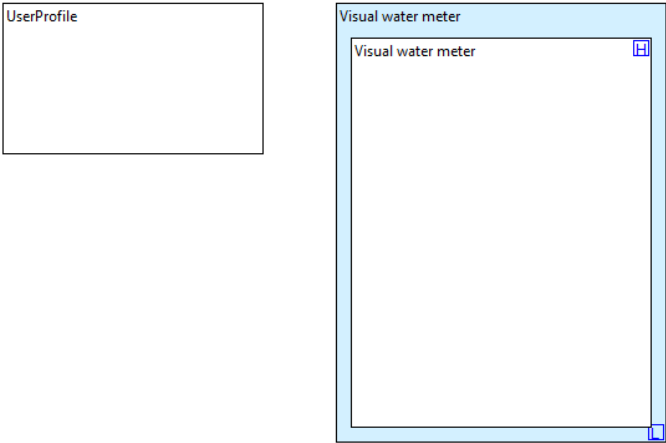
XsdResource Summary

 CommunityWS /  Community

Database Summary

 Domain Model /  CommunityDB

8.2.2 [SiteView] Consumer Portal



Summary Sections

8.2.2..1 Area Summary

 Visual water meter

8.2.2..2 MasterPage Summary

 UserProfile

8.2.2..3 Component Summary

 [ModuleInstanceUnit] Change Language

 [ModuleInstanceUnit] Logout

8.2.2..4 Landmark Summary

 Visual water meter

8.2.2..5 Incoming Flow Summary

→ English	from  UserProfile /  [DataUnit] Welcome	to  [ModuleInstanceUnit] Change Language
→ Italian	from  UserProfile /  [DataUnit] Welcome	to  [ModuleInstanceUnit] Change Language
→ Logout	from  UserProfile /  [DataUnit] Welcome	to  [ModuleInstanceUnit] Logout

8.2.2..6 Outgoing Flow Summary

→ OKFlow8 from  [ModuleInstanceUnit] Change Language to  UserProfile

[Area] Visual water meter



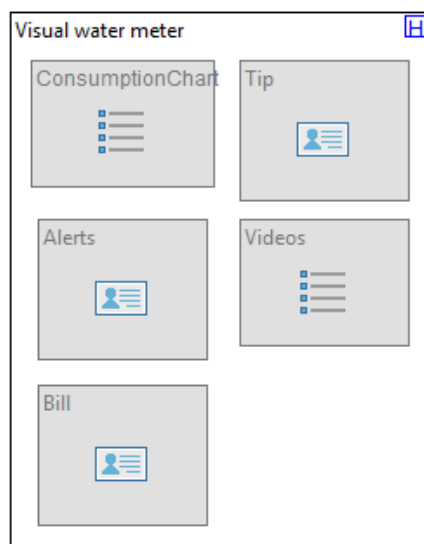
8.2.2..1 Page Summary

 Visual water meter

8.2.2..2 Landmark Summary

 Visual water meter

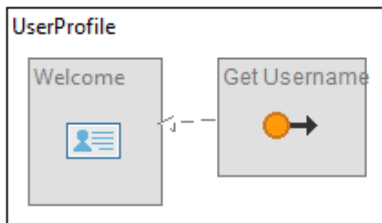
[Page] Visual water meter



Component Summary

-  [DataUnit] Alerts
-  [DataUnit] Bill
-  [PowerIndexUnit] ConsumptionChart
-  [DataUnit] Tip
-  [PowerIndexUnit] Videos

8.2.3 [MasterPage] UserProfile



Component Summary

-  [GetUnit] Get Username
-  [DataUnit] Welcome

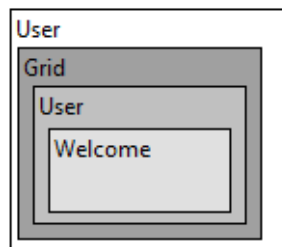
Incoming Flow Summary

→ OKFlow8 from  Consumer Portal /  [ModuleInstanceUnit] Change Language to  UserProfile

Outgoing Flow Summary

→ English from  [DataUnit] Welcome to  Consumer Portal/  [ModuleInstanceUnit] Change Language
 → Italian from  [DataUnit] Welcome to  Consumer Portal/  [ModuleInstanceUnit] Change Language
 → Logout from  [DataUnit] Welcome to  Consumer Portal/  [ModuleInstanceUnit] Logout

8.2.4 [MasterPage] UserProfile (Layout)



8.2.5 Statistics

Structure

 Entity	28(4 derived)(5 volatile)
 Attribute	168(5 derived)
 Attribute per  Entity	6(0 derived)
 Relationship	22(1 derived)
 Relationship per  Entity	0(0 derived)

Navigation

 Site View	4
 Service View	5
 Module View	1
 Context Parameter	13
 Area	12
 Area per  Site View	3
 Page	64
 Page per  Site View	16
 Master Page	4
 Operation Group	43
 Operation Group per    View	4
 Port	19
 Job	2
 Content Module	0
 Operation Module	65
 Hybrid Module	8

Content Units

Content Units	218
Content Components per  Site View	54
Content Components per  Page	3
 Details	26
 Form	33
 Get	25
 Hierarchy	16
 Simple List	5
 Input Port	2

 Module	3
 Checkable List	1
 Multiple Details	6
 Multiple Form	2
 Message	13
 View Component	8
 Output Port	6
 List	26
 Script	4
 Scroller	2
 Selector	40

Operation Units

Operation Units	638
Operation Components per  Site View	159
Operation Components per  Area	53
Operation Components per  Operation Group	14
 Adapter	5
 BLOB Utils Component	2
 Connect	7
 Create	18
 Delete	9
 Disconnect	10
 Error Response	19
 Get	8
 Init Job	2
 Input Port	71
 Is Not Null	35
 Jump	9
 KO Port	52
 Login	1
 Logout	1
 Loop	4
 Mail	1
 Update	23
 Module	97

 No Op	4
 OK Port	73
 Parameter Collector	9
 Password	1
 Query	1
 Reset	6
 Response	19
 Scale Image Unit	6
 Schedule Job	4
 Script	25
 Selector	70
 Solicit	19
 Strings Function Unit	8
 Switch	13
 Time	6

8.3 IFML specification of the Customer Portal Advanced Version

8.3.1 *Project*

Locale Summary

-  English
-  French
-  Italian
-  Spanish

RunningProfiles Summary

-  Running Profiles

Property Summary

- ajaxDebug
- app-code

SMTPServer Summary

-  smtp server

XsdProvider Summary

-  CommunityWS

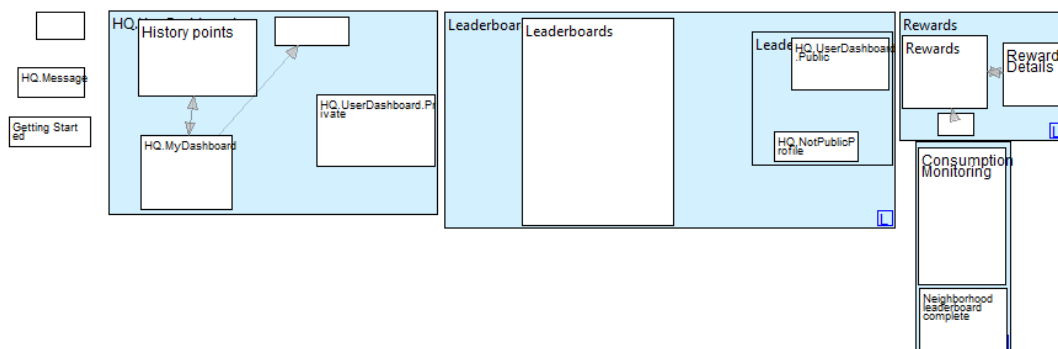
XsdResource Summary

 CommunityWS /  Community

Database Summary

 Domain Model /  CommunityDB

8.3.2 [SiteView] Private



Summary Sections

8.3.2..1 Area Summary

-  Gamified Water Meter
-  HQ.UserDashboard
-  Leaderboards
-  Rewards

8.3.2..2 Page Summary

-  Getting Started
-  HQ.Message

8.3.2..3 MasterPage Summary

-  UserProfile

8.3.2..4 Component Summary

-  [ModuleInstanceUnit] Change Language
-  [ParameterCollectorUnit] GenericMessage

 [ParameterCollectorUnit] Legend

 [ModuleInstanceUnit] Logout

8.3.2..5 Landmark Summary

 Gamified Water Meter

 Leaderboards

 Rewards

8.3.2..6 Incoming Flow Summary

→ English from  UserProfile /  [DataUnit] Welcome to  [ModuleInstanceUnit] Change Language

→ Italian from  UserProfile /  [DataUnit] Welcome to  [ModuleInstanceUnit] Change Language

→ Logout from  UserProfile /  [DataUnit] Welcome to  [ModuleInstanceUnit] Logout

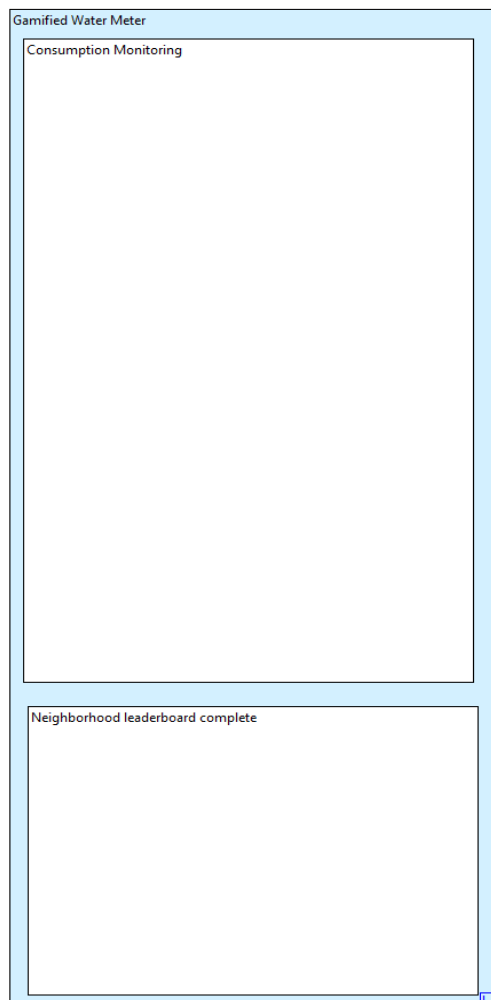
8.3.2..7 Outgoing Flow Summary

→ OKFlow8 from  [ModuleInstanceUnit] Change Language to  UserProfile

→ OKLink47 from  [ParameterCollectorUnit] Legend to  Getting Started

→ OKLink71 from  [ParameterCollectorUnit] GenericMessage to  HQ.Message /  [DataUnit] Generic Message

[Area] Gamified Water Meter



8.3.2..1 Page Summary

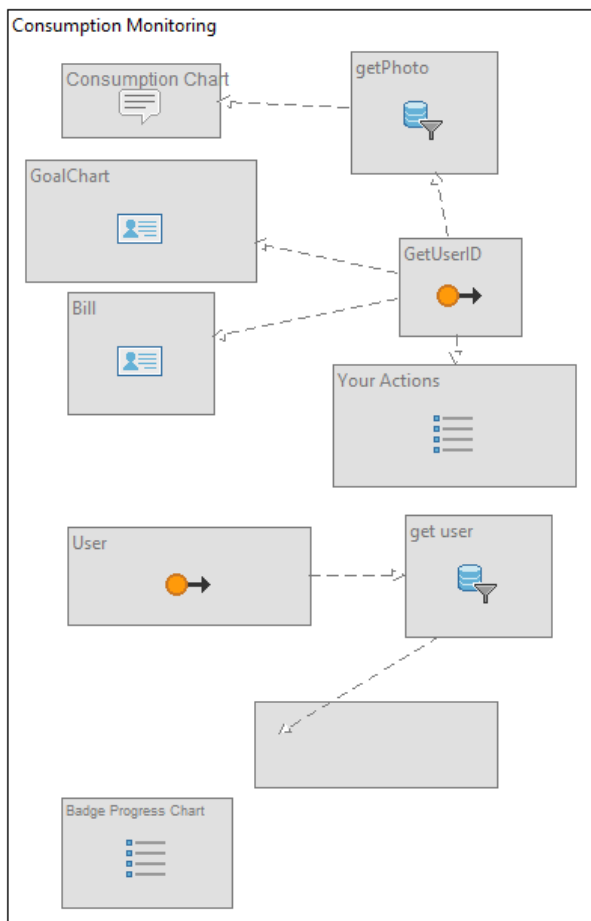
 Consumption Monitoring

 Neighborhood leaderboard complete

8.3.2..2 Landmark Summary

 Gamified Water Meter

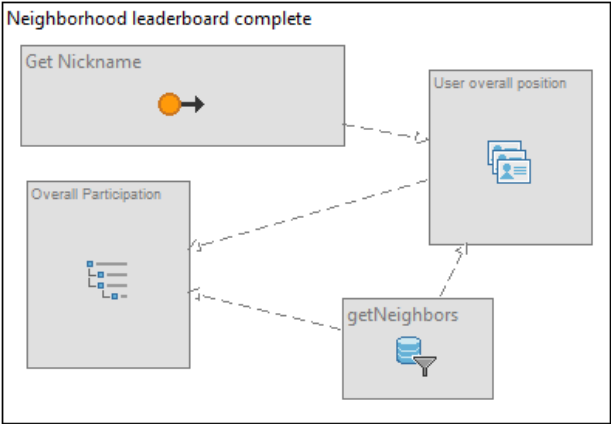
Consumption Monitoring



8.3.2..1 Component Summary

- [PowerIndexUnit] Badge Progress Chart
- [DataUnit] Bill
- [MultiMessageUnit] Consumption Chart
- [ModuleInstanceUnit] DashboardNeigh
- [SelectorUnit] get user
- [SelectorUnit] getPhoto
- [GetUnit] GetUserID
- [DataUnit] GoalChart
- [GetUnit] User
- [PowerIndexUnit] Your Actions

[Page] Neighborhood leaderboard complete



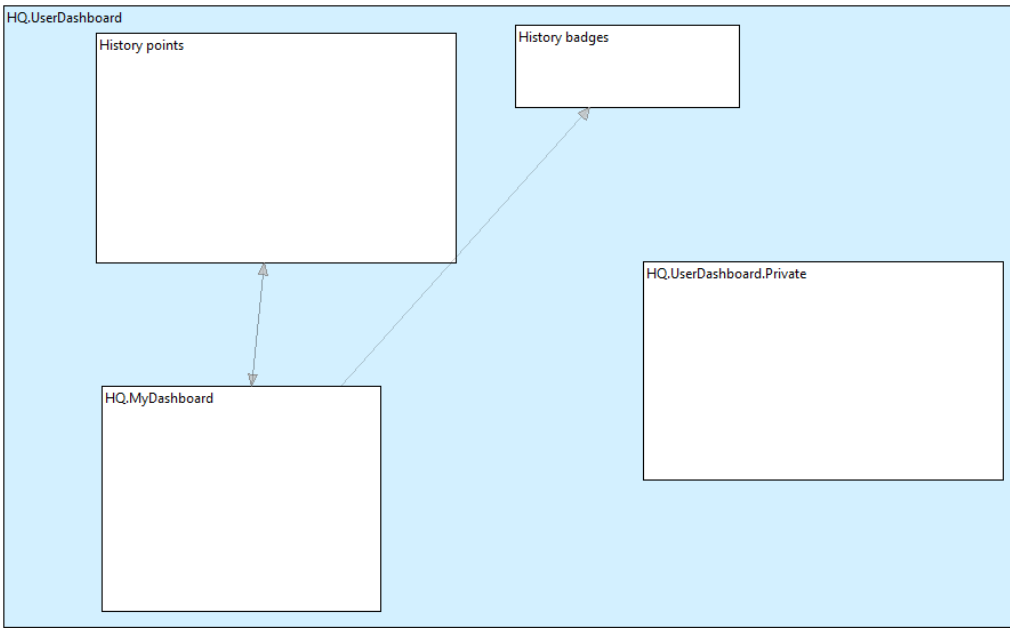
8.3.2..1 Component Summary

- [GetUnit] Get Nickname
- [SelectorUnit] getNeighbors
- [HierarchicalIndexUnit] Overall Participation
- [MultiDataUnit] User overall position

8.3.2..2 Internal Flow Summary

→ Go to participation from [MultiDataUnit] User overall position to [HierarchicalIndexUnit] Overall Participation

[Area] HQ.UserDashboard



8.3.2..1 Page Summary

- History badges
- History points
- HQ.MyDashboard
- HQ.UserDashboard.Private

8.3.2..2 Component Summary

- [ModuleInstanceUnit] Access Dashboard
- [JumpUnit] go to public dashboard
- [ParameterCollectorUnit] HQ.UserDashboard
- [JumpUnit] Jump To Rewards
- [ParameterCollectorUnit] my dashboard
- [ParameterCollectorUnit] User History Points

8.3.2..3 Incoming Flow Summary

- Link49 from HQ.MyDashboard / [ModuleInstanceUnit] Dashboard to [JumpUnit] go to public dashboard
- See Your Rewards from History points / [DataUnit] Text Chunk to [JumpUnit] Jump To Rewards

8.3.2..4 Outgoing Flow Summary

- OKFlow14 from [ParameterCollectorUnit] my dashboard to HQ.MyDashboard
- OKFlow39 from [ParameterCollectorUnit] User History Points to History points
- OKFlow58 from [ModuleInstanceUnit] Access Dashboard to HQ.UserDashboard.Private

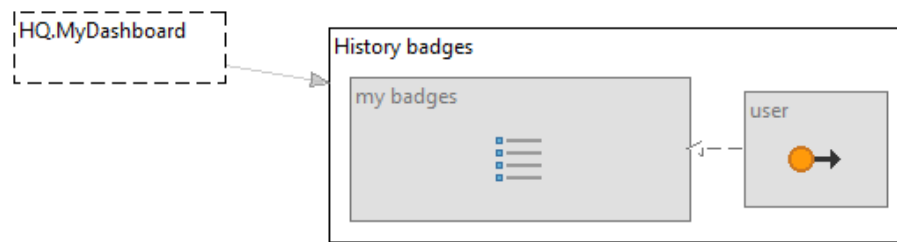
8.3.2..5 Internal Flow Summary

- OKFlow59 from [ModuleInstanceUnit] Access Dashboard to [ParameterCollectorUnit] my dashboard
- OKLink102 from [ParameterCollectorUnit] HQ.UserDashboard to [ModuleInstanceUnit] Access Dashboard

8.3.2..6 Property Summary

- app-code

[Page] History badges



8.3.2..1 Component Summary

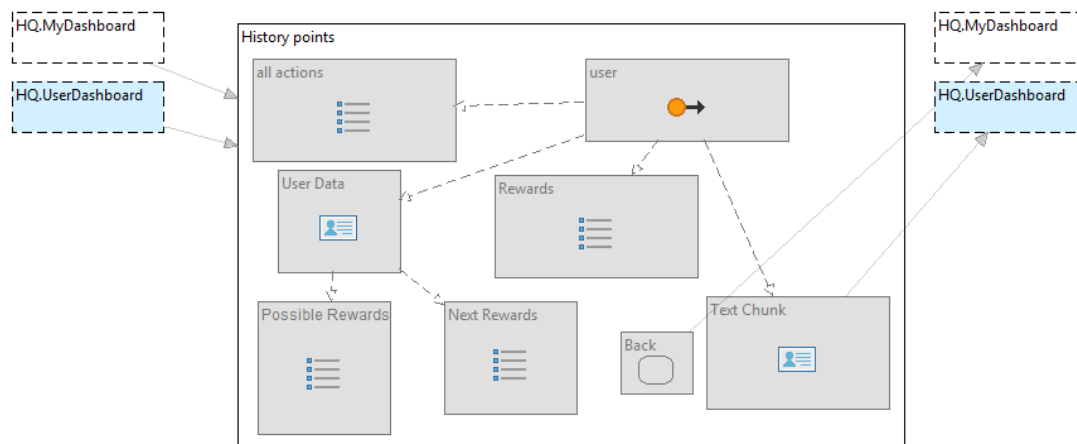
☰ [PowerIndexUnit] my badges

➡ [GetUnit] user

8.3.2..2 Incoming Flow Summary

→ HQ.YourBadges from HQ.MyDashboard Dashboard / [ModuleInstanceUnit] to History badges

[Page] History points



8.3.2..1 Component Summary

☰ [PowerIndexUnit] all actions

◻ [NoOpContentUnit] Back

☰ [PowerIndexUnit] Next Rewards

☰ [PowerIndexUnit] Possible Rewards

☰ [PowerIndexUnit] Rewards

👤 [DataUnit] Text Chunk

➡ [GetUnit] user

👤 [DataUnit] User Data

8.3.2..2 Incoming Flow Summary

→ History actions from HQ.MyDashboard / [ModuleInstanceUnit] Dashboard to History points

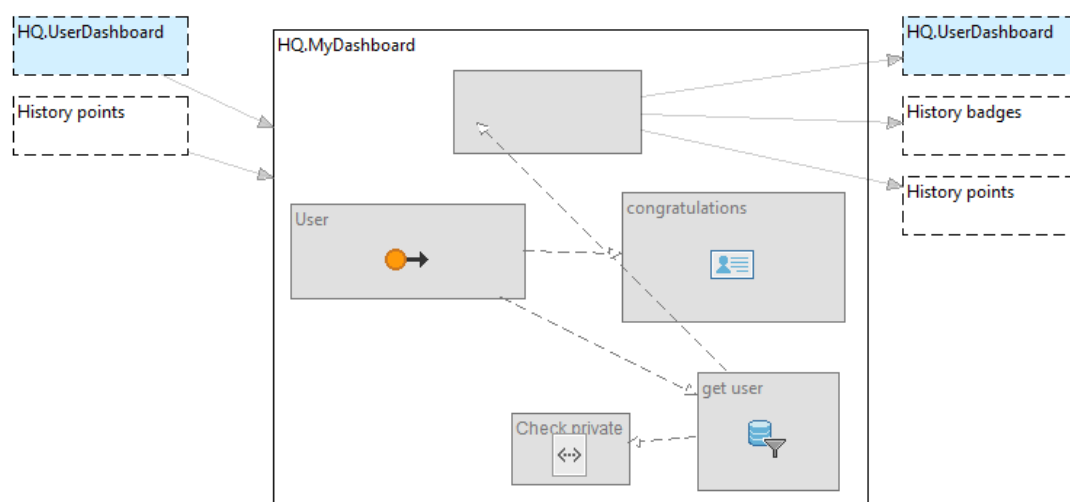
→ OKFlow39 from HQ.UserDashboard / [ParameterCollectorUnit] User History Points to History points

8.3.2..3 Outgoing Flow Summary

→ Go Back from [NoOpContentUnit] Back to HQ.MyDashboard

→ See Your Rewards from [DataUnit] Text Chunk to HQ.UserDashboard / [JumpUnit] Jump To Rewards

[Page] HQ.MyDashboard



8.3.2..1 ConditionExpression Summary

AreYouLogged?

ExsistComponent?

ExsistPost?

8.3.2..2 Component Summary

[ScriptUnit] Check private

[DataUnit] congratulations

[ModuleInstanceUnit] Dashboard

[SelectorUnit] get user

[GetUnit] User

8.3.2..3 Variable Summary

Component

NickDashboard

participationPosition

participationScore

- ◆ Post
- ◆ reputationPosition
- ◆ reputationScore
- ◆ User
- ◆ userid

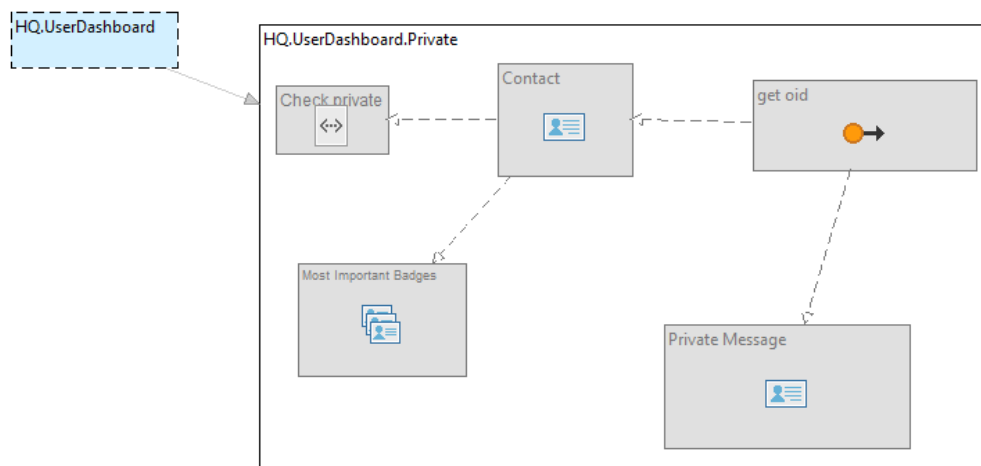
8.3.2..4 Incoming Flow Summary

- Go Back from History points / [NoOpContentUnit] Back to HQ.MyDashboard
- OKFlow14 from HQ.UserDashboard dashboard / [ParameterCollectorUnit] my to HQ.MyDashboard

8.3.2..5 Outgoing Flow Summary

- History actions from [ModuleInstanceUnit] Dashboard to History points
- HQ.YourBadges from [ModuleInstanceUnit] Dashboard to History badges
- Link49 from [ModuleInstanceUnit] Dashboard to HQ.UserDashboard / [JumpUnit] go to public dashboard

[Page] HQ.UserDashboard.Private



8.3.2..1 ConditionExpression Summary

- ExistComponent?
- ExistTopic?

8.3.2..2 Component Summary

- [ScriptUnit] Check private
- [DataUnit] Contact
- [GetUnit] get oid
- [MultiDataUnit] Most Important Badges
- [DataUnit] Private Message

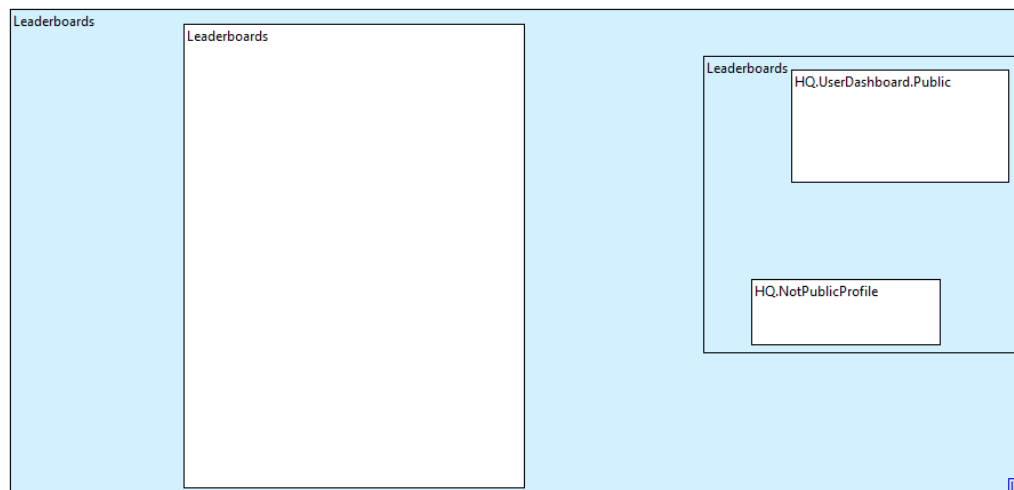
8.3.2..3 Variable Summary

- ◆ Component
- ◆ Topic
- ◆ User

8.3.2..4 Incoming Flow Summary

→ OKFlow58 from  HQ.UserDashboard /  [ModuleInstanceUnit] Access to  HQ.UserDashboard.Private Dashboard

[Area] Leaderboards



8.3.2..1 Area Summary

 Leaderboards

8.3.2..2 Page Summary

 Leaderboards

8.3.2..3 Component Summary

- ◆ [JumpUnit] go to dashboard
- ◆ [ModuleInstanceUnit] Go To User LeaderBoard Position
- ◆ [ModuleInstanceUnit] OnChange NoOp
- ◆ [ModuleInstanceUnit] OnChange NoOp
- ◆ [ParameterCollectorUnit] Public Dashboard
- ◆ [ParameterCollectorUnit] UserRanking

8.3.2..4 Landmark Summary

 Leaderboards

8.3.2..5 Incoming Flow Summary

→ Clear	from  Leaderboards /  [EntryUnit] Monthly Form	to  [ModuleInstanceUnit] OnChange NoOp
→ Link1	from  Leaderboards /  [HierarchicalIndexUnit] Overall Participation	to  [JumpUnit] go to dashboard
→ Link145	from  HQ.UserDashboard.Public [ModuleInstanceUnit] Dashboard / 	to  [ParameterCollectorUnit] Public Dashboard
→ Link5	from  Leaderboards /  [HierarchicalIndexUnit] Monthly Participation	to  [JumpUnit] go to dashboard
→ Search	from  Leaderboards /  [EntryUnit] Monthly Form	to  [ModuleInstanceUnit] OnChange NoOp
→ Set	from  Leaderboards /  [EntryUnit] Monthly Form	to  [ModuleInstanceUnit] OnChange NoOp

8.3.2..6 Outgoing Flow Summary

→ OKFlow1	from  [ModuleInstanceUnit] OnChange NoOp	to  Leaderboards
→ OKFlow19	from  [ModuleInstanceUnit] OnChange NoOp	to  Leaderboards
→ OKFlow25	from  [ModuleInstanceUnit] Go To User LeaderBoard Position	to  Leaderboards
→ OKFlow81	from  [ParameterCollectorUnit] Public Dashboard	to  Leaderboards /  [ModuleInstanceUnit] Check Public Dashboard Access

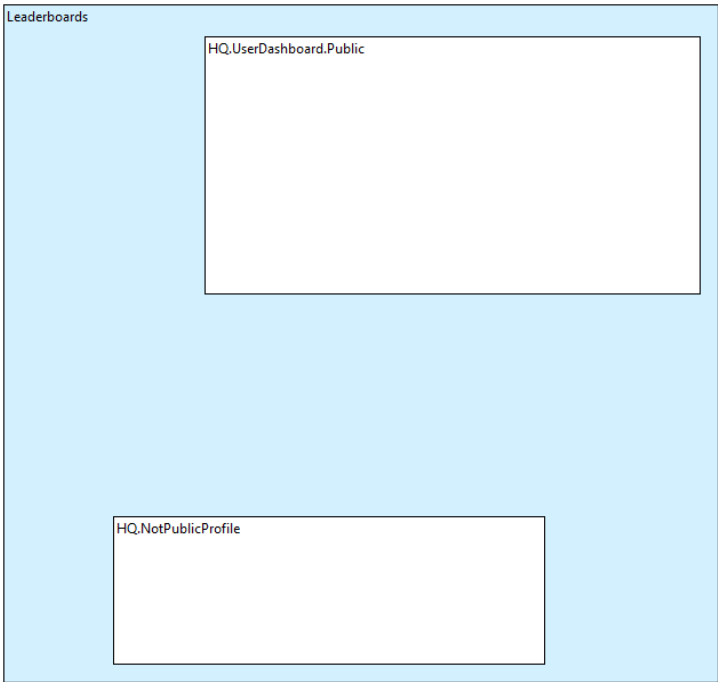
8.3.2..7 Internal Flow Summary

→ OKFlow370	from  [ParameterCollectorUnit] UserRanking	to  [ModuleInstanceUnit] Go To User LeaderBoard Position
-------------	---	---

8.3.2..8 Property Summary

- app-code

[Area] Leaderboards



8.3.2..1 Page Summary

- HQ.NotPublicProfile
- HQ.UserDashboard.Public

8.3.2..1 Component Summary

- [ModuleInstanceUnit] Check Public Dashboard Access
- [JumpUnit] go to my dashboard

8.3.2..2 Incoming Flow Summary

- OKFlow81 from Leaderboards / [ParameterCollectorUnit] to [ModuleInstanceUnit] Check Public Dashboard Access

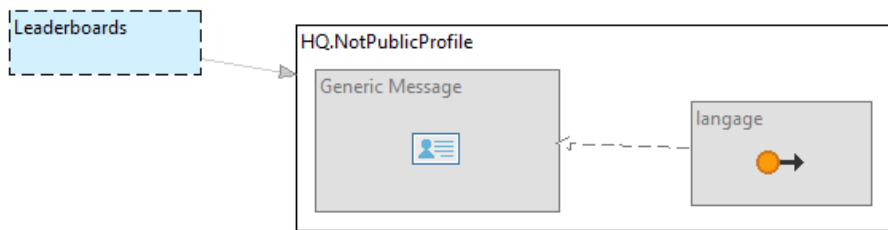
8.3.2..3 Outgoing Flow Summary

- OKFlow49 from [ModuleInstanceUnit] Check Public Dashboard Access to HQ.UserDashboard.Public
- KOFlow10 from [ModuleInstanceUnit] Check Public Dashboard Access to HQ.NotPublicProfile

8.3.2..4 Internal Flow Summary

- OKFlow52 from [ModuleInstanceUnit] Check Public Dashboard Access to [JumpUnit] go to my dashboard

[Page] HQ.NotPublicProfile



8.3.2..1 Component Summary

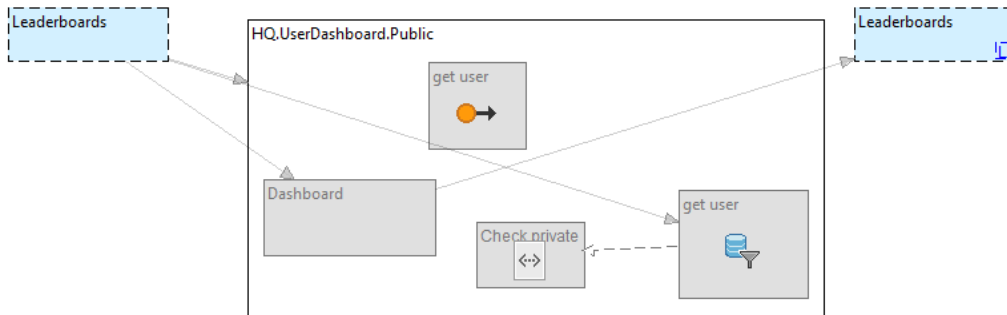
[DataUnit] Generic Message

→ [GetUnit] langage

8.3.2..2 Incoming Flow Summary

→ KOFlow10 from [ModuleInstanceUnit] Leaderboards / [ModuleInstanceUnit] Check Public to HQ.NotPublicProfile Dashboard Access

[Page] HQ.UserDashboard.Public



8.3.2..1 Component Summary

[ScriptUnit] Check private

[ModuleInstanceUnit] Dashboard

[SelectorUnit] get user

→ [GetUnit] get user

8.3.2..2 Variable Summary

◆ Name

◆ NamePageTitle

◆ SurnamePageTitle

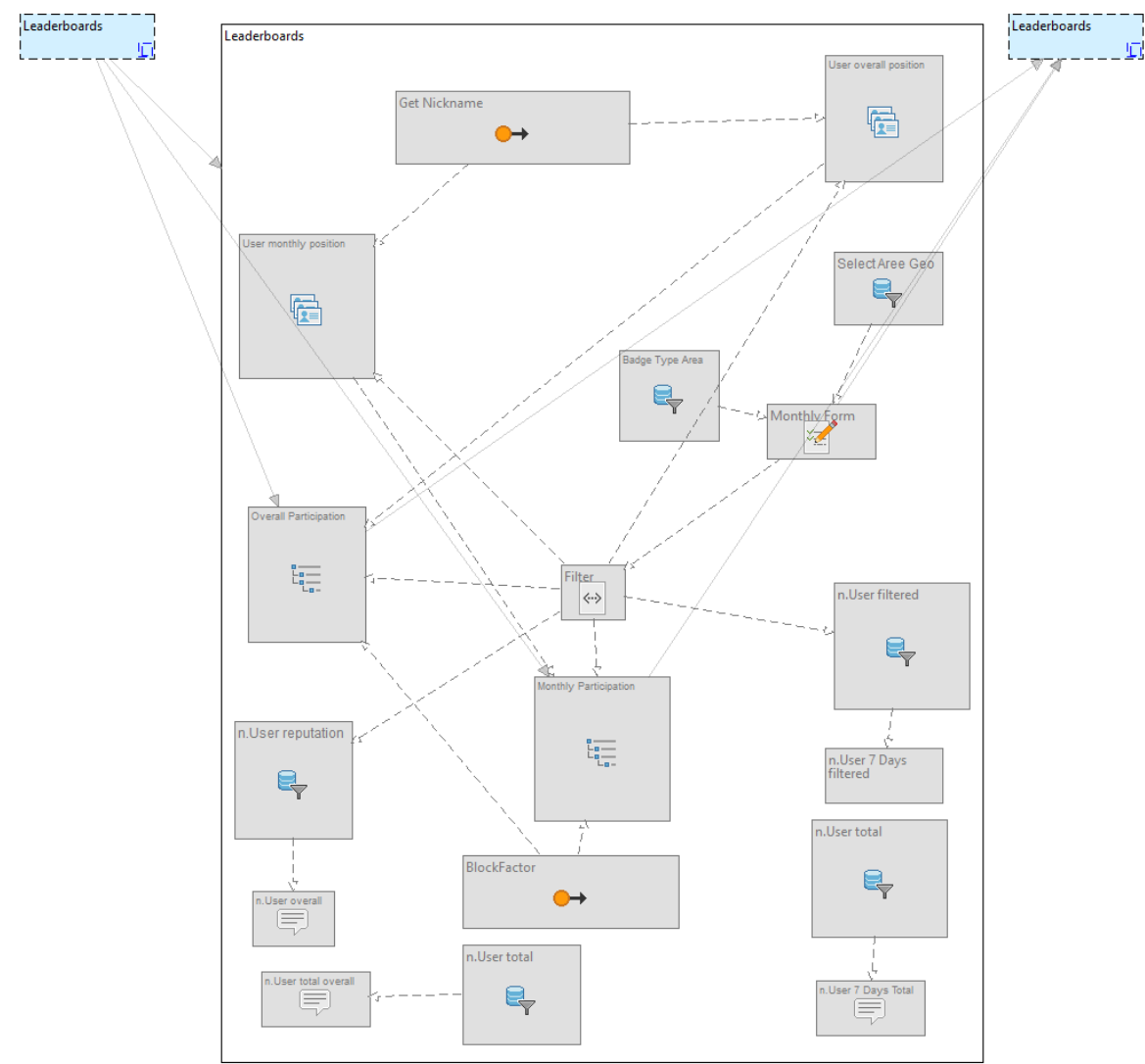
8.3.2..3 Incoming Flow Summary

→ OKFlow49 from [ModuleInstanceUnit] Leaderboards / [ModuleInstanceUnit] Check Public to HQ.UserDashboard.Public Dashboard Access

8.3.2..4 Outgoing Flow Summary

→ Link145 from [ModuleInstanceUnit] Dashboard to Leaderboards / [ParameterCollectorUnit] Public Dashboard

[Page] Leaderboards



8.3.2..1 ConditionExpression Summary

[isPrivate?] isPrivate?

8.3.2..2 Component Summary

[SelectorUnit] Badge Type Area

[GetUnit] BlockFactor

[ScriptUnit] Filter

[GetUnit] Get Nickname

[EntryUnit] Monthly Form

-  [HierarchicalIndexUnit] Monthly Participation
-  [MultiMessageUnit] n.User 7 Days filtered
-  [MultiMessageUnit] n.User 7 Days Total
-  [SelectorUnit] n.User filtered
-  [MultiMessageUnit] n.User overall
-  [SelectorUnit] n.User reputation
-  [SelectorUnit] n.User total
-  [SelectorUnit] n.User total
-  [MultiMessageUnit] n.User total overall
-  [HierarchicalIndexUnit] Overall Participation
-  [SelectorUnit] Select Aree Geo
-  [MultiDataUnit] User monthly position
-  [MultiDataUnit] User overall position

8.3.2..3 Variable Summary

-  privateProfi

8.3.2..4 Incoming Flow Summary

- OKFlow1 from  Leaderboards /  [ModuleInstanceUnit] OnChange NoOp to  Leaderboards
- OKFlow19 from  Leaderboards /  [ModuleInstanceUnit] OnChange NoOp to  Leaderboards
- OKFlow25 from  Leaderboards /  [ModuleInstanceUnit] Go To User LeaderBoard Position to  Leaderboards

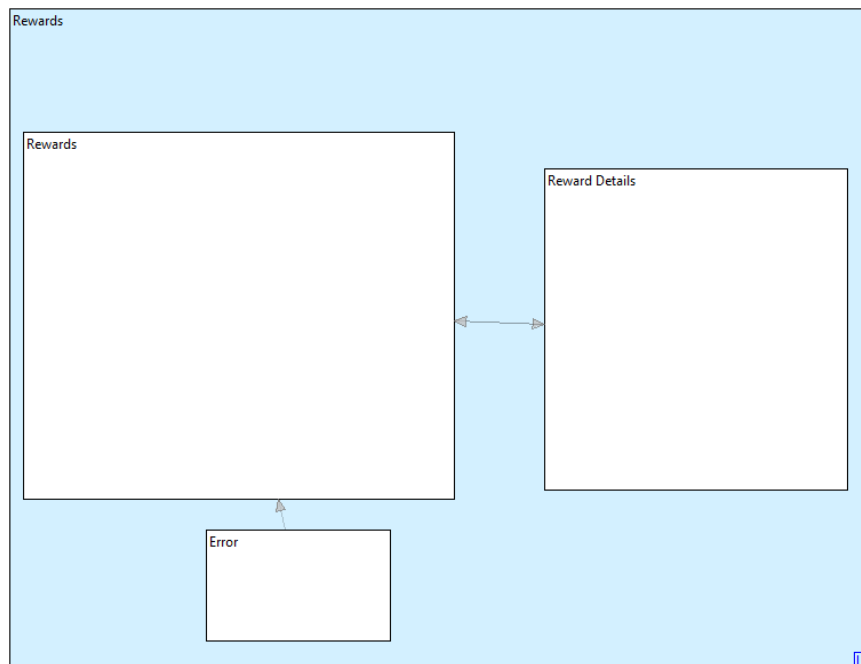
8.3.2..5 Outgoing Flow Summary

- Clear from  [EntryUnit] Monthly Form to  Leaderboards /  [ModuleInstanceUnit] OnChange NoOp
- Link1 from  [HierarchicalIndexUnit] Overall Participation to  Leaderboards /  [JumpUnit] go to dashboard
- Link5 from  [HierarchicalIndexUnit] Monthly Participation to  Leaderboards /  [JumpUnit] go to dashboard
- Search from  [EntryUnit] Monthly Form to  Leaderboards /  [ModuleInstanceUnit] OnChange NoOp
- Set from  [EntryUnit] Monthly Form to  Leaderboards /  [ModuleInstanceUnit] OnChange NoOp

8.3.2..6 Internal Flow Summary

- Go to participation from  [MultiDataUnit] User overall position to  [HierarchicalIndexUnit] Overall Participation
- Go to participation from  [MultiDataUnit] User monthly position to  [HierarchicalIndexUnit] Monthly Participation

[Area] Rewards



8.3.2..1 Page Summary

- Error
- Reward Details
- Rewards

8.3.2..2 Component Summary

- [JumpUnit] Go to user points details
- [ModuleInstanceUnit] Redeem Reward
- [ParameterCollectorUnit] Rewards

8.3.2..3 Landmark Summary

- Rewards

8.3.2..4 Incoming Flow Summary

→ Get Reward from Reward Details / [DataUnit] Reward to [ModuleInstanceUnit] Redeem Reward

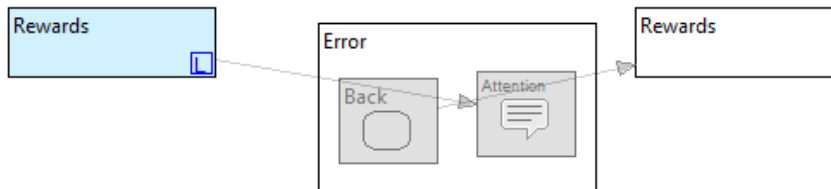
8.3.2..5 Outgoing Flow Summary

→ OKFlow164 from [ParameterCollectorUnit] Rewards to Rewards
 → KOFlow30 from [ModuleInstanceUnit] Redeem Reward to Error / [MultiMessageUnit] Attention

8.3.2..6 Internal Flow Summary

→ OKFlow169 from [ModuleInstanceUnit] Redeem Reward to [JumpUnit] Go to user points details

[Page] Error



8.3.2..1 Component Summary

[MultiMessageUnit] Attention

[NoOpContentUnit] Back

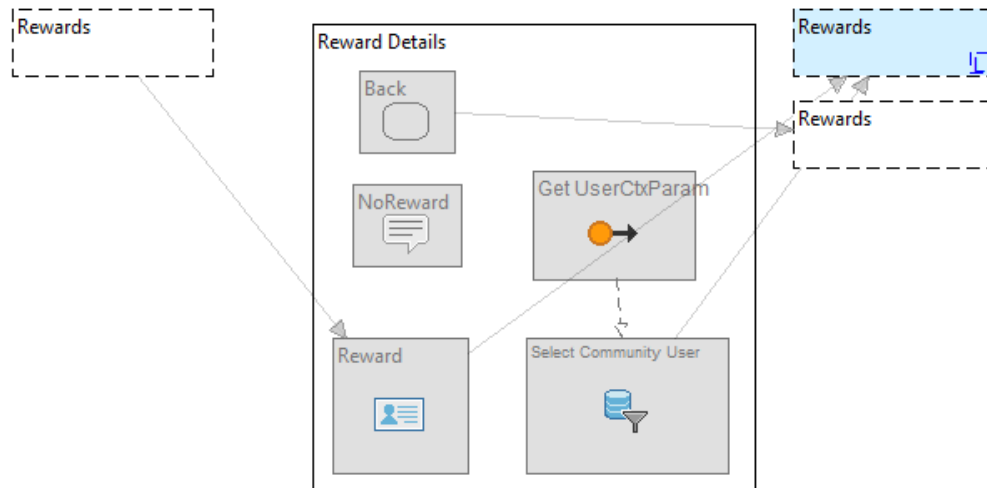
8.3.2..2 Incoming Flow Summary

→ KOFlow30 from Rewards / [ModuleInstanceUnit] Redeem Reward to [MultiMessageUnit] Attention

8.3.2..3 Outgoing Flow Summary

→ Go Back from [NoOpContentUnit] Back to Rewards

[Page] Reward Details



8.3.2..1 ConditionExpression Summary

[isRedeemable]

8.3.2..2 Component Summary

[NoOpContentUnit] Back

→ [GetUnit] Get UserCtxParam

- [MultiMessageUnit] NoReward
- [DataUnit] Reward
- [SelectorUnit] Select Community User

8.3.2..3 Variable Summary

- availableCredits
- neededPoints

8.3.2..4

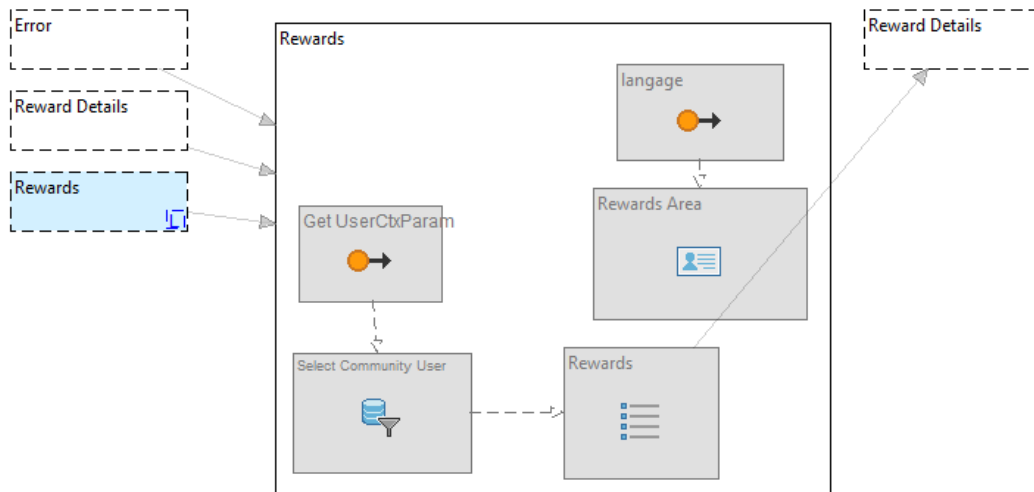
8.3.2..5 Incoming Flow Summary

→ View Reward from [Rewards / [PowerIndexUnit] Rewards] to [DataUnit] Reward

8.3.2..6 Outgoing Flow Summary

→ Get Reward from [DataUnit] Reward to [Rewards / [ModuleInstanceUnit] Redeem Reward]
 → Go Back from [NoOpContentUnit] Back to [Rewards]

[Page] Rewards



8.3.2..1 Component Summary

- [GetUnit] Get UserCtxParam
- [GetUnit] language
- [PowerIndexUnit] Rewards
- [DataUnit] Rewards Area
- [SelectorUnit] Select Community User

8.3.2..2 Incoming Flow Summary

→ Go Back from [Error / [NoOpContentUnit] Back] to [Rewards]

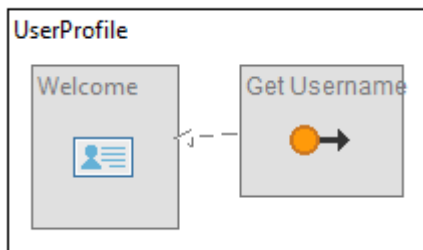
→ Go Back from Reward Details / [NoOpContentUnit] Back to Rewards

→ OKFlow164 from Rewards / [ParameterCollectorUnit] Rewards to Rewards

8.3.2..3 Outgoing Flow Summary

→ View Reward from [PowerIndexUnit] Rewards to Reward Details / [DataUnit] Reward

[MasterPage] UserProfile



8.3.2..1 Component Summary

→ [GetUnit] Get Username

[DataUnit] Welcome

8.3.2..2 Incoming Flow Summary

→ OKFlow8 from Private / [ModuleInstanceUnit] Change Language to UserProfile

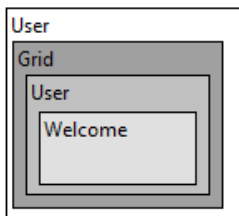
8.3.2..3 Outgoing Flow Summary

→ English from [DataUnit] Welcome to Private / [ModuleInstanceUnit] Change Language

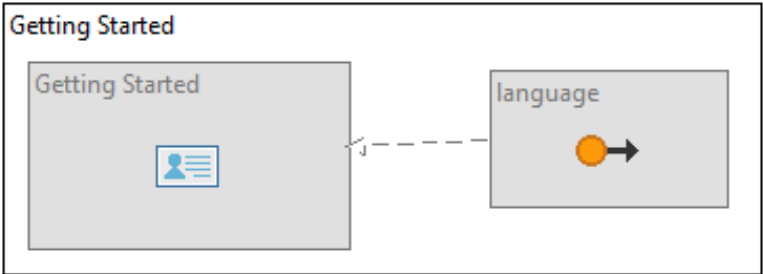
→ Italian from [DataUnit] Welcome to Private / [ModuleInstanceUnit] Change Language

→ Logout from [DataUnit] Welcome to Private / [ModuleInstanceUnit] Logout

[MasterPage] UserProfile (Layout)



[Page] Getting Started



8.3.2..1 Component Summary

[DataUnit] Getting Started

[GetUnit] language

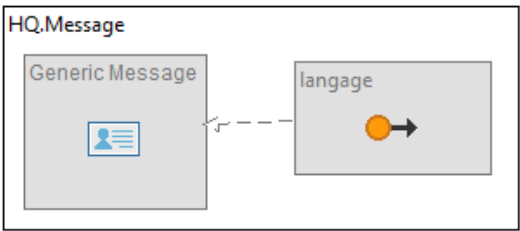
8.3.2..2 Incoming Flow Summary

→ OKLink47 from Private / [ParameterCollectorUnit] Legend to Getting Started

8.3.2..3 Property Summary

- app-code

[Page] HQ.Message



8.3.2..1 Component Summary

[DataUnit] Generic Message

[GetUnit] language

8.3.2..2 Incoming Flow Summary

→ OKLink71 from Private / [ParameterCollectorUnit] to [DataUnit] Generic Message

8.3.3 Statistics

Structure

Entity

28(4 derived)(5 volatile)

 Attribute	168(5 derived)
 Attribute per  Entity	6(0 derived)
 Relationship	22(1 derived)
 Relationship per  Entity	0(0 derived)

Navigation

 Site View	4
 Service View	5
 Module View	1
 Context Parameter	13
 Area	12
 Area per  Site View	3
 Page	64
 Page per  Site View	16
 Master Page	4
 Operation Group	43
 Operation Group per    View	4
 Port	19
 Job	2
 Content Module	0
 Operation Module	65
 Hybrid Module	8

Content Units

Content Units	218
Content Components per  Site View	54
Content Components per  Page	3
 Details	26
 Form	33
 Get	25
 Hierarchy	16
 Simple List	5
 Input Port	2
 Module	3
 Checkable List	1
 Multiple Details	6

 Multiple Form	2
 Message	13
 View Component	8
 Output Port	6
 List	26
 Script	4
 Scroller	2
 Selector	40

Operation Units

Operation Units	638
Operation Components per  Site View	159
Operation Components per  Area	53
Operation Components per  Operation Group	14
 Adapter	5
 BLOB Utils Component	2
 Connect	7
 Create	18
 Delete	9
 Disconnect	10
 Error Response	19
 Get	8
 Init Job	2
 Input Port	71
 Is Not Null	35
 Jump	9
 KO Port	52
 Login	1
 Logout	1
 Loop	4
 Mail	1
 Update	23
 Module	97
 No Op	4
 OK Port	73
 Parameter Collector	9

 Password	1
 Query	1
 Reset	6
 Response	19
 Scale Image Unit	6
 Schedule Job	4
 Script	25
 Selector	70
 Solicit	19
 Strings Function Unit	8
 Switch	13
 Time	6

8.4 IFML specification of the Customer Portal Admin Version

8.4.1 Project

Locale Summary

-  English
-  French
-  Italian
-  Spanish

RunningProfiles Summary

-  Running Profiles

Property Summary

- ajaxDebug
- app-code

SMTPServer Summary

-  smtp server

XsdProvider Summary

-  CommunityWS

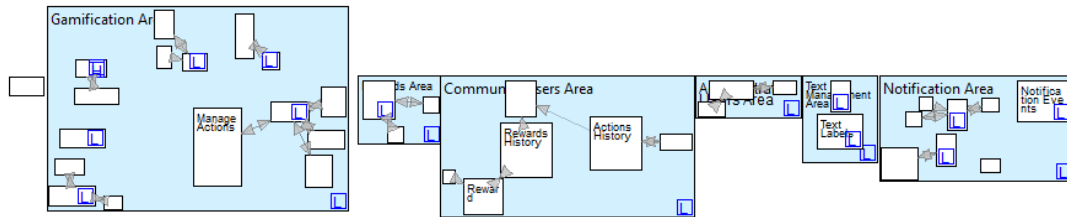
XsdResource Summary

-  CommunityWS /  Community

Database Summary

 Domain Model /  CommunityDB

8.4.2 [SiteView] Administration



Summary Sections

8.4.2..1 Area Summary

-  Administrator Users Area
-  Community Users Area
-  Gamification Area
-  Notification Area
-  Rewards Area
-  Text Management Area

8.4.2..2 MasterPage Summary

-  UserProfile

8.4.2..3 Component Summary

-  [ModuleInstanceUnit] Logout

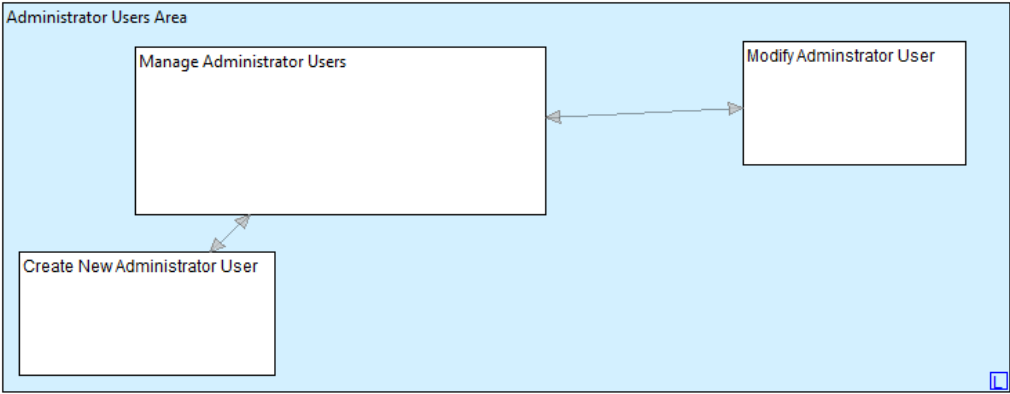
8.4.2..4 Landmark Summary

-  Administrator Users Area
-  Community Users Area
-  Gamification Area
-  Notification Area
-  Rewards Area
-  Text Management Area

8.4.2..5 Incoming Flow Summary

→ Logout from  UserProfile /  [DataUnit] Welcome to  [ModuleInstanceUnit] Logout

[Area] Administrator Users Area



8.4.2..1 Page Summary

- Create New Administrator User
- Manage Administrator Users
- Modify Adminstrator User

8.4.2..2

8.4.2..3 Component Summary

- [ModuleInstanceUnit] Create Adminstrator User
- [ModuleInstanceUnit] Modify Administrator User

8.4.2..4 Landmark Summary

- Administrator Users Area

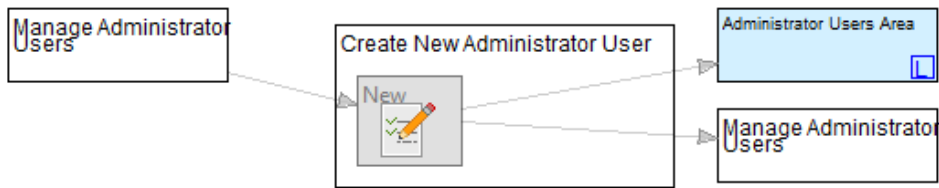
8.4.2..5 Incoming Flow Summary

→ Save	from Create New Administrator User / [EntryUnit] New	to [ModuleInstanceUnit] Create Administrator User
→ Update	from Modify Adminstrator User / [EntryUnit] Modify	to [ModuleInstanceUnit] Modify Administrator User

8.4.2..6 Outgoing Flow Summary

→ OKFlow92	from [ModuleInstanceUnit] Modify Administrator User	to Manage Administrator Users
→ OKFlow93	from [ModuleInstanceUnit] Create Adminstrator User	to Manage Administrator Users

[Page] Create New Administrator User



8.4.2..1 Component Summary

[EntryUnit] New

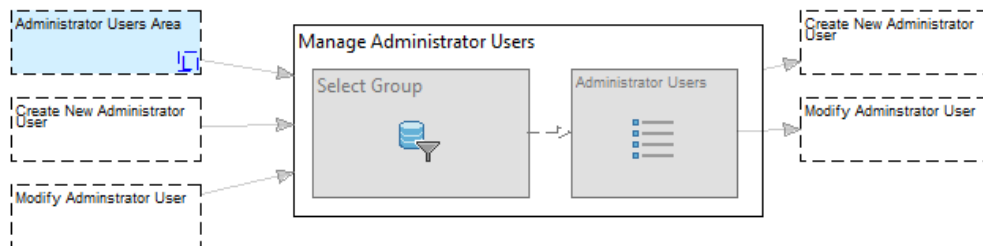
8.4.2..2 Incoming Flow Summary

→ Add New Administrator User	from Manage Administrator Users	to [EntryUnit] New
------------------------------	----------------------------------	---------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] New	to Manage Administrator Users
→ Save	from [EntryUnit] New	to Administrator Users Area / [ModuleInstanceUnit] Create Administrator User

[Page] Manage Administrator Users



8.4.2..1 Component Summary

[PowerIndexUnit] Administrator Users

[SelectorUnit] Select Group

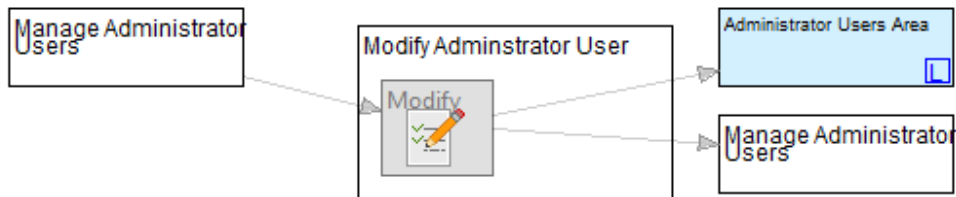
8.4.2..2 Incoming Flow Summary

→ Back	from Create New Administrator User / [EntryUnit] New	to Manage Administrator Users
→ Back	from Modify Administrator User / [EntryUnit] Modify	to Manage Administrator Users
→ OKFlow92	from Administrator Users Area / [ModuleInstanceUnit] Modify Administrator User	to Manage Administrator Users
→ OKFlow93	from Administrator Users Area / [ModuleInstanceUnit] Create Administrator User	to Manage Administrator Users

8.4.2..3 Outgoing Flow Summary

→ Add New Administrator User	from Manage Administrator Users	to Create New Administrator User / [EntryUnit] New
→ Modify	from [PowerIndexUnit] Administrator Users	to Modify Administrator User / [EntryUnit] Modify

[Page] Modify Adminstrator User



8.4.2..1 Component Summary

[EntryUnit] Modify

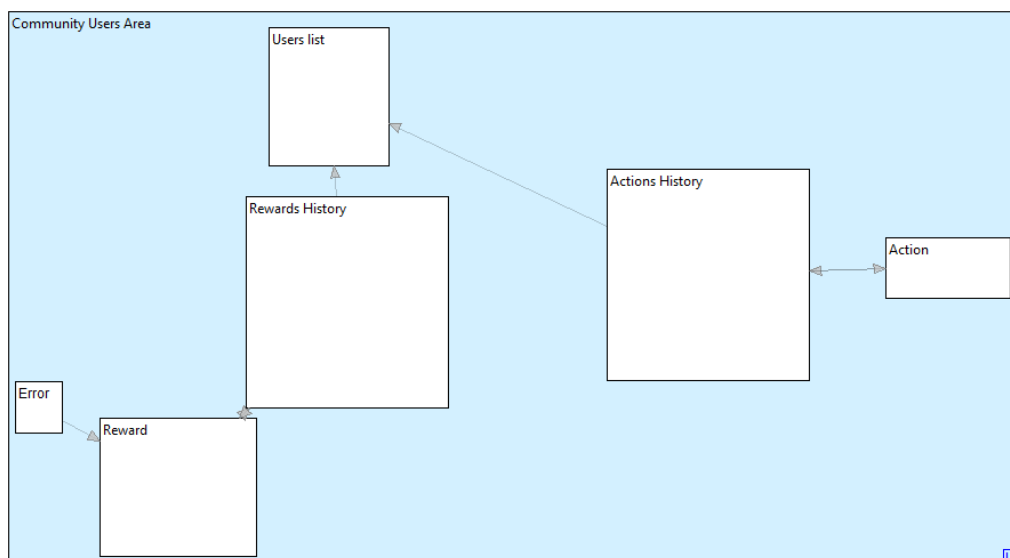
8.4.2..2 Incoming Flow Summary

→ Modify	from Manage Administrator Users / [PowerIndexUnit] Administrator Users	to [EntryUnit] Modify
----------	--	------------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] Modify	to Manage Administrator Users
→ Update	from [EntryUnit] Modify	to Administrator Users Area / [ModuleInstanceUnit] Modify Administrator User

[Area] Community Users Area



8.4.2..1 Page Summary

-  Action
-  Actions History
-  Error
-  Reward
-  Rewards History
-  Users list

8.4.2..2 Component Summary

-  [ModuleInstanceUnit] Assign Action To User
-  [ModuleInstanceUnit] Assign Reward To User
-  [ModuleInstanceUnit] Delete Community User
-  [ModuleInstanceUnit] On Change User Reward
-  [ModuleInstanceUnit] OnChange User Action Type
-  [ModuleInstanceUnit] Remove User Action Instance
-  [ModuleInstanceUnit] Set User Selected
-  [ModuleInstanceUnit] Set User Selected

8.4.2..3 Landmark Summary

-  Community Users Area

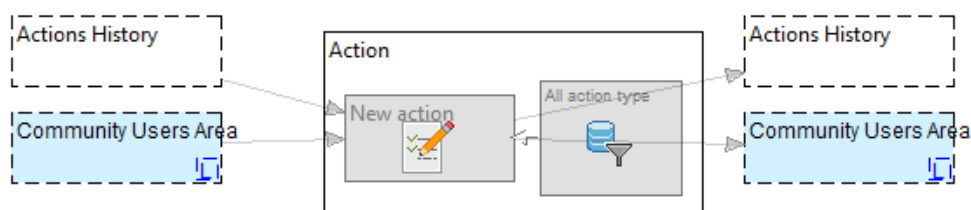
8.4.2..4 Incoming Flow Summary

→ Delete	from Users list / [PowerIndexUnit] Users	to [ModuleInstanceUnit] Delete Community User
→ Delete	from Actions History / [PowerIndexUnit] Actions	to [ModuleInstanceUnit] Remove User Action Instance
→ Flow97	from Action / [EntryUnit] New action	to [ModuleInstanceUnit] OnChange User Action Type
→ OnChange User Reward	from Reward / [EntryUnit] New Reward	to [ModuleInstanceUnit] On Change User Reward
→ Save	from Reward / [EntryUnit] New Reward	to [ModuleInstanceUnit] Assign Reward To User
→ Save	from Action / [EntryUnit] New action	to [ModuleInstanceUnit] Assign Action To User
→ View Actions History	from Users list / [PowerIndexUnit] Users	to [ModuleInstanceUnit] Set User Selected
→ View Rewards History	from Users list / [PowerIndexUnit] Users	to [ModuleInstanceUnit] Set User Selected

8.4.2..5 Outgoing Flow Summary

→ OKFlow115	from [ModuleInstanceUnit] Remove User Action Instance	to Actions History
→ OKFlow150	from [ModuleInstanceUnit] On Change User Reward	to Reward / [EntryUnit] New Reward
→ OKFlow172	from [ModuleInstanceUnit] Assign Reward To User	to Rewards History / [PowerIndexUnit] Rewards
→ OKFlow177	from [ModuleInstanceUnit] Set User Selected	to Rewards History / [PowerIndexUnit] Rewards
→ OKFlow223	from [ModuleInstanceUnit] Assign Action To User	to Actions History / [PowerIndexUnit] Actions
→ OKFlow229	from [ModuleInstanceUnit] OnChange User Action Type	to Action / [EntryUnit] New action
→ OKFlow33	from [ModuleInstanceUnit] Set User Selected	to Actions History / [PowerIndexUnit] Actions
→ OKFlow98	from [ModuleInstanceUnit] Delete Community User	to Users list
→ KOFlow35	from [ModuleInstanceUnit] Assign Reward To User	to Error

[Page] Action



8.4.2..1 Component Summary

 [SelectorUnit] All action type

 [EntryUnit] New action

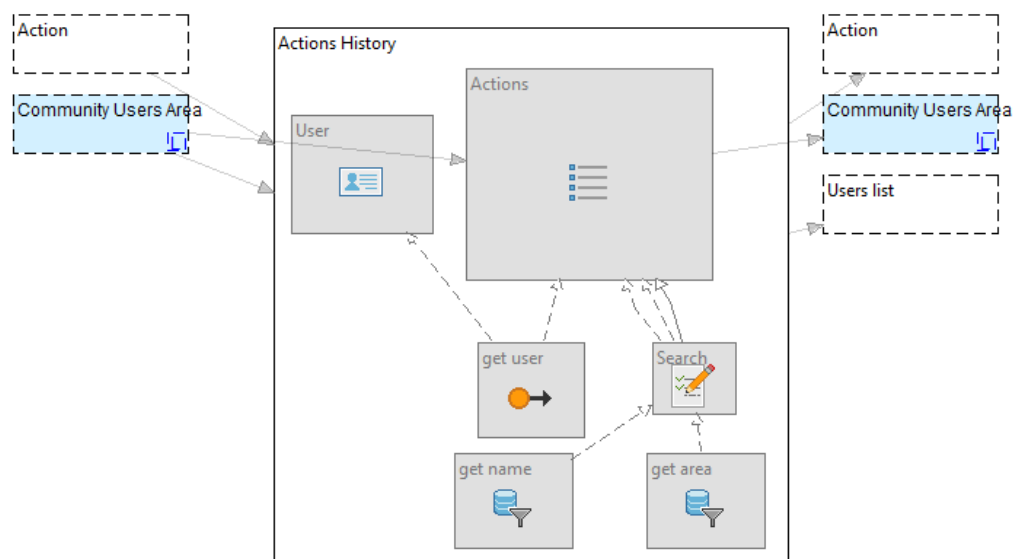
8.4.2..2 Incoming Flow Summary

→ Assign new action	from  Actions History	to  [EntryUnit] New action
→ OKFlow229	from  Community Users Area /  [ModuleInstanceUnit] OnChange User Action Type	to  [EntryUnit] New action

8.4.2..3 Outgoing Flow Summary

→ Back	from  [EntryUnit] New action	to  Actions History
→ Flow97	from  [EntryUnit] New action	to  Community Users Area /  [ModuleInstanceUnit] OnChange User Action Type
→ Save	from  [EntryUnit] New action	to  Community Users Area /  [ModuleInstanceUnit] Assign Action To User

[Page] Actions History



8.4.2..1 Component Summary

 [PowerIndexUnit] Actions

 [SelectorUnit] get area

 [SelectorUnit] get name

 [GetUnit] get user

 [EntryUnit] Search

 [DataUnit] User

8.4.2..2 Incoming Flow Summary

→ Back	from  Action /  [EntryUnit] New action	to  Actions History
→ OKFlow115	from  Community Users Area /  [ModuleInstanceUnit] Remove User Action Instance	to  Actions History
→ OKFlow223	from  Community Users Area /  [ModuleInstanceUnit] Assign Action To User	to  [PowerIndexUnit] Actions
→ OKFlow33	from  Community Users Area /  [ModuleInstanceUnit] Set User Selected	to  [PowerIndexUnit] Actions

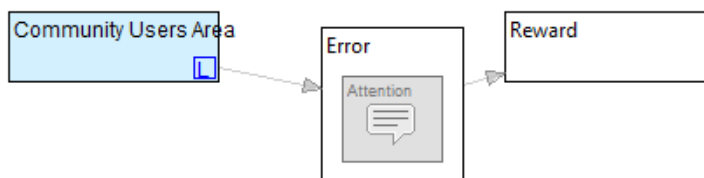
8.4.2..3 Outgoing Flow Summary

→ Assign new action	from  Actions History	to  Action /  [EntryUnit] New action
→ Back	from  Actions History	to  Users list /  [PowerIndexUnit] Users
→ Delete	from  [PowerIndexUnit] Actions	to  Community Users Area /  [ModuleInstanceUnit] Remove User Action Instance

8.4.2..4 Internal Flow Summary

→ Flow120	from  [EntryUnit] Search	to  [PowerIndexUnit] Actions
→ Flow122	from  [EntryUnit] Search	to  [PowerIndexUnit] Actions
→ Search	from  [EntryUnit] Search	to  [PowerIndexUnit] Actions

[Page] Error



8.4.2..1 Component Summary

 [MultiMessageUnit] Attention

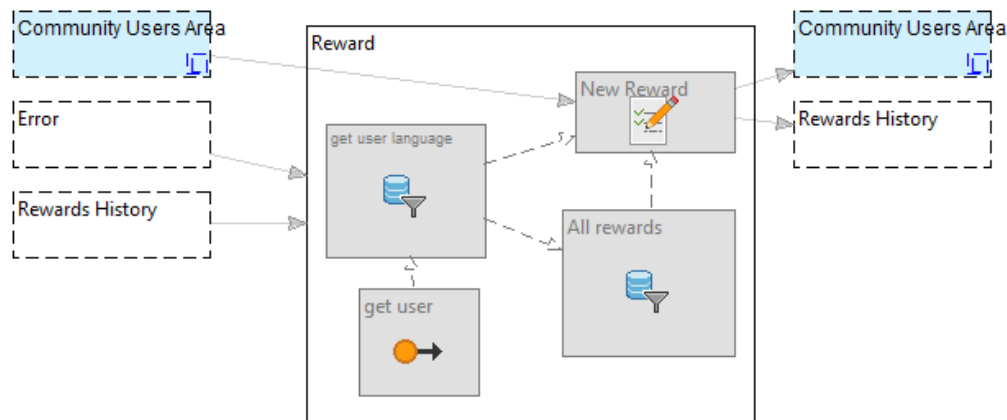
8.4.2..2 Incoming Flow Summary

→ KOFlow35	from  Community Users Area /  [ModuleInstanceUnit] Assign Reward To User	to  Error
------------	--	--

8.4.2..3 Outgoing Flow Summary

→ Back	from  Error	to  Reward
--------	--	---

[Page] Reward



8.4.2..1 ConditionExpression Summary

[dis] disable language

8.4.2..2 Component Summary

[SelectorUnit] All rewards

[GetUnit] get user

[SelectorUnit] get user language

[EntryUnit] New Reward

8.4.2..3 Incoming Flow Summary

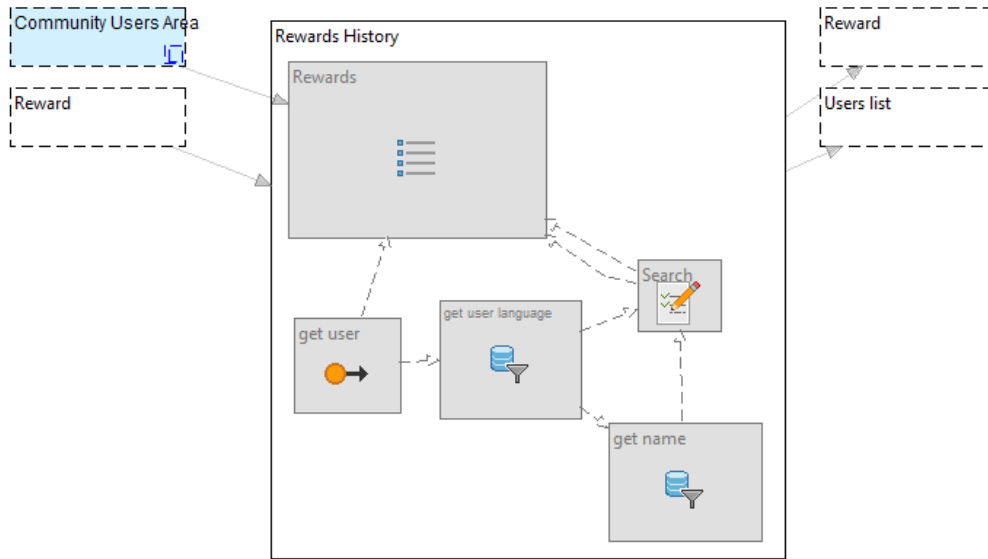
→ Assign new reward	from [Rewards History]	to [Reward]
→ Back	from [Error]	to [Reward]
→ OKFlow150	from [Community Users Area / [ModuleInstanceUnit] On Change User Reward]	to [EntryUnit] New Reward

8.4.2..4

8.4.2..5 Outgoing Flow Summary

→ Back	from [EntryUnit] New Reward	to [Rewards History]
→ OnChange User Reward	from [EntryUnit] New Reward	to [Community Users Area / [ModuleInstanceUnit] On Change User Reward]
→ Save	from [EntryUnit] New Reward	to [Community Users Area / [ModuleInstanceUnit] Assign Reward To User]

[Page] Rewards History



8.4.2..1 ConditionExpression Summary

[66] disable language

8.4.2..2 Component Summary

 [SelectorUnit] get name

- ➡ [GetUnit] get user

 [SelectorUnit] get user language

☰ [PowerIndexUnit] Rewards

 [EntryUnit] Search

8.4.2.3 Incoming Flow Summary

→ Back	from  Reward /  [EntryUnit] New Reward	to  Rewards History
→ OKFlow172	from  Community Users Area /  [ModuleInstanceUnit] Assign Reward To User	to  [PowerIndexUnit] Rewards
→ OKFlow177	from  Community Users Area /  [ModuleInstanceUnit] Set User Selected	to  [PowerIndexUnit] Rewards

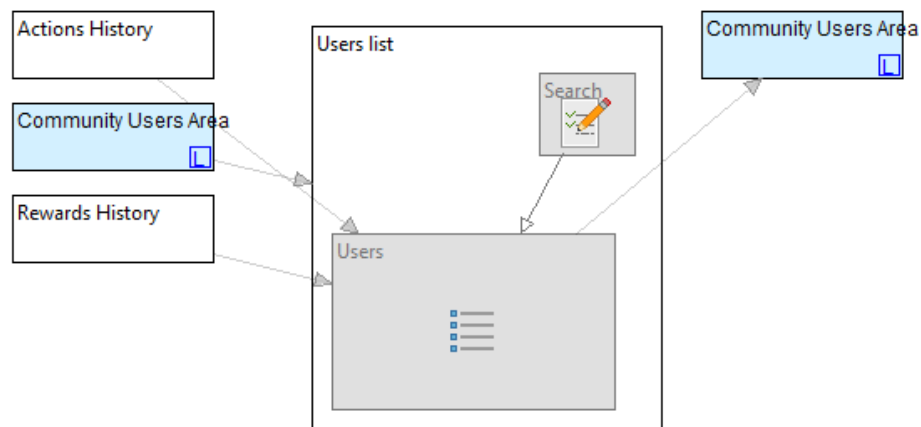
8.4.2.4 Outgoing Flow Summary

→ Assign new reward	from  Rewards History	to  Reward
→ Back	from Rewards History	to Users list / [PowerIndexUnit] Users

8.4.2..5 Internal Flow Summary

→ Flow127	from  [EntryUnit] Search	to  [PowerIndexUnit] Rewards
→ Search	from  [EntryUnit] Search	to  [PowerIndexUnit] Rewards

[Page] Users list



8.4.2..1 Component Summary

[EntryUnit] Search

[PowerIndexUnit] Users

8.4.2..2 Incoming Flow Summary

→ Back	from Actions History	to: [PowerIndexUnit] Users
→ Back	from Rewards History	to: [PowerIndexUnit] Users
→ OKFlow98	from Community Users Area / [ModuleInstanceUnit] Delete Community User	to Users list

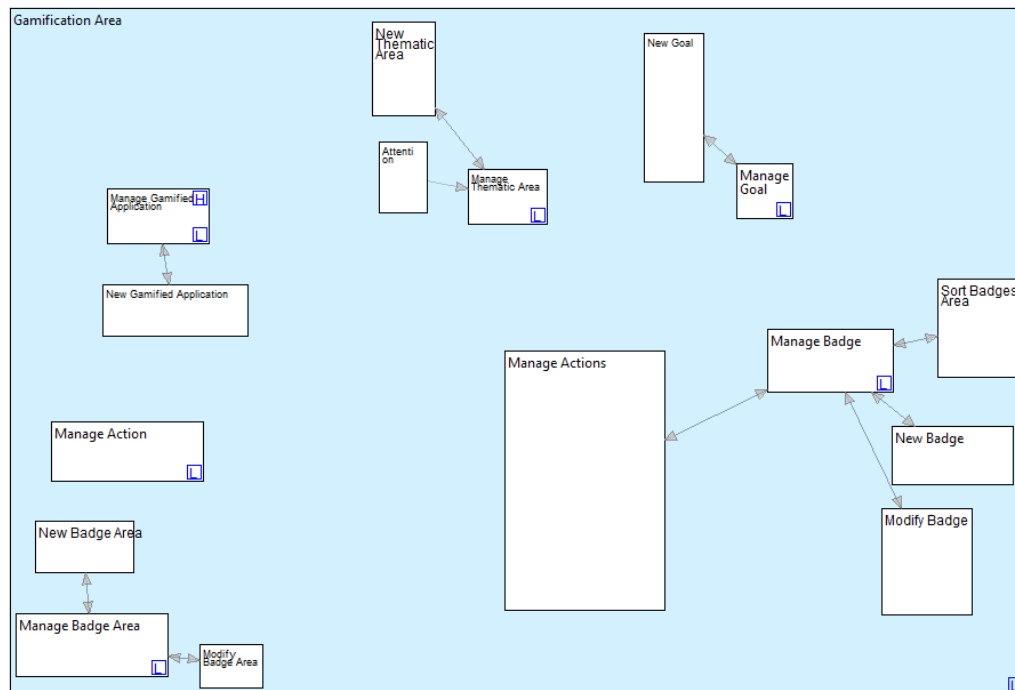
8.4.2..3 Outgoing Flow Summary

→ Delete	from: [PowerIndexUnit] Users	to Community Users Area / [ModuleInstanceUnit] Delete Community User
→ View Actions History	from: [PowerIndexUnit] Users	to Community Users Area / [ModuleInstanceUnit] Set User Selected
→ View Rewards History	from: [PowerIndexUnit] Users	to Community Users Area / [ModuleInstanceUnit] Set User Selected

8.4.2..4 Internal Flow Summary

→ Search	from [EntryUnit] Search	to: [PowerIndexUnit] Users
----------	--------------------------	-----------------------------

[Area] Gamification Area



8.4.2..1 Page Summary

- Attention
- Manage Action
- Manage Actions
- Manage Badge
- Manage Badge Area
- Manage Gamified Application
- Manage Goal
- Manage Thematic Area
- Modify Badge
- Modify Badge Area
- New Badge
- New Badge Area
- New Gamified Application
- New Goal
- New Thematic Area
- Sort Badges Area

8.4.2..2 Component Summary

-  [ModuleInstanceUnit] Add Action To Badge
-  [ModuleInstanceUnit] Add New Gamified Application
-  [ModuleInstanceUnit] Create Action Type
-  [ModuleInstanceUnit] Create Badge Area
-  [ModuleInstanceUnit] Create Goal
-  [ModuleInstanceUnit] Create New Badge
-  [ModuleInstanceUnit] Create Thematic Area
-  [ModuleInstanceUnit] Delete Action From Badge
-  [ModuleInstanceUnit] Delete Action Type
-  [ModuleInstanceUnit] Delete All The Associated Actions
-  [ModuleInstanceUnit] Delete Badge
-  [ModuleInstanceUnit] Delete Gamified Application
-  [ModuleInstanceUnit] Delete Goal
-  [ModuleInstanceUnit] Delete Thematic Area
-  [ModuleInstanceUnit] Edit Gamified Application
-  [ModuleInstanceUnit] Edit Thematic Area
-  [ModuleInstanceUnit] Modify Action Type
-  [ModuleInstanceUnit] Modify Badge
-  [ModuleInstanceUnit] Modify Badge Area
-  [ModuleInstanceUnit] Modify Badge Sorting

8.4.2..3 Landmark Summary

-  Gamification Area
-  Manage Action
-  Manage Badge
-  Manage Badge Area
-  Manage Gamified Application
-  Manage Goal
-  Manage Thematic Area

8.4.2..4 Incoming Flow Summary

→ Add	from  Manage Actions /  [MultiChoiceIndexUnit] All Reputation Actions	to  [ModuleInstanceUnit] Add Action To Badge
→ Add Action	from  Manage Action	to  [ModuleInstanceUnit] Create Action Type
→ Delete	from  Manage Actions /  [PowerIndexUnit] Involved Actions Associated	to  [ModuleInstanceUnit] Delete Action From Badge
→ Delete	from  Manage Thematic Area /  [PowerIndexUnit] Thematic Area	to  [ModuleInstanceUnit] Delete Thematic Area
→ Delete	from  Manage Badge /  [PowerIndexUnit] List of Badges	to  [ModuleInstanceUnit] Delete Badge

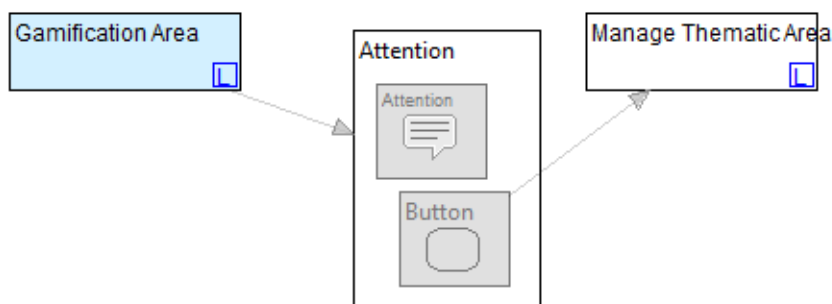
→ Delete	from  Manage Gamified Application /  [PowerIndexUnit] List of Gamified Applications	to  [ModuleInstanceUnit] Delete Gamified Application
→ Delete	from  Manage Goal /  [PowerIndexUnit] List of Goals	to  [ModuleInstanceUnit] Delete Goal
→ Delete	from  Manage Action /  [PowerIndexUnit] List of Actions	to  [ModuleInstanceUnit] Delete Action Type
→ Delete All	from  Manage Actions /  [DataUnit] Badge Selected	to  [ModuleInstanceUnit] Delete All The Associated Actions
→ Modify	from  Manage Thematic Area /  [PowerIndexUnit] Thematic Area	to  [ModuleInstanceUnit] Edit Thematic Area
→ Modify	from  Manage Gamified Application /  [PowerIndexUnit] List of Gamified Applications	to  [ModuleInstanceUnit] Edit Gamified Application
→ Modify	from  Manage Action /  [PowerIndexUnit] List of Actions	to  [ModuleInstanceUnit] Modify Action Type
→ Save	from  New Gamified Application /  [EntryUnit] New Gamified Application	to  [ModuleInstanceUnit] Add New Gamified Application
→ Save	from  New Badge /  [EntryUnit] New Badge	to  [ModuleInstanceUnit] Create New Badge
→ Save	from  New Goal /  [EntryUnit] Goal	to  [ModuleInstanceUnit] Create Goal
→ Save	from  New Thematic Area /  [EntryUnit] Thematic Area	to  [ModuleInstanceUnit] Create Thematic Area
→ Save	from  New Badge Area /  [EntryUnit] New	to  [ModuleInstanceUnit] Create Badge Area
→ Update	from  Sort Badges Area /  [MultiEntryUnit] Choose the Display Sorting	to  [ModuleInstanceUnit] Modify Badge Sorting
→ Update	from  Modify Badge Area /  [EntryUnit] Modify	to  [ModuleInstanceUnit] Modify Badge Area
→ Update	from  Modify Badge /  [EntryUnit] Modify Badge	to  [ModuleInstanceUnit] Modify Badge

8.4.2..5 Outgoing Flow Summary

→ Back	from  [ModuleInstanceUnit] Edit Thematic Area	to  Manage Thematic Area /  [PowerIndexUnit] Thematic Area
→ Back	from  [ModuleInstanceUnit] Edit Gamified Application	to  Manage Gamified Application
→ OKFlow101	from  [ModuleInstanceUnit] Create Action Type	to  Manage Action
→ OKFlow106	from  [ModuleInstanceUnit] Modify Action Type	to  Manage Action /  [PowerIndexUnit] List of Actions
→ OKFlow112	from  [ModuleInstanceUnit] Create Thematic Area	to  Manage Thematic Area
→ OKFlow12	from  [ModuleInstanceUnit] Delete Goal	to  Manage Goal /  [PowerIndexUnit] List of Goals
→ OKFlow124	from  [ModuleInstanceUnit] Delete Thematic Area	to  Attention
→ OKFlow153	from  [ModuleInstanceUnit] Delete Gamified Application	to  Manage Gamified Application
→ OKFlow157	from  [ModuleInstanceUnit] Delete Thematic Area	to  Manage Thematic Area
→ OKFlow307	from  [ModuleInstanceUnit] Create Badge Area	to  Manage Badge Area /  [PowerIndexUnit] List of Badge Areas

→ OKFlow311	from [ModuleInstanceUnit] Modify Badge Area	to Manage Badge Area / [PowerIndexUnit] List of Badge Areas
→ OKFlow318	from [ModuleInstanceUnit] Delete Badge	to Manage Badge / [PowerIndexUnit] List of Badges
→ OKFlow321	from [ModuleInstanceUnit] Modify Badge Sorting	to Manage Badge / [PowerIndexUnit] List of Badges
→ OKFlow327	from [ModuleInstanceUnit] Create New Badge	to Manage Badge / [PowerIndexUnit] List of Badges
→ OKFlow334	from [ModuleInstanceUnit] Modify Badge	to Manage Badge / [PowerIndexUnit] List of Badges
→ OKFlow338	from [ModuleInstanceUnit] Delete All The Associated Actions	to Manage Actions
→ OKFlow341	from [ModuleInstanceUnit] Add Action To Badge	to Manage Actions
→ OKFlow344	from [ModuleInstanceUnit] Delete Action From Badge	to Manage Actions
→ OKFlow42	from [ModuleInstanceUnit] Add New Gamified Application	to Manage Gamified Application
→ OKFlow5	from [ModuleInstanceUnit] Create Goal	to Manage Goal
→ OKFlow84	from [ModuleInstanceUnit] Delete Action Type	to Manage Action
→ KOFlow41	from [ModuleInstanceUnit] Delete Action Type	to Manage Action

[Page] Attention



8.4.2..1 Component Summary

[MultiMessageUnit] Attention

[NoOpContentUnit] Button

8.4.2..2

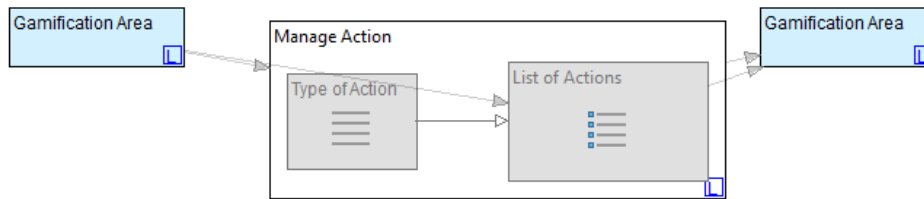
8.4.2..3 Incoming Flow Summary

→ OKFlow124	from Gamification Area / [ModuleInstanceUnit] Delete Thematic Area	to Attention
-------------	--	--------------

8.4.2..4 Outgoing Flow Summary

→ Ok	from [NoOpContentUnit] Button	to Manage Thematic Area
------	-------------------------------	-------------------------

[Page] Manage Action



8.4.2..1 Component Summary

☰ [PowerIndexUnit] List of Actions

☰ [IndexUnit] Type of Action

8.4.2..2 Landmark Summary

📄 Manage Action

8.4.2..3 Incoming Flow Summary

→ OKFlow101	from 📄 Gamification Area / ☰ [ModuleInstanceUnit] Create Action Type	to 📄 Manage Action
→ OKFlow106	from 📄 Gamification Area / ☰ [ModuleInstanceUnit] Modify Action Type	to ☰ [PowerIndexUnit] List of Actions
→ OKFlow84	from 📄 Gamification Area / ☰ [ModuleInstanceUnit] Delete Action Type	to 📄 Manage Action
→ KOFlow41	from 📄 Gamification Area / ☰ [ModuleInstanceUnit] Delete Action Type	to 📄 Manage Action

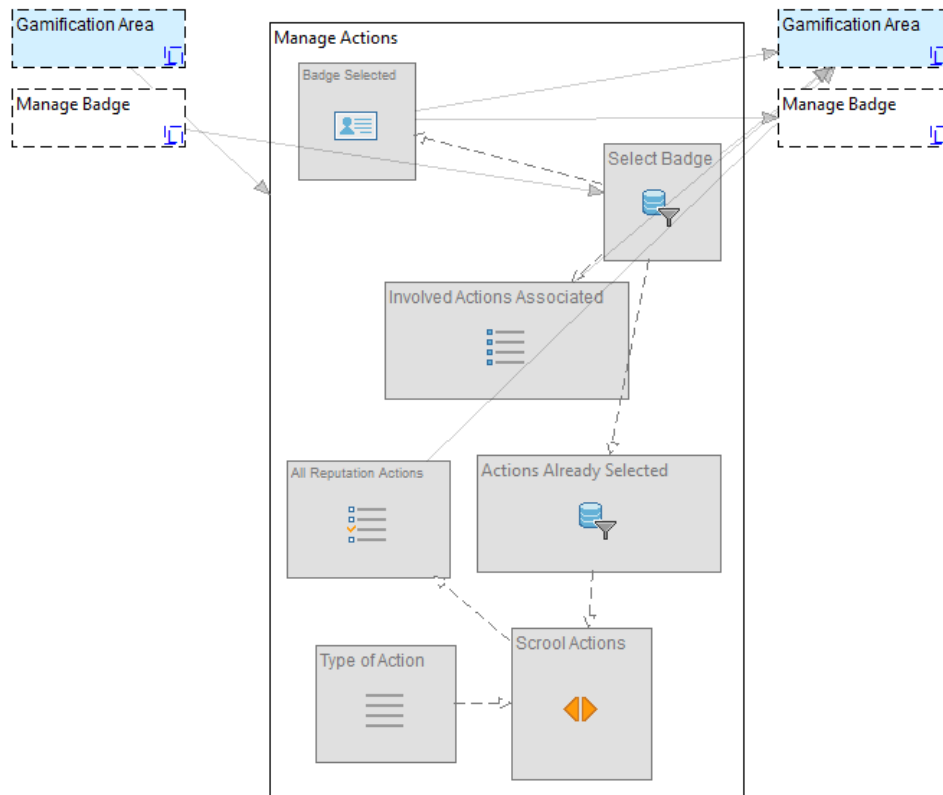
8.4.2..4 Outgoing Flow Summary

→ Add Action	from 📄 Manage Action	to 📄 Gamification Area / ☰ [ModuleInstanceUnit] Create Action Type
→ Delete	from ☰ [PowerIndexUnit] List of Actions	to 📄 Gamification Area / ☰ [ModuleInstanceUnit] Delete Action Type
→ Modify	from ☰ [PowerIndexUnit] List of Actions	to 📄 Gamification Area / ☰ [ModuleInstanceUnit] Modify Action Type

8.4.2..5 Internal Flow Summary

→ Link11	from ☰ [IndexUnit] Type of Action	to ☰ [PowerIndexUnit] List of Actions
----------	-----------------------------------	---------------------------------------

[Page] Manage Actions



8.4.2..1 ConditionExpression Summary

[DeleteAll]

8.4.2..2 Component Summary

- [SelectorUnit] Actions Already Selected
- [MultiChoiceIndexUnit] All Reputation Actions
- [DataUnit] Badge Selected
- [PowerIndexUnit] Involved Actions Associated
- [ScrollerUnit] Scrool Actions
- [SelectorUnit] Select Badge
- [IndexUnit] Type of Action

8.4.2..3 Variable Summary

- ◆ ActionsAssociated
- ◆ area
- ◆ score
- ◆ title

8.4.2..4 Incoming Flow Summary

→ Manage actions	from Manage Badge / [PowerIndexUnit] List of Badges	to [SelectorUnit] Select Badge
→ OKFlow338	from Gamification Area / [ModuleInstanceUnit] Delete All The Associated Actions	to Manage Actions
→ OKFlow341	from Gamification Area / [ModuleInstanceUnit] Add Action To Badge	to Manage Actions
→ OKFlow344	from Gamification Area / [ModuleInstanceUnit] Delete Action From Badge	to Manage Actions

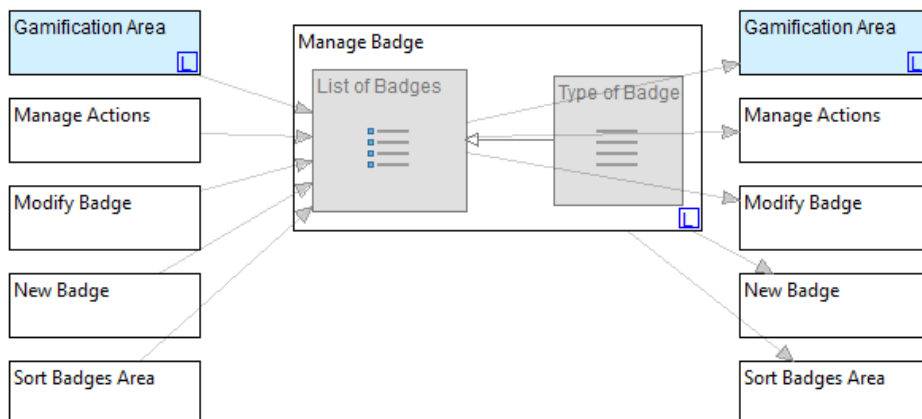
8.4.2..5 Outgoing Flow Summary

→ Add	from [MultiChoiceIndexUnit] All Reputation Actions	to Gamification Area / [ModuleInstanceUnit] Add Action To Badge
→ Back	from [DataUnit] Badge Selected	to Manage Badge / [PowerIndexUnit] List of Badges
→ Delete	from [PowerIndexUnit] Involved Actions Associated	to Gamification Area / [ModuleInstanceUnit] Delete Action From Badge
→ Delete All	from [DataUnit] Badge Selected	to Gamification Area / [ModuleInstanceUnit] Delete All The Associated Actions

8.4.2..6 Internal Flow Summary

→ Link11	from [IndexUnit] Type of Action	to [ScrollerUnit] Scrool Actions
→ Link206	from [ScrollerUnit] Scrool Actions	to [MultiChoiceIndexUnit] All Reputation Actions

[Page] Manage Badge



8.4.2..1 ConditionExpression Summary

 area

8.4.2..2 Component Summary

 [PowerIndexUnit] List of Badges

 [IndexUnit] Type of Badge

8.4.2..3

8.4.2..4 Variable Summary

 type

8.4.2..5 Landmark Summary

 Manage Badge

8.4.2..6 Incoming Flow Summary

→ Back	from  New Badge /  [EntryUnit] New Badge	to  [PowerIndexUnit] List of Badges
→ Back	from  Sort Badges Area /  [MultiEntryUnit] Choose the Display Sorting	to  [PowerIndexUnit] List of Badges
→ Back	from  Modify Badge /  [EntryUnit] Modify Badge	to  [PowerIndexUnit] List of Badges
→ Back	from  Manage Actions /  [DataUnit] Badge Selected	to  [PowerIndexUnit] List of Badges
→ OKFlow318	from  Gamification Area /  [ModuleInstanceUnit] Delete Badge	to  [PowerIndexUnit] List of Badges
→ OKFlow321	from  Gamification Area /  [ModuleInstanceUnit] Modify Badge Sorting	to  [PowerIndexUnit] List of Badges
→ OKFlow327	from  Gamification Area /  [ModuleInstanceUnit] Create New Badge	to  [PowerIndexUnit] List of Badges
→ OKFlow334	from  Gamification Area /  [ModuleInstanceUnit] Modify Badge	to  [PowerIndexUnit] List of Badges

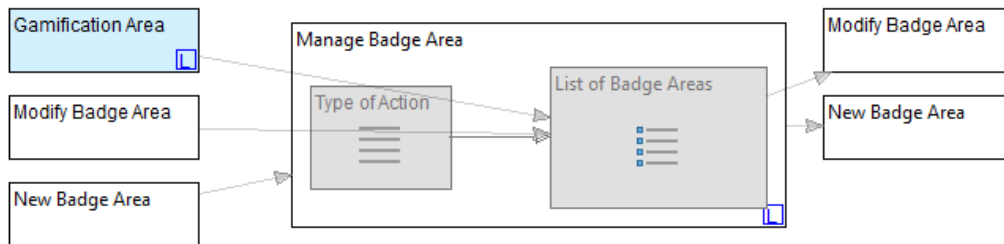
8.4.2..7 Outgoing Flow Summary

→ Add Badge	from  Manage Badge	to  New Badge /  [EntryUnit] New Badge
→ Choose the Display Sorting	from  Manage Badge	to  Sort Badges Area /  [MultiEntryUnit] Choose the Display Sorting
→ Delete	from  [PowerIndexUnit] List of Badges	to  Gamification Area /  [ModuleInstanceUnit] Delete Badge
→ Manage actions	from  [PowerIndexUnit] List of Badges	to  Manage Actions /  [SelectorUnit] Select Badge
→ Modify	from  [PowerIndexUnit] List of Badges	to  Modify Badge /  [SelectorUnit] Select Badge

8.4.2..8 Internal Flow Summary

→ Link11	from [IndexUnit] Type of Badge	to [PowerIndexUnit] List of Badges
----------	--------------------------------	------------------------------------

[Page] Manage Badge Area



8.4.2..1 Component Summary

- [PowerIndexUnit] List of Badge Areas
- [IndexUnit] Type of Action

8.4.2..2 Landmark Summary

- Manage Badge Area

8.4.2..3 Incoming Flow Summary

→ Back	from Modify Badge Area / [EntryUnit] Modify	to [PowerIndexUnit] List of Badge Areas
→ Back	from New Badge Area / [EntryUnit] New	to Manage Badge Area
→ OKFlow307	from Gamification Area / [ModuleInstanceUnit] Create Badge Area	to [PowerIndexUnit] List of Badge Areas
→ OKFlow311	from Gamification Area / [ModuleInstanceUnit] Modify Badge Area	to [PowerIndexUnit] List of Badge Areas

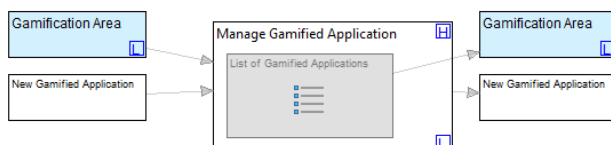
8.4.2..4 Outgoing Flow Summary

→ Add Area	from Manage Badge Area	to New Badge Area
→ Modify	from [PowerIndexUnit] List of Badge Areas	to Modify Badge Area / [EntryUnit] Modify

8.4.2..5 Internal Flow Summary

→ Flow171	from [IndexUnit] Type of Action	to [PowerIndexUnit] List of Badge Areas
-----------	---------------------------------	---

[Page] Manage Gamified Application



8.4.2..1 Component Summary

[PowerIndexUnit] List of Gamified Applications

8.4.2..2 Landmark Summary

Manage Gamified Application

8.4.2..3

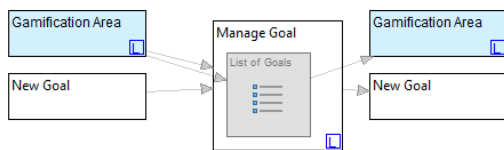
8.4.2..4 Incoming Flow Summary

→ Back	from [New Gamified Application] / [EntryUnit] New Gamified Application	to [Manage Gamified Application]
→ Back	from [Gamification Area] / [ModuleInstanceUnit] Edit Gamified Application	to [Manage Gamified Application]
→ OKFlow153	from [Gamification Area] / [ModuleInstanceUnit] Delete Gamified Application	to [Manage Gamified Application]
→ OKFlow42	from [Gamification Area] / [ModuleInstanceUnit] Add New Gamified Application	to [Manage Gamified Application]

8.4.2..5 Outgoing Flow Summary

→ Add Gamified Application	from [Manage Gamified Application]	to [New Gamified Application]
→ Delete	from [PowerIndexUnit] List of Gamified Applications	to [Gamification Area] / [ModuleInstanceUnit] Delete Gamified Application
→ Modify	from [PowerIndexUnit] List of Gamified Applications	to [Gamification Area] / [ModuleInstanceUnit] Edit Gamified Application

[Page] Manage Goal



8.4.2..1 Component Summary

[PowerIndexUnit] List of Goals

8.4.2..2 Landmark Summary

Manage Goal

8.4.2..3 Incoming Flow Summary

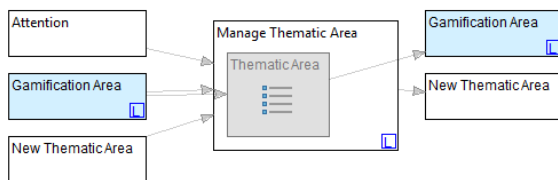
→ Back	from [New Goal] / [EntryUnit] Goal	to [Manage Goal]
--------	------------------------------------	------------------

→ OKFlow12	from  Gamification Area /  [ModuleInstanceUnit] Delete Goal	to  [PowerIndexUnit] List of Goals
→ OKFlow5	from  Gamification Area /  [ModuleInstanceUnit] Create Goal	to  Manage Goal

8.4.2..4 Outgoing Flow Summary

→ Delete	from  [PowerIndexUnit] List of Goals	to  Gamification Area /  [ModuleInstanceUnit] Delete Goal
→ New Goal	from  Manage Goal	to  New Goal

[Page] Manage Thematic Area



8.4.2..1 Component Summary

 [PowerIndexUnit] Thematic Area

8.4.2..2 Landmark Summary

 Manage Thematic Area

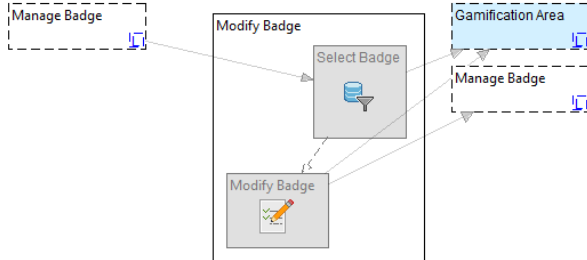
8.4.2..3 Incoming Flow Summary

→ Back	from  New Thematic Area /  [EntryUnit] Thematic Area	to  Manage Thematic Area
→ Ok	from  Attention /  [NoOpContentUnit] Button	to  Manage Thematic Area
→ Back	from  Gamification Area /  [ModuleInstanceUnit] Edit Thematic Area	to  [PowerIndexUnit] Thematic Area
→ OKFlow112	from  Gamification Area /  [ModuleInstanceUnit] Create Thematic Area	to  Manage Thematic Area
→ OKFlow157	from  Gamification Area /  [ModuleInstanceUnit] Delete Thematic Area	to  Manage Thematic Area

8.4.2..4 Outgoing Flow Summary

→ Delete	from  [PowerIndexUnit] Thematic Area	to  Gamification Area /  [ModuleInstanceUnit] Delete Thematic Area
→ Modify	from  [PowerIndexUnit] Thematic Area	to  Gamification Area /  [ModuleInstanceUnit] Edit Thematic Area
→ New Thematic Area	from  Manage Thematic Area	to  New Thematic Area

[Page] Modify Badge



8.4.2..1 Component Summary

[EntryUnit] Modify Badge

[SelectorUnit] Select Badge

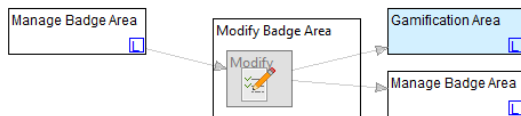
8.4.2..2 Incoming Flow Summary

→ Modify	from Manage Badge / [PowerIndexUnit] List of Badges	to [SelectorUnit] Select Badge
----------	---	---------------------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] Modify Badge	to Manage Badge / [PowerIndexUnit] List of Badges
→ Update	from [EntryUnit] Modify Badge	to Gamification Area / [ModuleInstanceUnit] Modify Badge

[Page] Modify Badge Area



8.4.2..1 Component Summary

[EntryUnit] Modify

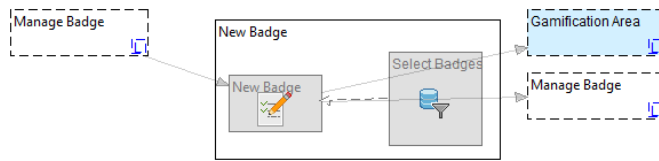
8.4.2..2 Incoming Flow Summary

→ Modify	from Manage Badge Area / [PowerIndexUnit] List of Badge Areas	to [EntryUnit] Modify
----------	---	------------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] Modify	to Manage Badge Area / [PowerIndexUnit] List of Badge Areas
→ Update	from [EntryUnit] Modify	to Gamification Area / [ModuleInstanceUnit] Modify Badge Area

[Page] New Badge



8.4.2..1 Component Summary

[EntryUnit] New Badge

[SelectorUnit] Select Badges

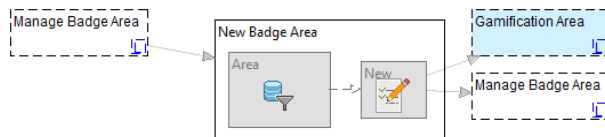
8.4.2..2 Incoming Flow Summary

→ Add Badge	from Manage Badge	to [EntryUnit] New Badge
-------------	--------------------	---------------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] New Badge	to Manage Badge / [PowerIndexUnit] List of Badges
→ Save	from [EntryUnit] New Badge	to Gamification Area / [ModuleInstanceUnit] Create New Badge

[Page] New Badge Area



8.4.2..1 Component Summary

[SelectorUnit] Area

[EntryUnit] New

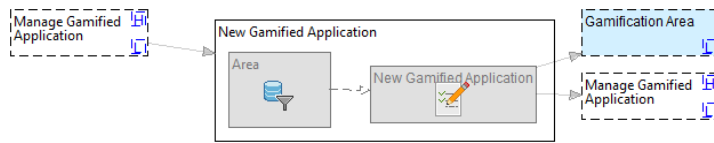
8.4.2..2 Incoming Flow Summary

→ Add Area	from Manage Badge Area	to New Badge Area
------------	-------------------------	--------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] New	to Manage Badge Area
→ Save	from [EntryUnit] New	to Gamification Area / [ModuleInstanceUnit] Create Badge Area

[Page] New Gamified Application



8.4.2..1 Component Summary

[SelectorUnit] Area

[EntryUnit] New Gamified Application

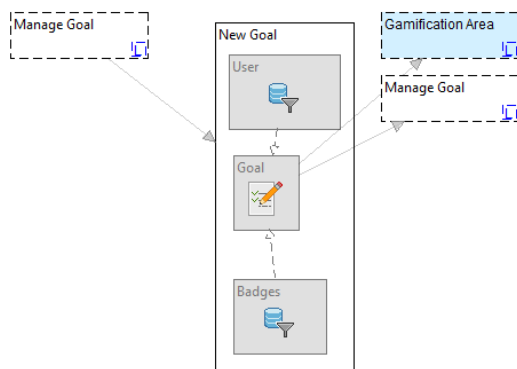
8.4.2..2 Incoming Flow Summary

→ Add Gamified Application	from [Icon] Manage Gamified Application	to [Icon] New Gamified Application
----------------------------	---	------------------------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [Icon] [EntryUnit] New Gamified Application	to [Icon] Manage Gamified Application
→ Save	from [Icon] [EntryUnit] New Gamified Application	to [Icon] Gamification Area / [Icon] [ModuleInstanceUnit] Add New Gamified Application

[Page] New Goal



8.4.2..1 Component Summary

[SelectorUnit] Badges

[EntryUnit] Goal

[SelectorUnit] User

8.4.2..2 Incoming Flow Summary

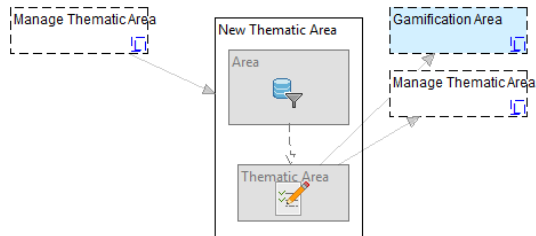
→ New Goal	from [Icon] Manage Goal	to [Icon] New Goal
------------	-------------------------	--------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [Icon] [EntryUnit] Goal	to [Icon] Manage Goal
--------	------------------------------	-----------------------

→ Save	from [EntryUnit] Goal	to Gamification Area / [ModuleInstanceUnit] Create Goal
--------	------------------------	---

[Page] New Thematic Area



8.4.2..1 Component Summary

[SelectorUnit] Area

[EntryUnit] Thematic Area

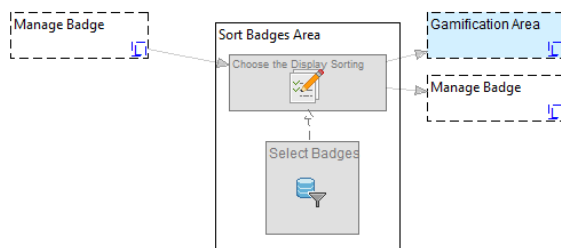
8.4.2..2 Incoming Flow Summary

→ New Thematic Area	from Manage Thematic Area	to New Thematic Area
---------------------	----------------------------	-----------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] Thematic Area	to Manage Thematic Area
→ Save	from [EntryUnit] Thematic Area	to Gamification Area / [ModuleInstanceUnit] Create Thematic Area

[Page] Sort Badges Area



8.4.2..1 Component Summary

[MultiEntryUnit] Choose the Display Sorting

[SelectorUnit] Select Badges

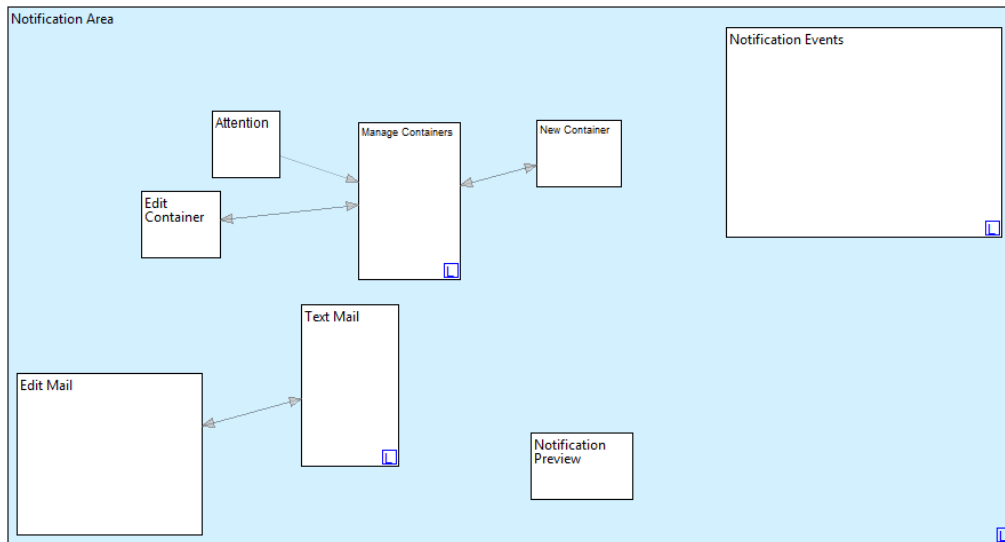
8.4.2..2 Incoming Flow Summary

→ Choose the Display Sorting	from Manage Badge	to [MultiEntryUnit] Choose the Display Sorting
------------------------------	--------------------	---

8.4.2..3 Outgoing Flow Summary

→ Back	from  [MultiEntryUnit] Choose the Display Sorting	to  Manage Badge /  [PowerIndexUnit] List of Badges
→ Update	from  [MultiEntryUnit] Choose the Display Sorting	to  Gamification Area /  [ModuleInstanceUnit] Modify Badge Sorting

[Area] Notification Area



8.4.2..1 Page Summary

-  Attention
-  Edit Container
-  Edit Mail
-  Manage Containers
-  New Container
-  Notification Events
-  Notification Preview
-  Text Mail

8.4.2..2 Component Summary

-  [ModuleInstanceUnit] Create Container
-  [ModuleInstanceUnit] Delete Container
-  [ModuleInstanceUnit] Email Preview
-  [ModuleInstanceUnit] OnChange NoOp
-  [ModuleInstanceUnit] OnChange NoOp
-  [ModuleInstanceUnit] Update Container
-  [ModuleInstanceUnit] Update Email Text

8.4.2..3 Landmark Summary

-  Notification Area
-  Manage Containers
-  Notification Events
-  Text Mail

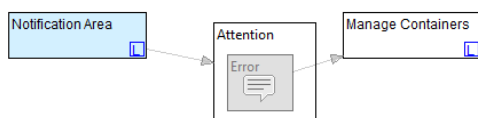
8.4.2..4 Incoming Flow Summary

→ Delete	from  Manage Containers /  [PowerIndexUnit] Containers	to  [ModuleInstanceUnit] Delete Container
→ Flow42	from  Manage Containers /  [EntryUnit] Language	to  [ModuleInstanceUnit] OnChange NoOp
→ Flow42	from  Text Mail /  [EntryUnit] Language	to  [ModuleInstanceUnit] OnChange NoOp
→ Preview	from  Text Mail /  [PowerIndexUnit] Mails	to  [ModuleInstanceUnit] Email Preview
→ Save	from  Edit Container /  [EntryUnit] Container	to  [ModuleInstanceUnit] Update Container
→ Save	from  New Container /  [EntryUnit] Container	to  [ModuleInstanceUnit] Create Container
→ Save	from  Edit Mail /  [EntryUnit] Mail	to  [ModuleInstanceUnit] Update Email Text

8.4.2..5 Outgoing Flow Summary

→ OKFlow104	from  [ModuleInstanceUnit] OnChange NoOp	to  Manage Containers
→ OKFlow104	from  [ModuleInstanceUnit] OnChange NoOp	to  Text Mail
→ OKFlow364	from  [ModuleInstanceUnit] Email Preview	to  Notification Preview /  [MultiMessageUnit] mail template
→ OKFlow376	from  [ModuleInstanceUnit] Delete Container	to  Manage Containers
→ OKFlow379	from  [ModuleInstanceUnit] Update Container	to  Manage Containers /  [PowerIndexUnit] Containers
→ OKFlow383	from  [ModuleInstanceUnit] Create Container	to  Manage Containers /  [PowerIndexUnit] Containers
→ OKFlow98	from  [ModuleInstanceUnit] Update Email Text	to  Text Mail /  [PowerIndexUnit] Mails
→ KOFlow101	from  [ModuleInstanceUnit] Email Preview	to  Notification Preview /  [MultiMessageUnit] mail template
→ KOFlow109	from  [ModuleInstanceUnit] Delete Container	to  Attention

[Page] Attention



8.4.2..1 Component Summary

 [MultiMessageUnit] Error

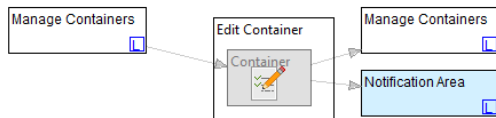
8.4.2..2 Incoming Flow Summary

→ KOFlow109	from  Notification Area /  [ModuleInstanceUnit] Delete Container	to  Attention
-------------	--	--

8.4.2..3 Outgoing Flow Summary

→ Ok	from  [MultiMessageUnit] Error	to  Manage Containers
------	---	--

[Page] Edit Container



8.4.2..1 Component Summary

 [EntryUnit] Container

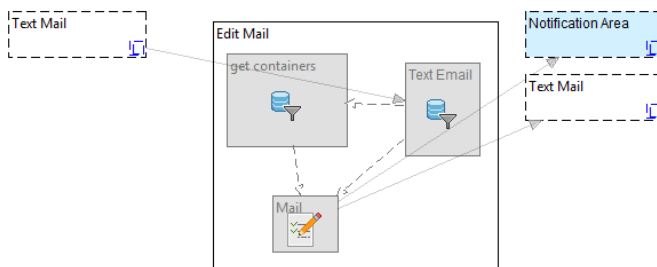
8.4.2..2 Incoming Flow Summary

→ Modify	from  Manage Containers /  [PowerIndexUnit] Containers	to  [EntryUnit] Container
----------	--	--

8.4.2..3 Outgoing Flow Summary

→ Back	from  [EntryUnit] Container	to  Manage Containers /  [PowerIndexUnit] Containers
→ Save	from  [EntryUnit] Container	to  Notification Area /  [ModuleInstanceUnit] Update Container

[Page] Edit Mail



8.4.2..1 Component Summary

 [SelectorUnit] get containers

 [EntryUnit] Mail

 [SelectorUnit] Text Email

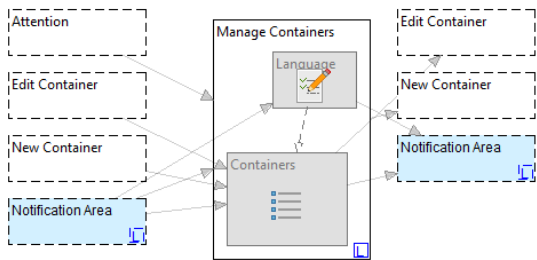
8.4.2..2 Incoming Flow Summary

→ Modify	from  Text Mail /  [PowerIndexUnit] Mails	to  [SelectorUnit] Text Email
----------	---	--

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] Mail	to Text Mail / [PowerIndexUnit] Mails
→ Save	from [EntryUnit] Mail	to Notification Area / [ModuleInstanceUnit] Update Email Text

[Page] Manage Containers



8.4.2..1 Component Summary

- [PowerIndexUnit] Containers
- [EntryUnit] Language

8.4.2..2 Landmark Summary

- Manage Containers

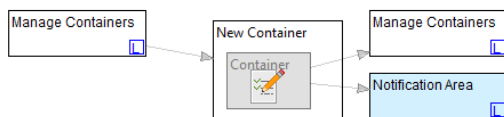
8.4.2..3 Incoming Flow Summary

→ Back	from  New Container /  [EntryUnit]	to  [PowerIndexUnit] Containers
→ Back	from  Edit Container /  [EntryUnit]	to  [PowerIndexUnit] Containers
→ Ok	from  Attention /  [MultiMessageUnit] Error	to  Manage Containers
→ OKFlow104	from  Notification Area /  [ModuleInstanceUnit] OnChange NoOp	to  Manage Containers
→ OKFlow376	from  Notification Area /  [ModuleInstanceUnit] Delete Container	to  Manage Containers
→ OKFlow379	from  Notification Area /  [ModuleInstanceUnit] Update Container	to  [PowerIndexUnit] Containers
→ OKFlow383	from  Notification Area /  [ModuleInstanceUnit] Create Container	to  [PowerIndexUnit] Containers

8.4.2..4 Outgoing Flow Summary

→ Add container	from  Manage Containers	to  New Container
→ Delete	from  [PowerIndexUnit] Containers	to  Notification Area /  [ModuleInstanceUnit] Delete Container
→ Flow42	from  [EntryUnit] Language	to  Notification Area /  [ModuleInstanceUnit] OnChange NoOp
→ Modify	from  [PowerIndexUnit] Containers	to  Edit Container /  [EntryUnit] Container

[Page] New Container



8.4.2..1 Component Summary

 [EntryUnit] Container

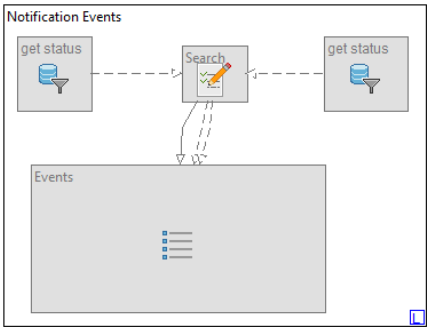
8.4.2..2 Incoming Flow Summary

→ Add container	from  Manage Containers	to  New Container
-----------------	--	--

8.4.2..3 Outgoing Flow Summary

→ Back	from  [EntryUnit] Container	to  Manage Containers /  [PowerIndexUnit] Containers
→ Save	from  [EntryUnit] Container	to  Notification Area /  [ModuleInstanceUnit] Create Container

[Page] Notification Events



8.4.2..1 Component Summary

- [PowerIndexUnit] Events
- [SelectorUnit] get status
- [SelectorUnit] get status
- [EntryUnit] Search

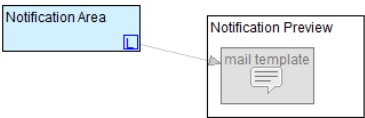
8.4.2..2 Landmark Summary

- Notification Events

8.4.2..3 Internal Flow Summary

→ Flow143	from [EntryUnit] Search	to [PowerIndexUnit] Events
→ Flow148	from [EntryUnit] Search	to [PowerIndexUnit] Events
→ Search	from [EntryUnit] Search	to [PowerIndexUnit] Events

[Page] Notification Preview



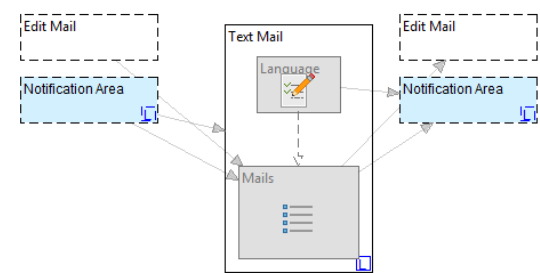
8.4.2..1 Component Summary

- [MultiMessageUnit] mail template

8.4.2..2 Incoming Flow Summary

→ OKFlow364	from Notification Area / [ModuleInstanceUnit] Email Preview	to [MultiMessageUnit] mail template
→ KOFlow101	from Notification Area / [ModuleInstanceUnit] Email Preview	to [MultiMessageUnit] mail template

[Page] Text Mail



8.4.2..1 Component Summary

- [EntryUnit] Language
- [PowerIndexUnit] Mails

8.4.2..2 Landmark Summary

- Text Mail

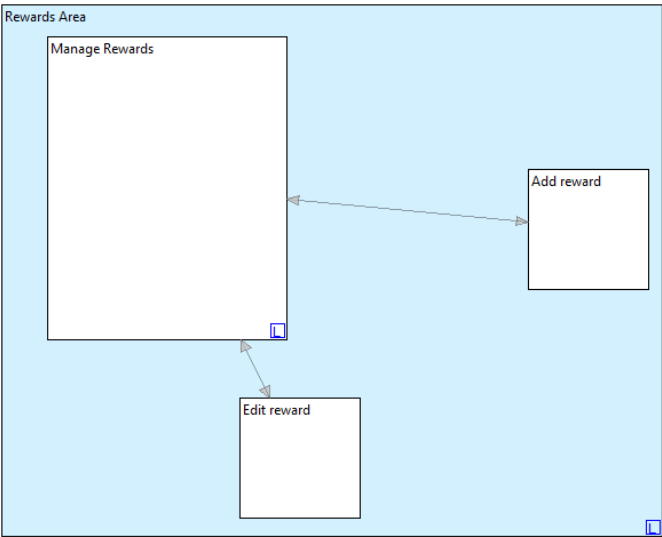
8.4.2..3 Incoming Flow Summary

→ Back	from Edit Mail / [EntryUnit] Mail	to [PowerIndexUnit] Mails
→ OKFlow104	from Notification Area / [ModuleInstanceUnit] OnChange NoOp	to Text Mail
→ OKFlow98	from Notification Area / [ModuleInstanceUnit] Update Email Text	to [PowerIndexUnit] Mails

8.4.2..4 Outgoing Flow Summary

→ Flow42	from [EntryUnit] Language	to Notification Area / [ModuleInstanceUnit] OnChange NoOp
→ Modify	from [PowerIndexUnit] Mails	to Edit Mail / [SelectorUnit] Text Email
→ Preview	from [PowerIndexUnit] Mails	to Notification Area / [ModuleInstanceUnit] Email Preview

[Area] Rewards Area



8.4.2..1 Page Summary

- 📄 Add reward
- 📄 Edit reward
- 📄 Manage Rewards

8.4.2..2 Component Summary

- ⬡ [ModuleInstanceUnit] Create New Reward
- ⬡ [ModuleInstanceUnit] Modify Reward
- ⬡ [ModuleInstanceUnit] OnChange NoOp

8.4.2..3 Landmark Summary

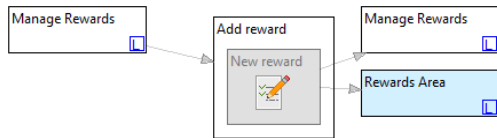
- 🏠 Rewards Area
- 📄 Manage Rewards

8.4.2..4 Incoming Flow Summary

→ Flow150	from 📄 Manage Rewards / 🗒️ [EntryUnit] Search	to ⬡ [ModuleInstanceUnit] OnChange NoOp
→ Save	from 📄 Add reward / 🗒️ [EntryUnit] New reward	to ⬡ [ModuleInstanceUnit] Create New Reward
→ Save	from 📄 Edit reward / 🗒️ [EntryUnit] Edit reward	to ⬡ [ModuleInstanceUnit] Modify Reward

8.4.2..5 Outgoing Flow Summary

→ OKFlow114	from ⬡ [ModuleInstanceUnit] OnChange NoOp	to 📄 Manage Rewards
→ OKFlow2	from ⬡ [ModuleInstanceUnit] Create New Reward	to 📄 Manage Rewards / 📄 [PowerIndexUnit] Rewards
→ OKFlow259	from ⬡ [ModuleInstanceUnit] Modify Reward	to 📄 Manage Rewards / 📄 [PowerIndexUnit] Rewards

[Page] Add reward**8.4.2..1 Component Summary**

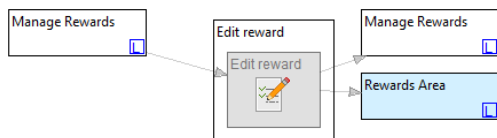
[EntryUnit] New reward

8.4.2..2 Incoming Flow Summary

→ Add reward	from Manage Rewards	to Add reward
--------------	----------------------	----------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] New reward	to Manage Rewards
→ Save	from [EntryUnit] New reward	to Rewards Area / [ModuleInstanceUnit] Create New Reward

[Page] Edit reward**8.4.2..1 Component Summary**

[EntryUnit] Edit reward

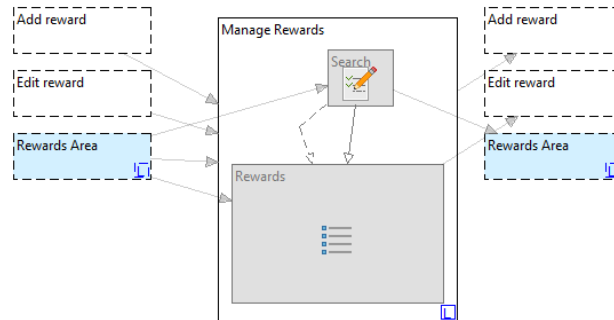
8.4.2..2 Incoming Flow Summary

→ Modify	from Manage Rewards / [PowerIndexUnit] Rewards	to [EntryUnit] Edit reward
----------	--	-----------------------------

8.4.2..3 Outgoing Flow Summary

→ Back	from [EntryUnit] Edit reward	to Manage Rewards
→ Save	from [EntryUnit] Edit reward	to Rewards Area / [ModuleInstanceUnit] Modify Reward

[Page] Manage Rewards



8.4.2..1

8.4.2..2 Component Summary

[PowerIndexUnit] Rewards

[EntryUnit] Search

8.4.2..3 Landmark Summary

Manage Rewards

8.4.2..4 Incoming Flow Summary

→ Back	from Add reward / [EntryUnit] New reward	to Manage Rewards
→ Back	from Edit reward / [EntryUnit] Edit reward	to Manage Rewards
→ OKFlow114	from Rewards Area / [ModuleInstanceUnit] OnChange NoOp	to Manage Rewards
→ OKFlow2	from Rewards Area / [ModuleInstanceUnit] Create New Reward	to [PowerIndexUnit] Rewards
→ OKFlow259	from Rewards Area / [ModuleInstanceUnit] Modify Reward	to [PowerIndexUnit] Rewards

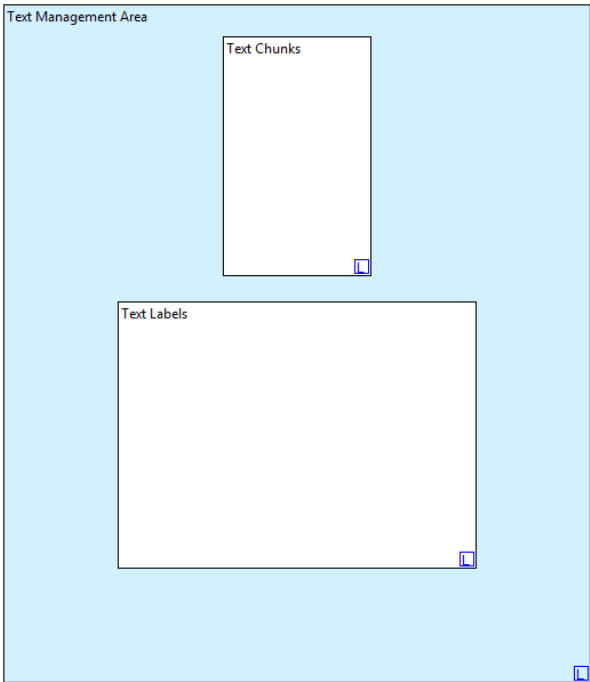
8.4.2..5 Outgoing Flow Summary

→ Add reward	from Manage Rewards	to Add reward
→ Flow150	from [EntryUnit] Search	to Rewards Area / [ModuleInstanceUnit] OnChange NoOp
→ Modify	from [PowerIndexUnit] Rewards	to Edit reward / [EntryUnit] Edit reward

8.4.2..6 Internal Flow Summary

→ Search	from [EntryUnit] Search	to [PowerIndexUnit] Rewards
----------	--------------------------	------------------------------

[Area] Text Management Area



8.4.2..1 Page Summary

- Text Chunks
- Text Labels

8.4.2..2 Component Summary

- [ModuleInstanceUnit] Edit Text Chunk
- [ModuleInstanceUnit] Modify Bundle Data
- [ModuleInstanceUnit] OnChange NoOp

8.4.2..3

8.4.2..4 Landmark Summary

- Text Management Area
- Text Chunks
- Text Labels

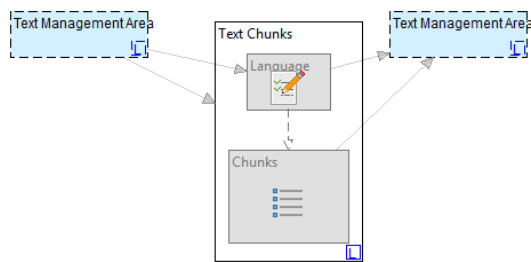
8.4.2..5 Incoming Flow Summary

→ Edit	from Text Chunks / [PowerIndexUnit] Chunks	to [ModuleInstanceUnit] Edit Text Chunk
→ Flow42	from Text Chunks / [EntryUnit] Language	to [ModuleInstanceUnit] OnChange NoOp
→ Save	from Text Labels / [MultiEntryUnit] Edit labels	to [ModuleInstanceUnit] Modify Bundle Data

8.4.2..6 Outgoing Flow Summary

→ OKFlow134	from [ModuleInstanceUnit] Edit Text Chunk	to Text Chunks / [EntryUnit] Language
→ OKFlow359	from [ModuleInstanceUnit] Modify Bundle Data	to Text Labels / [ScrollerUnit] Labels
→ OKFlow72	from [ModuleInstanceUnit] OnChange NoOp	to Text Chunks

[Page] Text Chunks



8.4.2..1 Component Summary

[PowerIndexUnit] Chunks

[EntryUnit] Language

8.4.2..2 Landmark Summary

Text Chunks

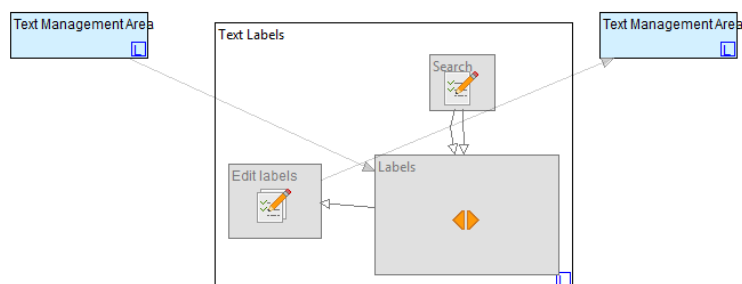
8.4.2..3 Incoming Flow Summary

→ OKFlow134	from Text Management Area / [ModuleInstanceUnit] Edit Text Chunk	to [EntryUnit] Language
→ OKFlow72	from Text Management Area / [ModuleInstanceUnit] OnChange NoOp	to Text Chunks

8.4.2..4 Outgoing Flow Summary

→ Edit	from [PowerIndexUnit] Chunks	to Text Management Area / [ModuleInstanceUnit] Edit Text Chunk
→ Flow42	from [EntryUnit] Language	to Text Management Area / [ModuleInstanceUnit] OnChange NoOp

[Page] Text Labels



8.4.2..1 Component Summary

 [MultiEntryUnit] Edit labels

 [ScrollerUnit] Labels

 [EntryUnit] Search

8.4.2..2 Landmark Summary

 Text Labels

8.4.2..3 Incoming Flow Summary

→ OKFlow359	from  Text Management Area /  [ModuleInstanceUnit] Modify Bundle Data	to  [ScrollerUnit] Labels
-------------	---	--

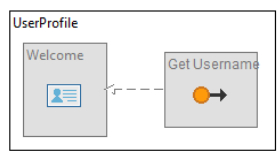
8.4.2..4 Outgoing Flow Summary

→ Save	from  [MultiEntryUnit] Edit labels	to  Text Management Area /  [ModuleInstanceUnit] Modify Bundle Data
--------	---	---

8.4.2..5 Internal Flow Summary

→ Flow64	from  [EntryUnit] Search	to  [ScrollerUnit] Labels
→ Flow65	from  [ScrollerUnit] Labels	to  [MultiEntryUnit] Edit labels
→ Search	from  [EntryUnit] Search	to  [ScrollerUnit] Labels

[MasterPage] UserProfile



8.4.2..1 Component Summary

→ [GetUnit] Get Username

[DataUnit] Welcome

8.4.2..2 Outgoing Flow Summary

→ Logout	from [DataUnit] Welcome	to Administration / [ModuleInstanceUnit] Logout
----------	-------------------------	---

8.4.3 Statistics

Structure

Entity	28(4 derived)(5 volatile)
Attribute	168(5 derived)
Attribute per Entity	6(0 derived)
Relationship	22(1 derived)
Relationship per Entity	0(0 derived)

Navigation

Site View	4
Service View	5
Module View	1
Context Parameter	13
Area	12
Area per Site View	3
Page	64
Page per Site View	16
Master Page	4
Operation Group	43
Operation Group per View	4
Port	19
Job	2
Content Module	0
Operation Module	65

 Hybrid Module	8
---	---

Content Units

Content Units	218
Content Components per  Site View	54
Content Components per  Page	3
 Details	26
 Form	33
 Get	25
 Hierarchy	16
 Simple List	5
 Input Port	2
 Module	3
 Checkable List	1
 Multiple Details	6
 Multiple Form	2
 Message	13
 View Component	8
 Output Port	6
 List	26
 Script	4
 Scroller	2
 Selector	40

Operation Units

Operation Units	638
Operation Components per  Site View	159
Operation Components per  Area	53
Operation Components per  Operation Group	14
 Adapter	5
 BLOB Utils Component	2
 Connect	7
 Create	18
 Delete	9
 Disconnect	10
 Error Response	19

 Get	8
 Init Job	2
 Input Port	71
 Is Not Null	35
 Jump	9
 KO Port	52
 Login	1
 Logout	1
 Loop	4
 Mail	1
 Update	23
 Module	97
 No Op	4
 OK Port	73
 Parameter Collector	9
 Password	1
 Query	1
 Reset	6
 Response	19
 Scale Image Unit	6
 Schedule Job	4
 Script	25
 Selector	70
 Solicit	19
 Strings Function Unit	8
 Switch	13
 Time	6