

# Energy Demand-Aware Open Services for Smart Grid Intelligent Automation

## SmartHG

EU FP7 Project #317761



## Deliverable D4.2.2

# Second Year Prototype of Grid Intelligent Automation Services

Deliverable due on : M26

Output of WP : WP4

WP Responsible : IMDEA

### Consortium

Participant Organization Name	Participant Short Name	Country
Sapienza University of Rome	UNIROMA1	Italy
Aarhus University	AU	Denmark
IMDEA Energía	IMDEA	Spain
A. V. Luikov Heat and Mass Transfer Institute of the National Academy of Sciences of Belarus	HMTI	Belarus
ATANVO GmbH	ATANVO	Germany
Panoramic Power	PANPOW	Israel
Solintel	SOLINTEL	Spain
SEAS – NVE	SEAS	Denmark
Kalundborg Municipality	KAL	Denmark
Minskenergo	MINSKENG	Belarus
Develco Products A/S	DEVELCO	Denmark

# Document Information

<b>Version</b>	June 5, 2015, 17:33
<b>Date</b>	June 5, 2015
<b>Contributors</b>	UNIROMA1, AU, IMDEA, HMTI, ATANVO, PANPOW, SEAS, DEVELCO
<b>Reason for release</b>	Second year review
<b>Dissemination level</b>	Public (PU)
<b>Status</b>	Final

<b>Project title</b>	Energy Demand-Aware Open Services for Smart Grid Intelligent Automation
<b>Project acronym</b>	SmartHG
<b>Project number</b>	317761
<b>Call (part) identifier</b>	FP7-ICT-2011-8

Work programme topic addressed	
<b>Challenge</b>	<b>6:</b> <i>ICT for a low carbon economy</i>
<b>Objective</b>	<b>ICT-2011.6.1</b> <i>Smart Energy Grids</i>
<b>Target Outcome</b>	d) Home energy controlling hubs that will collect real-time or near real-time data on energy consumption data from smart household appliances and enable intelligent automation.

<b>Project coordinator</b>	Enrico Tronci
<b>E-mail</b>	tronci@di.uniroma1.it

*To make this deliverable suitable for public dissemination, test-bed data have been anonymized.*

# Contents

<b>Executive Summary</b>	<b>1</b>
<b>1 Retrospect</b>	<b>2</b>
<b>2 Introduction</b>	<b>3</b>
2.1 Outline . . . . .	3
<b>3 GIAS Prototypes: General Architecture and Usage</b>	<b>5</b>
<b>4 GIAS Prototypes User Manual</b>	<b>9</b>
4.1 DAPP-H Web Service . . . . .	9
4.2 PPSV Web Service . . . . .	11
4.3 EVT Web Service . . . . .	15
4.4 DAPP-K Web Service and Tarball File . . . . .	15
4.5 DB&A RESTful Service . . . . .	17
<b>5 DSO–IAS Protocol Implementation</b>	<b>22</b>
<b>6 Conclusions</b>	<b>24</b>
6.1 Advancements . . . . .	24
6.2 Limitations and Future Work . . . . .	25
<b>Bibliography</b>	<b>27</b>

# List of Figures

3.1	Software architecture for WP4 services prototypes . . . . .	6
3.2	UML-like Use Cases for Software as a Service (SaaS) Grid Intelligent Automation Service (GIAS) prototypes. Outgoing arrows show which actor makes requests, ingoing arrows show which actor answers to requests . . . . .	7
3.3	UML-like Use Cases for real-time GIAS prototypes. Outgoing arrows show which actor makes requests, ingoing arrows show which actor answers to requests . . . . .	8
4.1	Login page for the DAPP-H service . . . . .	11
4.2	DAPP-H service after successful login: search an already requested execution . . . . .	11
4.3	DAPP-H service: result of execution search . . . . .	12
4.4	DAPP-H service: search an already created figure . . . . .	12
4.5	DAPP-H service: interactively visualise an already created figure . . . . .	12
4.6	DAPP-H service: requesting a new execution . . . . .	13
4.7	PPSV service: requesting a new execution . . . . .	14
4.8	PPSV service: visualising an output probability distribution . . . . .	14
4.9	EVT service: request a new execution . . . . .	15
4.10	DAPP-K service: page for download . . . . .	17
4.11	DB&A service: login page . . . . .	18
4.12	DB&A service: high-level APIs . . . . .	19
4.13	DB&A service: low-level APIs inside the high-level <b>homes</b> . . . . .	19
4.14	DB&A service: content of <b>residential_homes</b> . . . . .	20
4.15	DB&A service: how to <b>POST</b> to <b>residential_homes</b> (bottom part of Figure 4.14) . . . . .	20
4.16	DB&A service: how to <b>DELETE</b> <b>residential_homes/25/</b> . . . . .	21
5.1	Communication between the Electric Car and DR server with SEP2. . . . .	23

# List of Tables

2.1	GIAS prototypes synopsis . . . . .	3
2.2	Correspondence between SmartHG tasks inside WP4 and sections of this deliverable . . . . .	4
5.1	Use case timed scenario description of Tom's activities . . . . .	22
6.1	Limitations for the second year evaluation & how we plan to overcome such limitations in the third year evaluation. . . . .	26

# List of Acronyms

**API** Application Programming Interface

**CSV** Comma Separated Value

**DAPP** Demand Aware Price Policies

**DAPP-H** Demand Aware Price Policies for Homes

**DAPP-K** Demand Aware Price Policies for Substation-Level Energy Storage Control

**DB** DataBase

**DB&A** Database and Analytics

**DBMS** DataBase Management System

**DBService** Database Service

**DRLC** Demand Response and Load Control

**DR** Demand Response

**DSO** Distribution System Operator

**EDN** Electric Distribution Network

**ESS** Energy Storage System

**EUMF-K** Energy Usage Modelling and Forecasting for Control

**EVT** EDN Virtual Tomography

**GIAS** Grid Intelligent Automation Service

**GUI** Graphical User Interface

**HIAS** Home Intelligent Automation Service

**IAS** Intelligent Automation Service

**JSON** JavaScript Object Notation

**PHP** PHP: Hypertext Preprocessor

**PPSV** Price Policy Safety Verification

**RESTful** REpresentational State Transfer



**SaaS** Software as a Service

**UML** Unified Modeling Language

**URL** Unified Resource Location

**XML** eXtended Markup Language

# Executive Summary

The main objective of the SmartHG project is to develop effective Intelligent Automation Services (IASs) to minimise users energy bill for end residential users while optimising operation on the grid for Distribution System Operators (DSOs). This deliverable, together with Deliverable D3.2.2, describes the second year effective implementation of such IASs. Namely, in this deliverable we focus on the IASs working on the DSO side, i.e., on the Grid Intelligent Automation Services (GIASs). Instead, Deliverable D3.2.2 focuses on the Home Intelligent Automation Services (HIASs), which work on the residential user side. Following the design described in Deliverable D4.2.1, the GIASs developed in the second year are the following: the Database and Analytics (DB&A), Demand Aware Price Policies for Homes (DAPP-H), Demand Aware Price Policies for Substation-Level Energy Storage Control (DAPP-K), Price Policy Safety Verification (PPSV) and EDN Virtual Tomography (EVT) services. The implementation of the GIASs is based on the GIASs design described in Deliverable D4.2.1. To this aim, first of all we describe the overall architecture of the prototypes, which is the same as the HIASs architecture described in Deliverable D3.2.2. Then, one section is dedicated to each prototype, describing what it does and how it must be used. W.r.t. the first year version of the GIAS prototypes, the second year version of all GIAS prototypes changed not only to reflect the changes in the design (described in Deliverable D4.2.1), but also in the overall architecture. Namely, all GIAS prototypes are now accessible from a Web service, which allows either a Software as a Service (SaaS) usage (submit an execution and wait for completion) or software download (for services with real time requirements).

The effectiveness of the prototypes described here is evaluated, together with the HIAS prototypes, in the second year iteration of the SmartHG IASs evaluation (Deliverable D5.2.1). In the third and last project year, the prototypes presented here will be improved by both implementing the new improved algorithms proposed in the future version of the services design, and by tailoring such prototypes to directly work on the project test-beds in Minsk (Belarus, if available), Kalundborg (Denmark) and Central District (Israel). This will allow SmartHG IASs to be effectively released as the major final product of SmartHG project.



# Chapter 1

## Retrospect

In this section we briefly recall the main achievements obtained in the first year version of the SmartHG Grid Intelligent Automation Services (GIASs) prototype implementation, which was described in Deliverable D4.1.2. The detailed list of all advancements of the second year version of such prototypes (described in this deliverable) w.r.t. the first year version (described in first year Deliverable D4.1.2) is instead reported in Section 6.

The first year prototypes for all GIASs to be developed within the SmartHG project, namely Demand Aware Price Policies (DAPP), Price Policy Safety Verification (PPSV), EDN Virtual Tomography (EVT), plus the protocol SEP2.0 (for the communication protocol between the Distribution System Operator (DSO) and all project Intelligent Automation Services (IASs)), were developed following the specification described in Deliverable D2.1.1 and the algorithms design described in Deliverable D4.1.1. They were used to perform the first year evaluation described in Deliverable D5.1.1. Nearly all prototypes were developed on Linux using Open Source Software. The only exception was the EVT service, which was developed on Windows due to the fact that it needs the PowerWorld Simulator. Finally, the only services with a Graphical User Interface (GUI) were Database and Analytics (DB&A), in order to visualise input and output for each other SmartHG IAS, and EVT (though not publicly accessible), in order to graphically show the Electric Distribution Network (EDN) status.

The major challenges identified for the GIAS prototypes (in common with the Home Intelligent Automation Service (HIAS) prototypes) is to provide each service in a more user-friendly way, e.g., by setting up a Web service for each GIAS.

# Chapter 2

## Introduction

This deliverable describes the second year prototypes for all Grid Intelligent Automation Services (GIASs) to be developed within the second year iteration of the SmartHG project, namely Demand Aware Price Policies for Homes (DAPP-H), Demand Aware Price Policies for Substation-Level Energy Storage Control (DAPP-K), Price Policy Safety Verification (PPSV), EDN Virtual Tomography (EVT) and Database and Analytics (DB&A). The second year version of the algorithms for the above services have been described in Deliverable D4.2.1. Moreover, these prototypes are used for the evaluation phase described in D5.2.1.

Table 2.1 shows the main properties of the prototypes discussed in this deliverable. For all services, a Web service has been developed and deployed. For the DAPP-H, EVT and PPSV services, a Software as a Service (SaaS) Web service has been developed. A user, after authentication, may interact with the corresponding service through such Web service, by uploading inputs, requesting executions and retrieving outputs. For the DAPP-K service, which has real-time constraints, the Web service only allows to download the software and the documentation, upon authentication. Finally, for the DB&A service, which targets communications between other services, only the REpresentational State Transfer (RESTful) service (with its default Web pages) has been implemented and deployed.

Prototype	Web	SaaS	Software Download	RESTful
DAPP-H	Yes [1]	Yes	No	Yes [2]
DAPP-K	Yes [3]	No	Yes	Authentication only [4]
PPSV	Yes [5]	Yes	No	Yes [6]
EVT	Yes [7]	Yes	No	Yes [8]
DB&A	Yes [9]	No	No	Yes [9]

Table 2.1: GIAS prototypes synopsis. A Web page collecting all links above is here [10]

### 2.1 Outline

This deliverable is organised as follows. Section 3 shows the common architecture on which all second year GIAS prototypes relies upon. Then, Section 4 describes, for each GIAS, the Web service interface which has been developed in the second year. Moreover, Section 5 also describes an implementation and a test case for the SEP2.0 protocol used

Table 2.2: Correspondence between SmartHG tasks inside WP4 and sections of this deliverable

Task	Task Name	Sections
T4.1	Design and Development of DB&A	Section 4.5
T4.2	Design and Development of energy DAPP service	Sections 4.1 and 4.4
T4.3	Design and Development of the EVT service	Section 4.3
T4.4	Design and Development of the PPSV service	Section 4.2
T4.5	Design and Development of open standard Internet based communication between DSO and IASs	Section 5

for communication between DSO and IASs. The overall results of this deliverable are summarised in Section 6. Furthermore, Section 6 describes in detail the advancements of this year GIAS prototypes w.r.t. the first year prototypes of the same services, discusses the current limitations and plans future work. Finally, Table 2.2 shows the correspondence between SmartHG tasks inside WP4 and sections of this deliverable.

## Chapter 3

# GIAS Prototypes: General Architecture and Usage

The goal of tasks 4.1, 4.2, 4.3, 4.4 and 4.5 of WP4 is to design, implement and deploy the SmartHG Grid Intelligent Automation Services (GIASs), i.e., the services to be proposed to the Distribution System Operator (DSO). The design of such GIASs is described in Deliverable D4.2.1. In this deliverable, instead, we focus on implementation and deployment. To this aim, first of all we categorise the GIASs as follows.

**Infrastructure services** As in Deliverable D5.2.1, the only infrastructure service in SmartHG is the Database and Analytics (DB&A) service, which collects information (e.g., energy consumption data) from residential homes and provides them to the other services.

**Software as a Service (SaaS)** These are services which do not have real-time constraints, and thus may be deployed as Web services. This allows us to provide such services without requiring installation of any additional software, as a standard Internet browser is sufficient. Such Web services must allow users (i.e., DSOs) to upload inputs, request an execution, and download (and visualise, if applicable) outputs. The GIASs which fall in this category are EDN Virtual Tomography (EVT), Demand Aware Price Policies for Homes (DAPP-H) and Price Policy Safety Verification (PPSV).

**Real-time services** These are services which control some device (or serve as an auxiliary service to control some device). The only real-time GIAS is the Demand Aware Price Policies for Substation-Level Energy Storage Control (DAPP-K) service, which every  $t$  minutes (1 hour in the evaluation shown in Deliverable D5.2.1) must compute a charge/discharge action for an Energy Storage System (ESS) installed on an Electric Distribution Network (EDN) substation. Real-time services cannot be invoked from a Web interface (as this the real-time requirements will fail), thus they must be downloaded, installed and run from a local host owned by the user.

Given this, we designed the GIAS prototypes to have a common architecture, depicted in Figure 3.1. Such architecture is based on four different modules, thus resulting in a four-tier architecture:

- A relational database, implemented by the PostgreSQL DataBase Management System (DBMS).

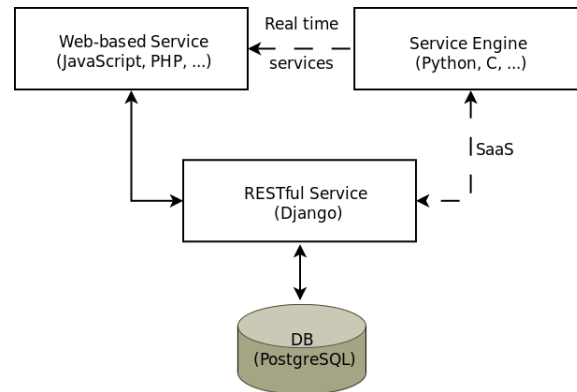


Figure 3.1: Software architecture for real-time services and for SaaS. Communications which take place for both SaaS and real-time services are shown with arrows. Dotted arrows show communications which take place either only for SaaS or only for real-time service (to allow service download). For the DB&A service, only the first two layers are used.

- A REpresentational State Transfer (RESTful) service, implemented on the top of the relational database by the Django framework. In the case of the DB&A service, which in this second year iteration is designed to be accessed only by other services and not by a human user, the prototype only consists in these two modules.
- A Web service, which offers a human interface to the GIAS, written either in the JavaScript language or in the PHP: Hypertext Preprocessor (PHP) language.
- The actual GIAS engine, which directly implements the corresponding algorithm for the GIAS outlined in Deliverable D4.2.1. For such engine, we have the following:
  - For SaaS GIASs (i.e., EVT, DAPP-H and PPSV), such engine is executed on the service provider servers.
  - For real-time GIASs (i.e., DAPP-K), such engine consists in a tarball file containing Linux source files to be downloaded, compiled, installed and run on a user host. In this case, there is no communication between the service engine and the RESTful service.

The use cases of such architecture are depicted in Figures 3.2 (for SaaS services) and 3.3 (for real-time services). Namely, Figure 3.2 shows that SaaS services must allow both a human user and other SmartHG Intelligent Automation Services (IASs) to submit an execution request, and see the result when it is completed. This entails that each SaaS GIAS prototype must allow both a human user and other SmartHG IASs to upload the complete input of an execution request, see the current status of an execution (“pending”, “running”, “failed”, or “complete”), and download the output when the execution has been completed. For human users, the output is also graphically depicted (this has been implemented only for DAPP-H and PPSV). As a consequence, each actor behaves as follows:

**Web service** The Web service has a twofold role: it both accepts requests from human users and generate requests for the RESTful service. More in detail:

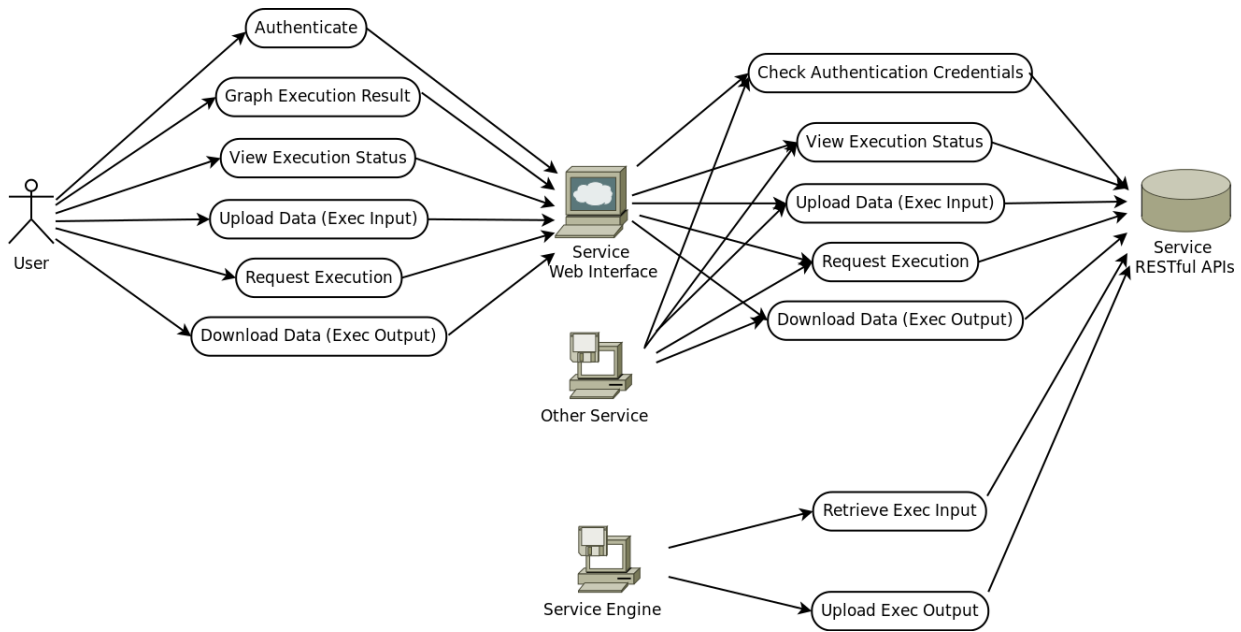


Figure 3.2: UML-like Use Cases for SaaS GIAS prototypes. Outgoing arrows show which actor makes requests, ingoing arrows show which actor answers to requests

- It accepts the following requests from a human user: authentication, upload input data for an execution, submit an execution request once all input data has been uploaded, check executions status (which includes retrieving execution current log), download output data of a given execution (only if the execution has been completed), depict output of a given execution in a graphical way (only if the execution has been completed). Note that executions input is typically organised in scenarios, in order to allow to submit multiple executions on the same input while only changing some tuning parameters.
- All requests from a human user are translated in **GET** or **POST** requests for the RESTful service. The only exception is for depicting in a graphical way an execution result, which is directly managed by the Web service itself (possibly after **GET**ting data from the RESTful service).

**Service engine** The service engine is divided in two parts:

1. An *execution requests monitor*, which in an endless loop checks, every 60 seconds, if there is a new (“pending”) execution request. Once a new execution is found, the monitor launches the service computation engine (see point below) and waits for its completion (thus the execution status goes through “running” to either “completed” or “failed”). Once the service engine terminates, the monitor resumes its endless loop.
2. The *service computation engine*, which requires, as an argument, the identifier  $i$  of an execution. Given  $i$ , the service computation engine downloads from the RESTful service all the input required by  $i$  (depending on the service, the input may be stored either in a local DataBase (DB) or in a collection of files), executes the service algorithm in order to obtain the corresponding output, and finally uploads such output to the RESTful service.

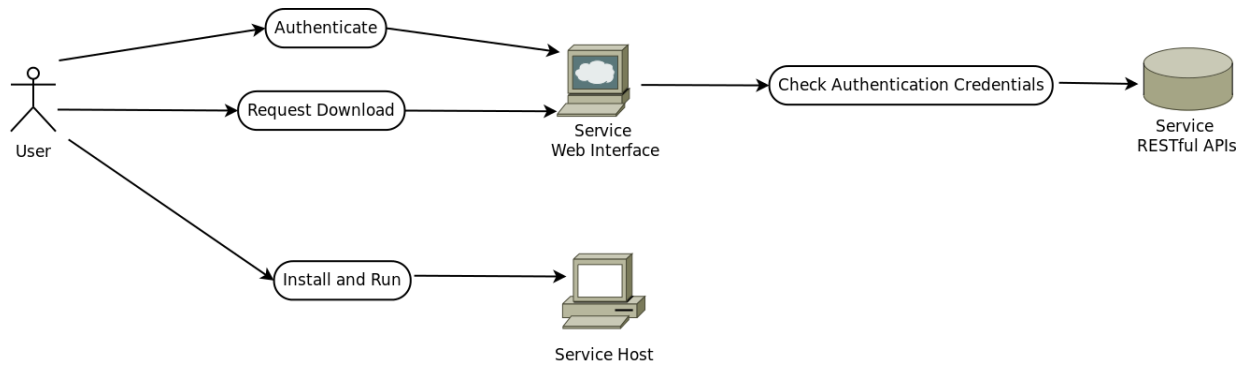


Figure 3.3: UML-like Use Cases for real-time GIAS prototypes. Outgoing arrows show which actor makes requests, ingoing arrows show which actor answers to requests

**RESTful service and relational DB** The RESTful service is implemented on the top of the relational DB by the Django framework. Namely, the relational DB cannot be directly accessed by the other services, which instead must interact with the RESTful service. As a result, all queries to the relational DB are accessible through Application Programming Interfaces (APIs), which are represented as verbs (the most important ones are **GET** for getting data, **POST** to create new data, and **PUT** to update data) followed by an Unified Resource Location (URL). As a response, the RESTful service provides a JavaScript Object Notation (JSON) string.

On the other hand, Figure 3.3 shows that for real-time services the interaction is much simpler: the user has to authenticate to the Web service, from which a tarball archive file may be downloaded. Such file must be copied on a Linux machine owned by the user. After archive extraction, the service may be directly launched through a wrapper script (which also handles compilation). Input is provided in files to be supplied by the user (examples for the file format are available in the tarball archive file).

## Chapter 4

# GIAS Prototypes User Manual

In this section we discuss in detail each Grid Intelligent Automation Service (GIAS) from the users perspective, i.e., i) for the Software as a Service (SaaS) services, we describe the Web service, ii) for the real-time Demand Aware Price Policies for Substation-Level Energy Storage Control (DAPP-K) service, we describe both how to download it and how to launch an execution on a local machine, and iii) for the infrastructure service Database and Analytics (DB&A) we describe the main REpresentational State Transfer (RESTful) Application Programming Interfaces (APIs).

### 4.1 DAPP-H Web Service

In this section, we describe the Demand Aware Price Policies for Homes (DAPP-H) Web service [1], which shares many features with the Web services of the other SaaS GIASs. For the design of the service and the nomenclature used, we refer the reader to Deliverable D4.2.1. First of all, the user must authenticate on a login page (Figure 4.1). Upon successful authentication, by choosing a link in the left sidebar (see Figures 4.2–4.6) the DAPP-H Web service allows the user to perform the following actions.

**New Execution** Request a new execution (see Figure 4.6). In order to submit an execution request, a complete input must be provided. To this aim, the input is organised in two *scenarios*:

- The *substation scenario* contains all substation-related information required by DAPP-H. That is, the substation scenario is a collection of  $(t, P_s(t))$  pairs, where  $t$  is a time-slot (represented by the starting and ending time-stamps) and  $P_s(t)$  is the desired substation maximum aggregated demand (in kW) in  $t$ .
- The *home scenario* contains all homes-related information required by DAPP-H. That is, the home scenario is a collection of  $(u, P_u, C_u, t, p_u(t), d_u(t))$  tuples, where  $u$  is the identifier of an home connected to the substation,  $P_u$  (resp.,  $C_u$ ) is the electricity production (resp., consumption) contract for user  $u$ ,  $t$  is a time-slot, and  $p_u(t) < P_u$  (resp.,  $d_u(t) < C_u$ ) is the power production (resp., consumption) of electricity (in kW) of user  $u$  in time-slot  $t$ .

In order to create a new execution request, it is possible to both choose already uploaded scenarios (i.e., scenarios already present in the RESTful service), or to create new scenarios. If already present scenarios are chosen, it is possible to check



the actual profiles contained in the selected scenarios by downloading them as text files. On the other hand, to create new scenarios it is sufficient to upload a file describing both the home and the substation scenario. The other DAPP-H inputs which may be chosen in this form are the following.

- Starting date of the execution validity, that is, from which time-stamp the price policies output by DAPP-H will be applied. In this second year iteration, choosing a time-stamp  $t$  for such starting date will also set the ending date to  $t$  plus one day.
- The fixed length (in minutes) of the input and output time-slots.
- The maximum speed  $\beta$  for users flexibility (in kW). If we model users flexibility as an Energy Storage System (ESS), then  $\beta$  is the maximum charge/discharge speed (i.e., in one hour the ESS may be charged/discharged at most  $\beta$  kWh).

**Submitted Executions** Search and retrieve for already submitted executions (default choice after login, see Figure 4.2). For all execution, it is possible to check status and other information (see figure 4.3). For completed executions, it is possible to download both the complete input and the complete output. Namely, the service creates and allows the user download a zip file for the input, containing two text files representing the home and the substation scenario (again, see Sections 1.1 and 1.2 for the format), and one zip file for the output, containing the following 4 files:

- one file for the output price policies (see Section 1.4 for the format); thus, the file must contain a collection of  $(u, t, P_u^-(t), P_u^+(t))$ , where  $u$  and  $t$  are the same as in the home scenario (see above), and  $P_u^-(t), P_u^+(t)$  define the low tariff area for  $u$  in time-slot  $t$ ;
- one file for forecasting (computed by the same algorithm of Energy Usage Modelling and Forecasting for Control (EUMF-K)); such file has the same format of the home scenario file (see Sections 1.1), and provides, for each time-slot, the forecast for the consumption and the forecast for the power usage (in kW) and the forecast for the power production (in kW);
- one file for DAPP-H collaborative profiles (see Section 1.5 for the format); thus, the file must contain a collection of  $(u, t, \tilde{d}_u(t), a_u(t), b_u(t))$ , where  $u$  and  $t$  are the same as in the home scenario (see above),  $a_u(t)$  is the charge/discharge action in  $t$  (in kW) for the ESS representing user  $u$  flexibility,  $b_u(t)$  is the remaining capacity in  $t$  (in kWh) for the ESS representing user  $u$  flexibility, and  $\tilde{d}_u(t)$  is the resulting power demand after charge/discharge action  $a_u(t)$  (i.e., if  $d_u(t)$  is the forecasted demand in  $t$ , then  $\tilde{d}_u(t) = d_u(t) + a_u(t)$ );
- one file containing the required capacities for the ESSs representing users flexibility (see Section 1.6 for the format); thus, the file must contain a collection of  $(u, B_u(t))$ , where  $B_u(t)$  is the maximum capacity (in kWh) for the ESS representing user  $u$  flexibility.

**New Scenario** Submit a new home and/or a new substation scenario (without having to submit an execution too). Such combined scenario is provided as a text file, using the same syntax used for the “New Execution” task (see Section 1.3 for the format).

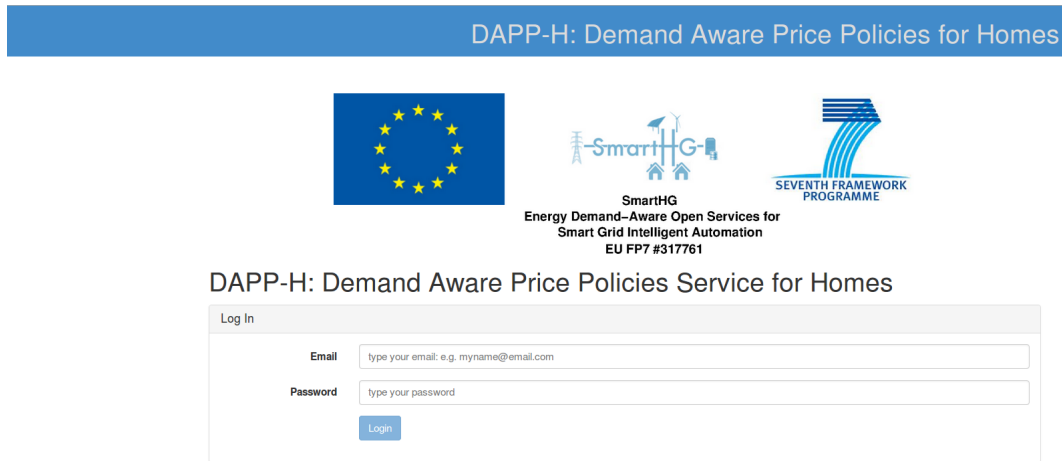


Figure 4.1: Login page for the DAPP-H service



Figure 4.2: DAPP-H service after successful login: search an already requested execution

**Figures** Search and retrieve already created figures (see Figure 4.4). Each retrieved figure may be interactively visualised (see Figure 4.5).

**New Figure** Submit a new figure, using figure templates. Currently, there is only one figure template, which allows the user to visualise in one graph the historical demand and the DAPP-H collaborative profile for a single user. When creating a new figure, the time period and the home to be visualised must be selected.

## 4.2 PPSV Web Service

In this section, we describe the Price Policy Safety Verification (PPSV) Web service [5]. For the design of the service and the nomenclature used, we refer the reader to Deliverable D4.2.1. As for the other GIASs, it is necessary to authenticate to a login page which is similar to the DAPP-H login page already shown in Figure 4.1. Upon authentication, by



Figure 4.3: DAPP-H service: result of execution search



Figure 4.4: DAPP-H service: search an already created figure

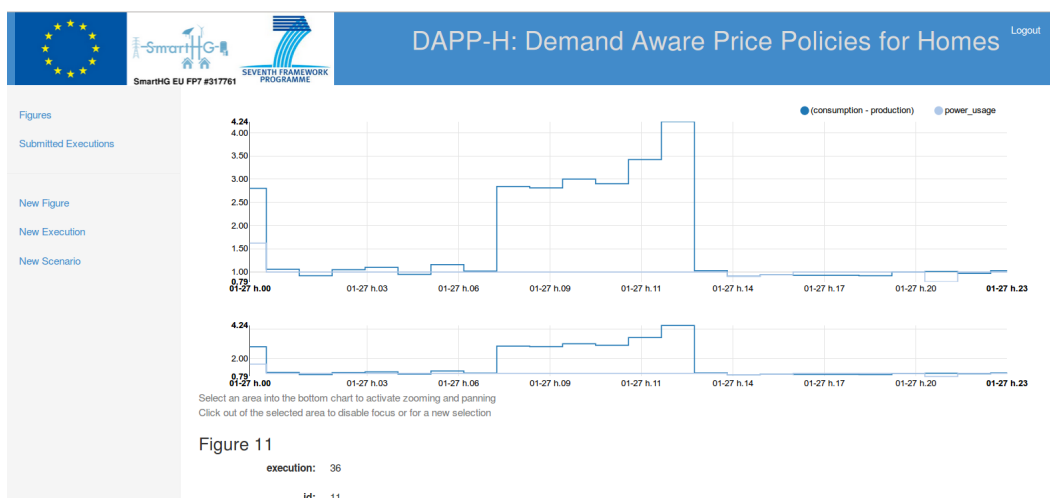


Figure 4.5: DAPP-H service: interactively visualise an already created figure

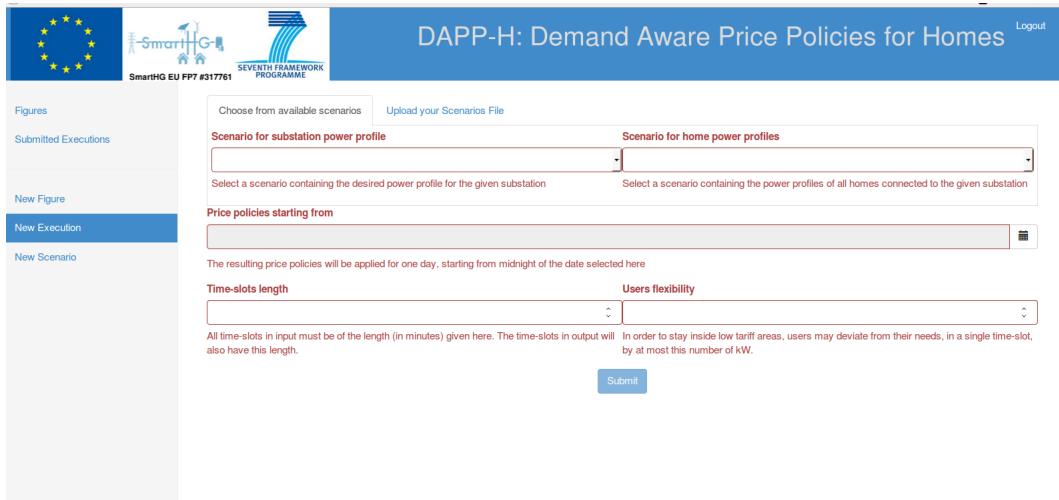


Figure 4.6: DAPP-H service: requesting a new execution

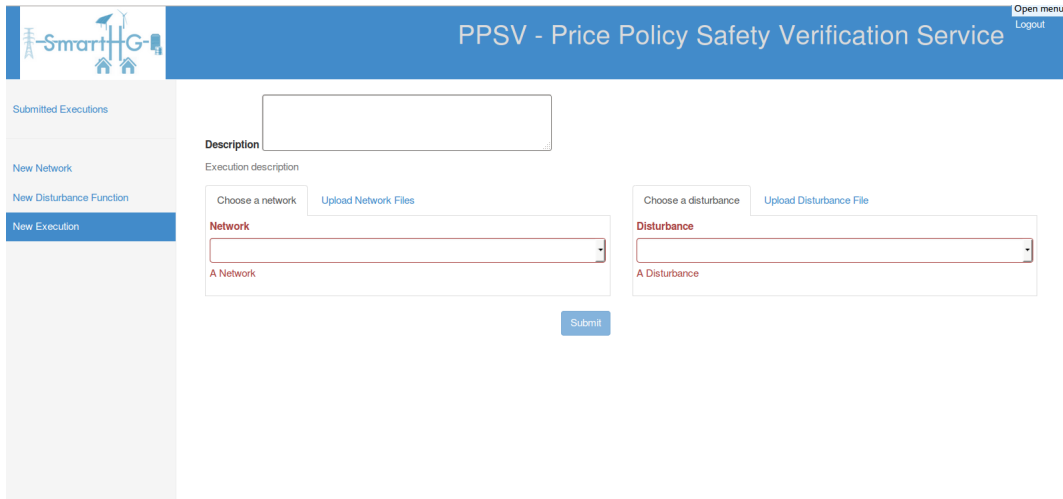
choosing a link in the left sidebar (again, the layout is the same as the one in DAPP-H), the PPSV Web service allows the user to perform the following actions.

**New Execution** Request a new execution (see Figure 4.6). In order to submit an execution request, a complete input must be provided. To this aim, the input is organised in two *scenarios*:

- The *network scenario* contains all Electric Distribution Network (EDN)-related information required by PPSV. Namely, the network scenario is a collection of *network components*, each composed by a DAPP-H substation and home scenarios (see Section 4.1). That is, for each substation  $s$  to be considered in the PPSV execution, the network scenario contains both: i) a DAPP-H substation scenario, representing the desired maximum aggregated demand, and ii) a DAPP-H home scenario, representing the power profiles for all homes connected to  $s$  (in the typical usage of PPSV, these profiles are the DAPP-H collaborative profiles output by DAPP-H). Finally, for each network component it is necessary to select which substation feeder of the Kalundborg EDN it refers, in order to enable communications with EDN Virtual Tomography (EVT).
- The *disturbance scenario* contains the disturbance model required by PPSV. That is, the disturbance scenario is a collection of (disturbance, probability) pairs, such that all probabilities sum to 1.

In order to create a new execution request, it is possible to both choose already uploaded scenarios (i.e., scenarios already present in the RESTful service), or to create new scenarios. If already present scenarios are chosen, it is possible to check the actual profiles contained in the selected scenarios by downloading them as text files (see Sections 2.1 and 2.2 for the format). On the other hand, to create new scenarios it is sufficient to upload a file for each scenario (with the same formats as above). It is also possible to upload a file for one scenario, and select an already uploaded scenario for the other.

**Submitted Executions** Search, retrieve and check status for already submitted executions (default choice after login). For completed executions, it is possible to



The screenshot shows the 'PPSV - Price Policy Safety Verification Service' interface. On the left is a sidebar with navigation links: 'Submitted Executions', 'New Network', 'New Disturbance Function', and 'New Execution' (highlighted in blue). The main area contains a form for 'Execution description'. It includes a 'Description' text box, a 'Choose a network' dropdown menu (showing 'A Network'), an 'Upload Network Files' button, a 'Choose a disturbance' dropdown menu (showing 'A Disturbance'), an 'Upload Disturbance File' button, and a 'Submit' button. In the top right corner, there are links for 'Open menu' and 'Logout'.

Figure 4.7: PPSV service: requesting a new execution

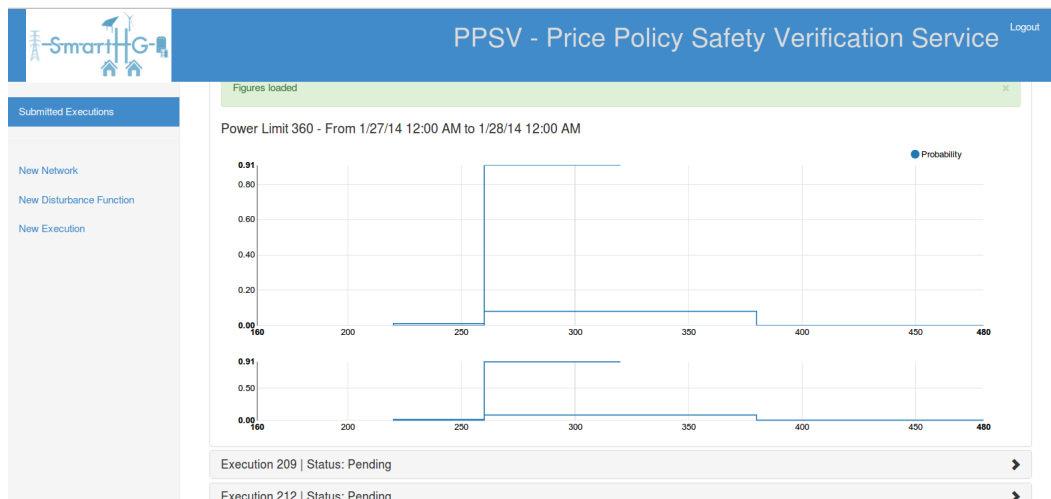


Figure 4.8: PPSV service: visualising an output probability distribution

download both the complete input and the complete output. Namely, the service creates and allows the user download a zip file for the output, containing the probability distribution on the aggregated demand output by PPSV (see Sections 2.3 for the format), and one zip file for the input, containing the text files for the network and the disturbance scenario (with the same format used for uploading them in the “New Execution” task, see Sections 2.1 and 2.2). Finally, for complete executions it is possible to graph the output probability distributions (see Figure 4.8).

**New Network** Submit a new network scenario (without having to submit an execution too). Such scenario is provided as a text file, using the same syntax used for the “New Execution” task (see Section 2.1 for the format).

**New Disturbance** Submit a new disturbance scenario (without having to submit an execution too). Such scenario is provided as a text file, using the same syntax used for the “New Execution” task (see Section 2.2 for the format).

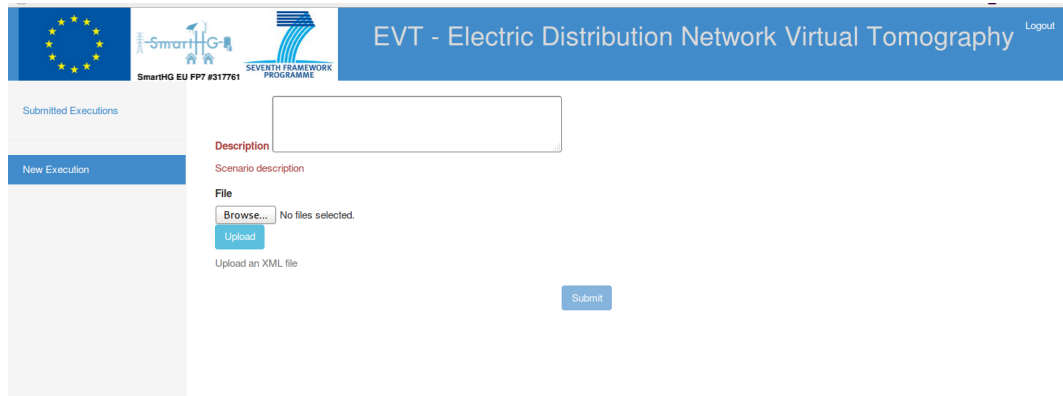


Figure 4.9: EVT service: request a new execution

### 4.3 EVT Web Service

In this section, we describe the EVT Web service [7]. For the design of the service and the nomenclature used, we refer the reader to Deliverable D4.2.1. As for the other GIASs, it is necessary to authenticate to a login page which is similar to the DAPP-H login page already shown in Figure 4.1. Upon authentication, by choosing a link in the left sidebar (again, the layout is the same as the one in DAPP-H) the EVT Web service allows the user to perform the following actions.

**New Execution** Submit a new execution (see Figure 4.9). In order to submit an execution request, a complete input must be provided. To this aim, the input is organised in single *EVT scenarios*, where each EVT scenario is simply a collection of eXtended Markup Language (XML) files to be provided as input to the PowerWorld simulator. Thus, in order to submit a new execution, it is sufficient to upload, one at a time, the XML input files. The output will consist in an XML file for each input XML file. The format for each of such XML file is described in Section 3.

**Submitted Executions** Search, retrieve and check status for already submitted executions (default choice after login). For completed executions, it is possible to download both the complete input and the complete output (as zip files containing XML files, see above).

### 4.4 DAPP-K Web Service and Tarball File

In this section, we describe both the DAPP-K Web service [3] and the DAPP-K tarball archive file which may be downloaded from [3]. For the design of the service and the nomenclature used, we refer the reader to Deliverable D4.2.1. As for the Web service, it is necessary to authenticate to a login page which is similar to the DAPP-H login page already shown in Figure 4.1. Upon authentication, there is only one available action: to download the DAPP-K tarball file, namely `dapp_k-1.0.tgz` (see Figure 4.10). Note that the user manual for the DAPP-K tarball file (namely `dapp_k-1.0.pdf`) is also available for download.

Once `dapp_k-1.0.tgz` has been downloaded on a Linux machine, it is possible to unzip it by typing on a terminal the following command:

```
tar xzf dapp_k-1.0.tgz; cd dapp_k-1.0
```

This creates a new directory named `dapp_k-1.0`, with the following directories: `aux_files`, `example_files` and `src`. Moreover, an explanatory `README.txt` file is provided, together with a wrapper Bash script `launch.sh`.

In order to run DAPP-K, the main requirement is to have either CPLEX or GLPK installed on the machine (i.e., either the command `cplex` or the command `glpsol` must be available in the system or user `PATH`). Given this, the user may directly launch the Bash script:

```
bash launch.sh
```

This will launch DAPP-K with default settings, which will reproduce the 500 kWh – 80 kW experiment described in Deliverable D5.2.1. In order to customise the input, the `launch.sh` script accepts the following command line arguments (note they are all optional: when an argument is not given, the default is used).

- h: prints an help message with all arguments and defaults. Des not run DAPP-K.
- l *h*: uses *h* as the number of hours to be forecasted for each charge/discharge action computation (default is 6).
- f *f*<sub>1</sub>: uses *f*<sub>1</sub> as the Comma Separated Value (CSV) file with the aggregated demand history (default is `example_files/profile.csv`, which may be used as an example for the format).
- pr *f*<sub>2</sub>: uses *f*<sub>2</sub> as the CSV file with the energy market prices for Distribution System Operators (DSOs) (default is `example_files/ELSPOT.DK2.csv`, which may be used as an example for the format). Note that this file is not downloaded from the Internet as stated in the design (see Deliverable D4.2.1), in order to speed up execution.
- bc *Q*: uses *Q* as the ESS capacity (i.e., maximum energy storage) in kWh (default is 500).
- br *R*: uses *R* as the ESS maximum charge/discharge rate in kW (default is 80).
- bde  $\alpha$ : uses  $\alpha \in [0, 1]$  as the ESS efficiency coefficient for discharging (default is 0.82). That is, if a discharge action *a* is sent to the ESS for 1 hour, then the ESS discharges of *a* kWh, but the energy provided is *aα* instead of *a* kWh.
- bce  $\beta$ : uses  $\beta \in [0, 1]$  as the ESS efficiency coefficient for charging (default is 0.98). That is, if a charge action *a* is sent to the ESS for 1 hour, then the energy provided to the ESS is *a* kWh, but the ESS charges of *aβ* kWh instead of *a* kWh.
- wst *w*: uses *w* as the length, in hours, of the time sliding window on which the substation aggregated demand is averaged (default is 24).
- bst *P*<sub>s</sub>: uses *P*<sub>s</sub> as the desired substation power threshold for aggregated demand in kW, when averaged on the past *w* hours (default is 360).
- p *p*: uses *p* as the number of days in the past to be used for forecast (default is 10).



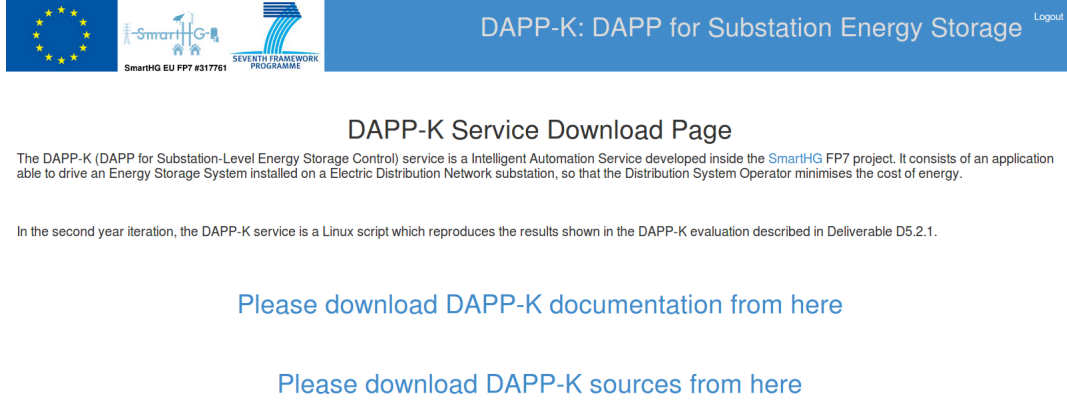


Figure 4.10: DAPP-K service: page for download

-pd  $p_d$ : uses  $p_d$  as the discounting factor for the days in the past. Format is  $x_1:\dots:x_p$ , and  $\sum_{i=1}^n x_i = 1$  must hold (default is  $\frac{1}{2}:\frac{1}{2^2}:\dots:\frac{1}{2^9}:\frac{1}{2^9}$ ).

Finally, the output of DAPP-K is the log of the decided actions and their effect on the resulting demand at the substation level. Such output is stored in a CSV file named `results.csv`, which is stored in an automatically created directory named `results/ $\tilde{t}$ /output/`, being  $\tilde{t}$  the time-stamp at which `launch.sh` was started. Namely, `results.csv` contains the following information, for each time-slot in the execution (note that DAPP-K actually starts to compute charge/discharge actions only after  $24p$  hours):

- starting and ending time-stamps of the current time-slot  $t$ ;
- aggregated demand  $d(t)$  (in kW) read from the grid, i.e., without using DAPP-K;
- ESS action  $a(t)$  computed by DAPP-K for  $t$  (in kW),
- aggregated demand  $\tilde{d}(t)$  (in kW) using DAPP-K (that is, after ESS action application), i.e., either  $\tilde{d}(t) = d(t) + a(t)$  (if  $a(t) \geq 0$ ) or  $\tilde{d}(t) = d(t) + a(t)\alpha$  (otherwise);
- ESS remaining capacity  $b(t)$  (in kWh), i.e., either  $b(t) = b(t-1) + \tau a(t)\beta$  (if  $a(t) \geq 0$ ) or  $b(t) = b(t-1) + \tau a(t)$  (otherwise), being  $\tau$  the length of time-slot  $t$ ;
- cost of energy  $c(t)$  (in EUR/kWh);
- cost of demand without ESS (in EUR), i.e.,  $c(t)d(t)$ ;
- cost of demand with ESS (in EUR), i.e.,  $c(t)\tilde{d}(t)$ .

## 4.5 DB&A RESTful Service

In this section, we describe the DB&A RESTful service [9, 11]. It is possible to access the DB&A RESTful service in two ways:



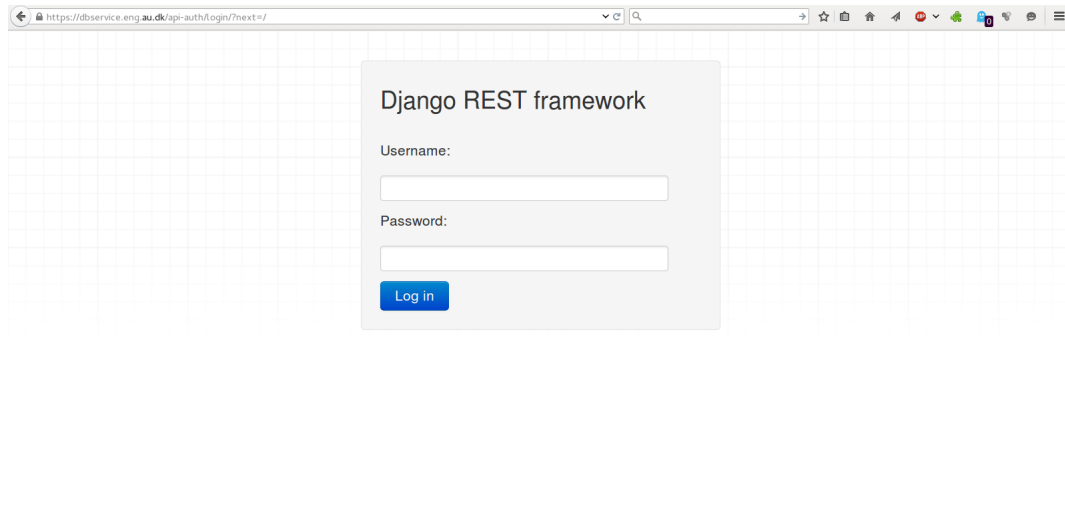


Figure 4.11: DB&A service: login page

1. Through the Web interface automatically created by Django. First of all, an authentication is required, by providing username and password to a login page (see Figure 4.11). Then, the list of the two high-level APIs provided by DB&A (namely, **users** and **homes**) is shown, where each API is a Web link (see Figure 4.12). By selecting one of such API links, the low-level underlying APIs are shown (see Figure 4.13). If again one of such links is selected, then the first page of the corresponding content is shown (e.g., by selecting **residential\_homes** inside **homes**, the first 100 residential homes are shown, e.g., see Figure 4.14). In this case, it is possible to insert a new object by filling the fields at the bottom of the page and then clicking on the **POST** button (see Figure 4.15). Furthermore, by selecting the identifier of a record in the page content, the corresponding record alone is shown. In this case, it is possible to delete the object (button **DELETE**) or to change some field of it (button **PUT**), as shown in Figure 4.16. Finally, more complex APIs (e.g., filtering) may be invoked by directly writing the corresponding Unified Resource Location (URL) in the browser address bar, using the standard syntax for RESTful services (e.g., `https://dbservice.eng.au.dk/homes/residential_homes/?country=denmark` will retrieve all homes in Denmark).
2. All DB&A APIs which may be invoked through the Web interface (with any method, i.e., **GET**, **POST**, **PUT**, **DELETE**, **OPTIONS**) are also accessible through the `curl` application. However, in this case it is necessary to specify, for each `curl` invocation, the user access token for authentication. Moreover, answers are provided as JavaScript Object Notation (JSON) objects, rather than Web pages.

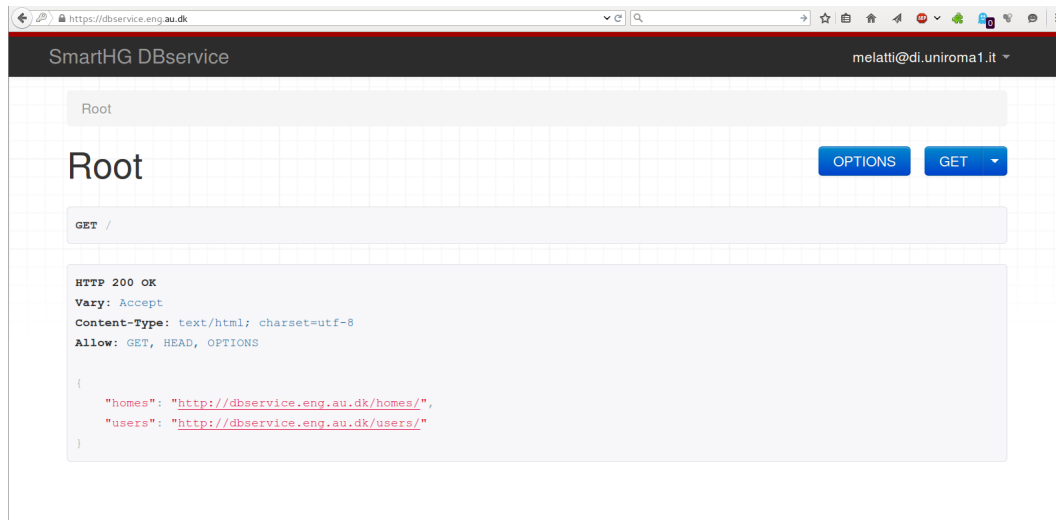


Figure 4.12: DB&A service: high-level APIs

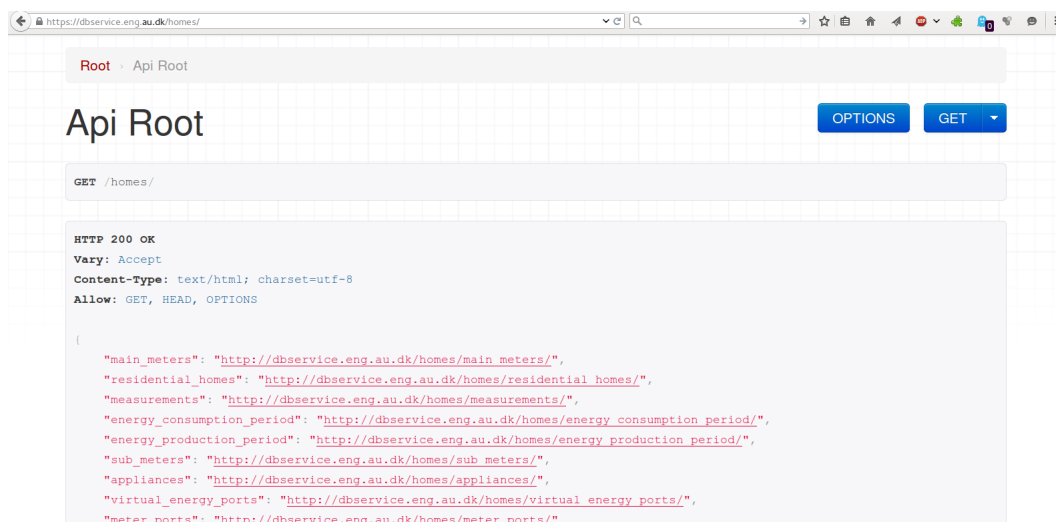
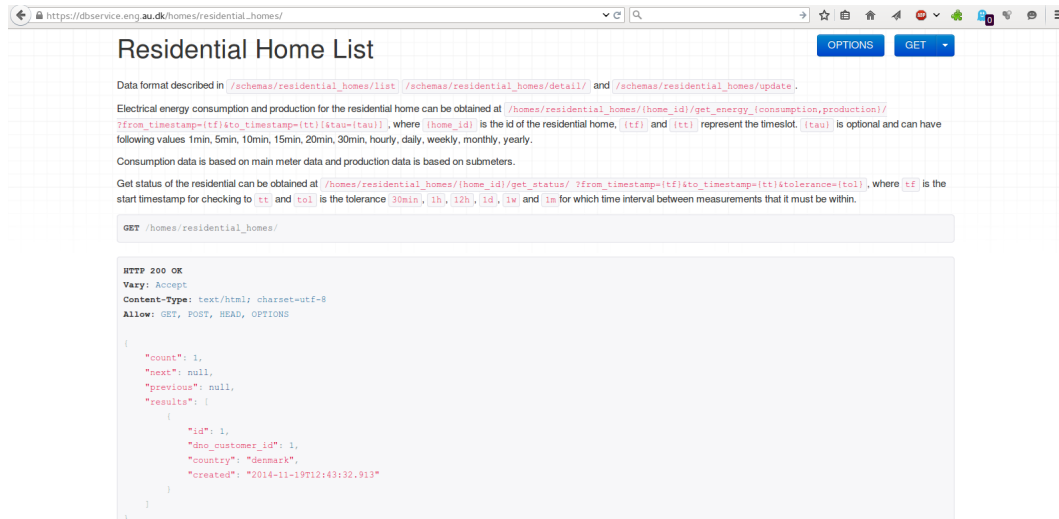


Figure 4.13: DB&A service: low-level APIs inside the high-level homes



**Residential Home List**

Data format described in [/schemas/residential\\_homes/list](/schemas/residential_homes/list), [/schemas/residential\\_homes/detail/](/schemas/residential_homes/detail/) and [/schemas/residential\\_homes/update](/schemas/residential_homes/update).

Electrical energy consumption and production for the residential home can be obtained at [/homes/residential\\_homes/{home\\_id}/get\\_energy\\_\(consumption,production\)/?from\\_timestamp={tf}&to\\_timestamp={tt}&tau={tau}](/homes/residential_homes/{home_id}/get_energy_(consumption,production)/?from_timestamp={tf}&to_timestamp={tt}&tau={tau}), where {home\_id} is the id of the residential home, {tf} and {tt} represent the timeslot. {tau} is optional and can have following values 1min, 5min, 10min, 15min, 20min, 30min, hourly, daily, weekly, monthly, yearly.

Consumption data is based on main meter data and production data is based on submeters.

Get status of the residential can be obtained at [/homes/residential\\_homes/{home\\_id}/get\\_status/?from\\_timestamp={tf}&to\\_timestamp={tt}&tolerance={tol}](/homes/residential_homes/{home_id}/get_status/?from_timestamp={tf}&to_timestamp={tt}&tolerance={tol}), where {tf} is the start timestamp for checking to {tt} and {tol} is the tolerance 30min, 1h, 12h, 1d, 1w and 1m for which time interval between measurements that it must be within.

GET /homes/residential\_homes/

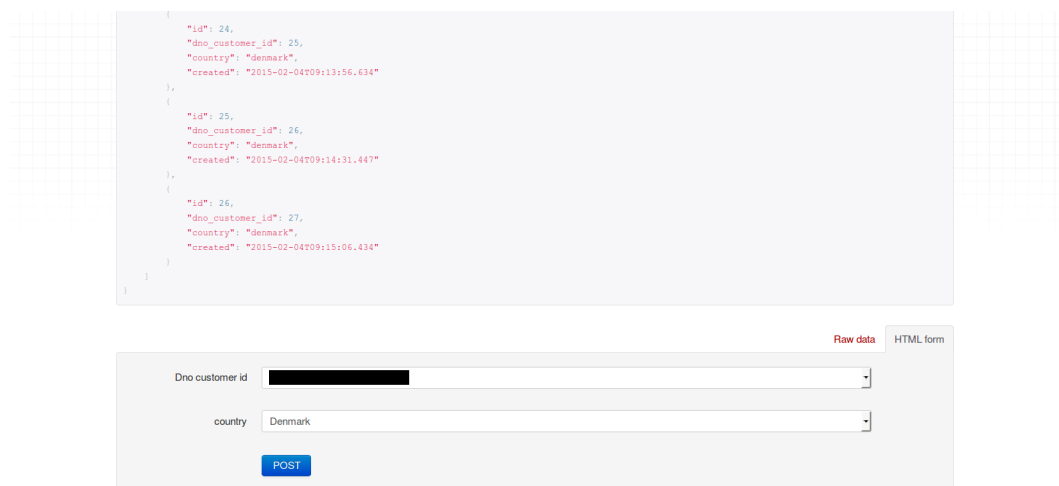
```

HTTP 200 OK
Vary: Accept
Content-Type: text/html; charset=utf-8
Allow: GET, POST, HEAD, OPTIONS

{
  "count": 1,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 1,
      "dno_customer_id": 1,
      "country": "denmark",
      "created": "2014-11-19T12:43:32.913"
    }
  ]
}

```

Figure 4.14: DB&A service: content of residential\_homes



```

{
  "id": 24,
  "dno_customer_id": 25,
  "country": "denmark",
  "created": "2015-02-04T09:13:56.634"
},
{
  "id": 25,
  "dno_customer_id": 26,
  "country": "denmark",
  "created": "2015-02-04T09:14:31.447"
},
{
  "id": 26,
  "dno_customer_id": 27,
  "country": "denmark",
  "created": "2015-02-04T09:15:06.434"
}
}

```

Raw data HTML form

Dno customer id

country

Figure 4.15: DB&A service: how to POST to residential\_homes (bottom part of Figure 4.14)

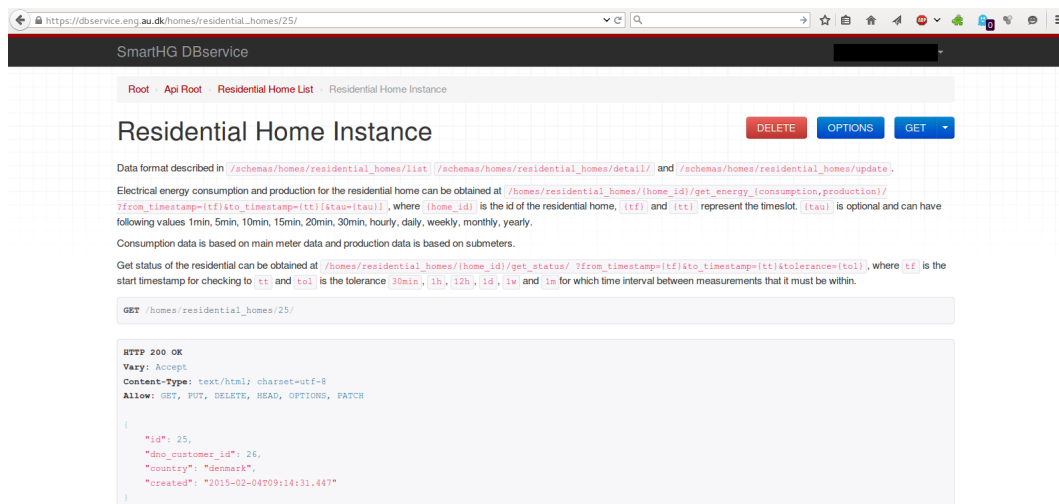


Figure 4.16: DB&A service: how to DELETE residential\_homes/25/

## Chapter 5

# DSO–IAS Protocol Implementation

In this section, we set up a test scenario for the SEP2.0 protocol, which implements the design for the open standard Internet based communication between DSO and IASs described in Deliverable D4.2.1. The test scenario is described in Table 5.1, where all the actions that Tom, a fictitious consumer, does are listed with their exact time.

Table 5.1: Use case timed scenario description of Tom’s activities

Time (hh:mm)	Activity description
18:00	Tom plugs his electric car in its station.
18:01	Tom opens his home’s front door and turns ON light.
18:05	Tom puts his laundry in the washing machine and turn it ON.
18:15	Tom prepares his food and turns ON the oven before cooking.
18:20	Tom starts the stove and cooks his food and uses the oven.
18:49	Tom has finished cooking and switches OFF the stove.
18:50	Tom switches OFF the oven.
19:00	Tom eats his food while watching television.
19:47	Tom has finished eating and switches OFF the TV.
19:50	The washing machine finishes and Tom puts his cloths into the tumble dryer.
20:00	The car is fully charged.
21:30	Tom switches ON the tumble dryer.
23:00	The tumble dryer has finished its operation.

A simple DR strategy based on a threshold constraint has been chosen, i.e., power consumption of Tom residence should not exceed 4000 W. SEP2.0 has been considered as a protocol to test. SEP2.0 is described from its specification document [12]. According to SEP2 specifications, when the client does not receive an *EndDeviceControlList* (see Figure 5.1), it waits a well-defined time before sending the message again. This timing mechanism, known as time between pooling events, is used as a tuning parameter. The maximum value of it is 5 minutes (see [12]).

Given such scenario, we develop Unified Modeling Language (UML)/MARTE [13] diagrams to describe its characteristics. As an example, the sequence diagram in Figure 5.1 models the communication between the appliances and the Demand Response (DR) server, using the SEP2.0 protocol.

Basing on the previous UML diagrams, we developed a simulation environment for the scenario. As the simulation results in a correct communication between the DR

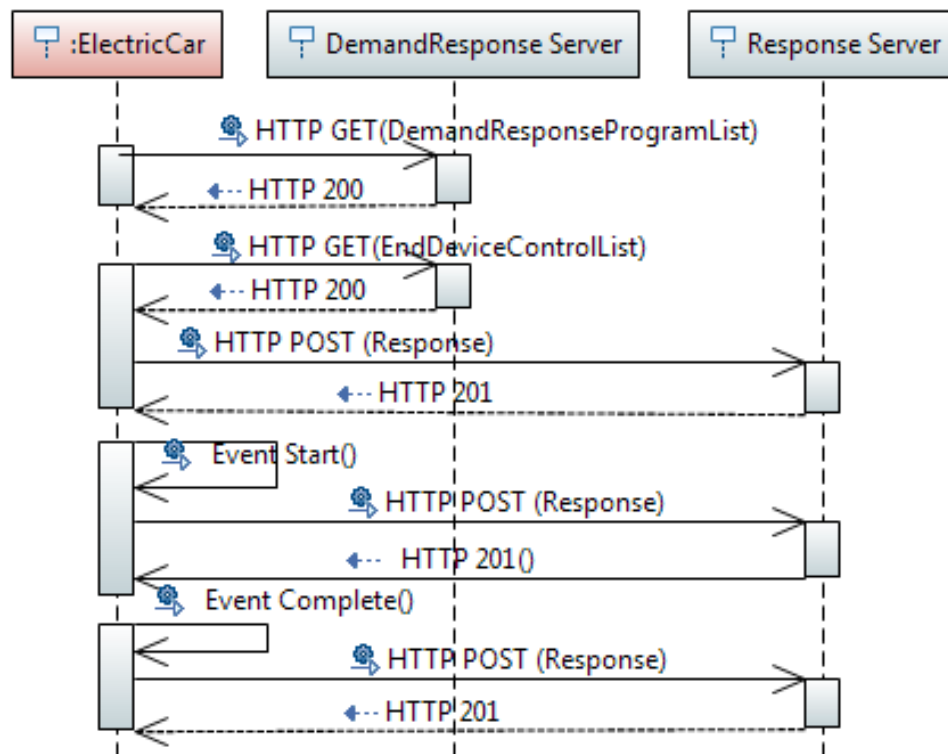


Figure 5.1: Communication between the Electric Car and DR server with SEP2.

server and the appliances, we conclude that the SEP2.0 protocol is correctly designed and implemented.

# Chapter 6

## Conclusions

In this deliverable we described the second year versions of prototypes for all SmartHG Grid Intelligent Automation Services (GIASs) Such prototypes are all based on the corresponding services design described in Deliverable D4.2.1. In the second year iteration of SmartHG, the main goal of such GIAS prototypes, together with the prototypes for Home Intelligent Automation Services (HIASs) described in Deliverable D3.2.2, is to enable the second year version evaluation of SmartHG Intelligent Automation Services (IASs) described in Deliverable D5.2.1. As a consequence, the positive evaluation of SmartHG IASs in Deliverable D5.2.1 is also a positive evaluation of the prototypes described here. Finally, a further objective of GIAS implementation is to prepare an exploitable version of each HIAS, as will be discussed in Deliverable D7.2.1.

This section is divided in three parts. First of all, in Section 6.1 we compare this year iteration of the SmartHG GIAS prototypes implementation with the first year version of the same prototypes (Deliverable D4.1.2). Then, in Section 6.2 we discuss the limitations of this year iteration of the SmartHG GIAS prototypes implementation. For each of such limitations, we outline the foreseen work to be done in the third year GIAS prototypes.

### 6.1 Advancements

In this section, we discuss the main advancements we obtained with this year SmartHG GIASs implementation w.r.t. the first year version of the same prototypes. To this aim, we discuss the enhancements for each GIAS.

**DB&A:** The most important enhancement w.r.t. the first year version is the implementation of the authentication part of the REpresentational State Transfer (RESTful) service, which was missing in the first year version. Moreover, the implementation of Application Programming Interfaces (APIs) in the second year version is completely different from the first year one, as it follows the new APIs design described in Deliverable D4.2.1.

**DAPP-H:** This year prototype of Demand Aware Price Policies for Homes (DAPP-H) went through a complete restyling w.r.t. the first year version of its ancestor Demand Aware Price Policies (DAPP). Namely, the first year version of the DAPP prototype consisted of a single executable relying on the first year version of the Database and Analytics (DB&A). Thus, in order to use it, it was necessary to i) configure and install it on a Linux machine; ii) manually load the DB&A with the required input; iii) run the DAPP application; iv) manually retrieve the output from the DB&A; and

v) optionally graph such output using some external tool (e.g. Gnuplot). Instead, in this year version of the DAPP-H prototype, all steps above may be invoked from a Web Graphical User Interface (GUI) (relying on a RESTful service dedicated to DAPP-H), thus DAPP-H is a Software as a Service (SaaS). This makes DAPP-H a service directly usable from the final user.

**PPSV:** The same described above for DAPP-H also holds for Price Policy Safety Verification (PPSV).

**EVT:** The same described above for DAPP-H and PPSV also holds for EDN Virtual Tomography (EVT).

**DAPP-K:** Since the Demand Aware Price Policies for Substation-Level Energy Storage Control (DAPP-K) service has been developed this year, the implementation of DAPP-K is completely new.

**DSO–IAS communication:** The first year prototype was limited to handling the normal Demand Response and Load Control (DRLC) user scenarios and has been made more robust with respect to foreseen end-user behavior such as a sudden cancellation of a DRLC program.

## 6.2 Limitations and Future Work

All limitations of the second year version of each GIAS, together with the planned actions to overcome them, are discussed in Table 6.1. We point out that the third year version of SmartHG GIAS prototypes, besides the planned actions discussed in Table 6.1, will also have to implement the new features (and algorithms) of the third year versions of the GIASs.



Table 6.1: Limitations for the second year evaluation & how we plan to overcome such limitations in the third year evaluation.

GIAS	Limitations of second year	Future work for third year
DB&A	Data on the Database Service (DBService) may only be accessed from the RESTful APIs, or through the default Web interface offered by the framework we used, i.e., Django	A Web interface able to offer to human users a high-level access to data (e.g., enabling upload and download of data via files) will be developed.
DAPP-H	If the underlying RESTful service contains lots of data, the service may be slow.	We plan to study more effective ways of implementing the RESTful service, or to move it to a more powerful workstation.
DAPP-H	Not all features have been implemented (e.g.: hints, kill an ongoing execution, delete figures, search figures and executions by day, search on scenarios...)	We plan to include such features in the third year.
PPSV	The same limitations and future work described above for DAPP-H also applies to PPSV.	
EVT	The same limitations and future work described above for DAPP-H and PPSV also applies to EVT.	
DAPP-K	The current version of DAPP-K assumes to interact with a program simulating an Energy Storage System (ESS) installed on an Electric Distribution Network (EDN) substation.	We will investigate the feasibility of providing a DAPP-K version able to read the aggregated demand on an actual EDN substation and send charge/discharge commands to an actual ESS installed on it.
DSO-IAS communication	The mapping of the platform-independent description of the consumer scenario, demand response strategy, and the protocol under test into executable description has been done manually.	In the third year we will implement a high-level synthesis tool to convert the Unified Modeling Language (UML) models into executable code.

# Bibliography

- [1] “Demand Aware Price Policies for Homes (DAPP-H) Web Service: `mclabservices.di.uniroma1.it/dapp_h/`,” 2014.
- [2] “DAPP-H RESTful Service: `mclabservices.di.uniroma1.it:10000`,” 2014.
- [3] “Demand Aware Price Policies for Substation-Level Energy Storage Control (DAPP-K) Web Service: `mclabservices.di.uniroma1.it/dapp_k/`,” 2014.
- [4] “DAPP-K RESTful Service: `mclabservices.di.uniroma1.it:10004`,” 2014.
- [5] “Price Policy Safety Verification (PPSV) Web Service: `mclabservices.di.uniroma1.it/ppsv/`,” 2014.
- [6] “PPSV RESTful Service: `mclabservices.di.uniroma1.it:10001`,” 2014.
- [7] “EDN Virtual Tomography (EVT) Web Service: `mclabservices.di.uniroma1.it/evt/`,” 2014.
- [8] “EVT RESTful Service: `mclabservices.di.uniroma1.it:10002`,” 2014.
- [9] “Database and Analytics (DB&A) Web Service (production): `dbservice.eng.au.dk/`,” 2014.
- [10] “All SmartHG Intelligent Automation Services (IASs) Web Page: `http://smarthg.di.uniroma1.it/second_year_release/`,” 2014.
- [11] “DB&A Web Service (testing): `radagast1.netlab.eng.au.dk/`,” 2014.
- [12] “SEP 2 Application Protocol Standard,” Tech. Rep. Document 13-0200-00, ZigBee Alliance, 2013.
- [13] Object Management Group, “A UML Profile for MARTE (version 1.1),” tech. rep., OMG document number: formal/2011-06-02, Jun. 2011.