MAS2TERING

Multi-Agent Systems and Secured coupling of Telecom and Energy gRIds for Next Generation smartgrid services

FP7 – 619682

D3.1 Multi-agent systems holonic platform generic components

Lead Author: Hassan Sleiman (CEA), Meritxell Vinyals (CEA)

With contributions from: Sandra Garcia Rodriguez and Loïs Vanhee (CEA), Michael Dibley (CU), Shaun Howell (CU), Jean-Laurent Hippolyte (CU), Yacine Rezgui (CU), Julien Ardeois (Engie)

1st Quality reviewer: Juan M. Espeche (R2M)

2nd Quality reviewer: Monjur Mourshed (CU)

Deliverable nature:	Software (O)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	29 February 2016 (M18)
Actual delivery date:	30 May 2016 (M21)
Version:	1.0

Abstract

This deliverable is intended to provide a limited public release of the software components of the multi-agent holonic platform developed in MAS2TERING. This document provides the specifications for the smart grid data model for MAS2TERING, the agents and their behaviours that will run in the multi-agent system platform, and the constraints and the objectives for these agents. These specifications were obtained from the requirements obtained in D2.1, the use cases that follow the Universal Smart Energy Framework (USEF) framework described in D6.1, and the multi-agent systems (MAS) platform described in D2.2. The GAIA methodology, which provides the methodological tools towards successfully and efficiently implementing problem solving MASs and which has been used in D2.2, has also been used in this deliverable. It is completed with the GAIA2JADE, which complements the implementation-independent GAIA methodology to support MAS development using the JADE framework. In addition, this document is accompanied by a software implementation of those components.

[End of abstract]

Executive summary

The development of the multi-agent, holonic, and secure platform in MAS2TERING is the main focus of this deliverable. This platform aims at providing an integrated platform for distributed management of the Smart Grid, based on multi-agent systems. Such platform shall allow the optimisation of generation, storage and distribution, and upgrade the grid with self-healing capabilities, which will be the focus of the deliverable D3.3. This platform will be closely integrated within the high-level grid architecture, provided in the deliverable D2.2 from WP2.

This document is based on the use cases defined in Deliverable D6.1, and the Universal Smart Energy Framework (USEF), extensively studied in the deliverable D1.6 and with which MAS2TERING project aligns. The specification is performed following the GAIA methodology, whereas the implementation is performed using the GAIA2JADE process; i.e., we devise the multi agent system, using the JADE framework, based on the GAIA models from the analysis and design phases.

Universal Smart Energy Framework (USEF) is a reference framework for market design, actor interactions and common flexibility services between the actors. Since MAS2TERING aligns with USEF, the data model, the agent types and their roles have been identified and specified based on USEF's specifications. These agents are: Device agent, Customer Energy Management System (CEMS) agent, Aggregator (AGR) agent, and Distributor System Operator (DSO) agent.

MAS2TERING defines three use cases in the deliverable D6.1, which will be used to validate the solution based on multi-agent systems. The first use case focuses on home-level optimisation, including the interoperability and the connections to handle requests/connections to the flexibility market via the aggregator. Agents involved in this use case are the Device agents, and the CEMS agent. The second use case deals with the local management, at the district level, by involving the AGR, which communicates with the CEMS agents deployed in the houses of the local community. The third use case extends the previous use case since it considers the entire low voltage power grid as the union of many local communities in a given area. This is performed by involving the DSO agent, which communicates with the aggregators to negotiate the power plans and to inform the congestion points of the power grid, if any, and consequently procure flexibility for congestion/capacity management.

GAIA methodology, which has been applied in deliverable D2.2, is also followed in this deliverable to complete the development phase. GAIA starts at the analysis phase by collecting the specifications of the multi-agent based system. It identifies the global behaviours of the system, the roles model that captures the basic skills required for each type of agent, the interaction modes that captures the needed interactions based on the previous roles, and the rules, which are the constraints on the execution activities of roles and protocols. The analysis phase of GAIA produces a preliminary roles model, a preliminary interaction model, and a set of organisational rules. Then, the design phase in GAIA aims at producing the complete specification of the MAS following these four sub-phases, namely: definition of the overall organisational structure, considering the adopted organisational structure to update the role and interaction models, the definition of the agents models by specifying agents' types and their instances, and the definition of the services model that defines blocks of activities with their conditions related with the agent roles. Finally, GAIA2JADE provides the process to develop the

specified agents, roles, behaviours, services, and protocols in JADE. Figure 1 illustrates an overview of this deliverable.

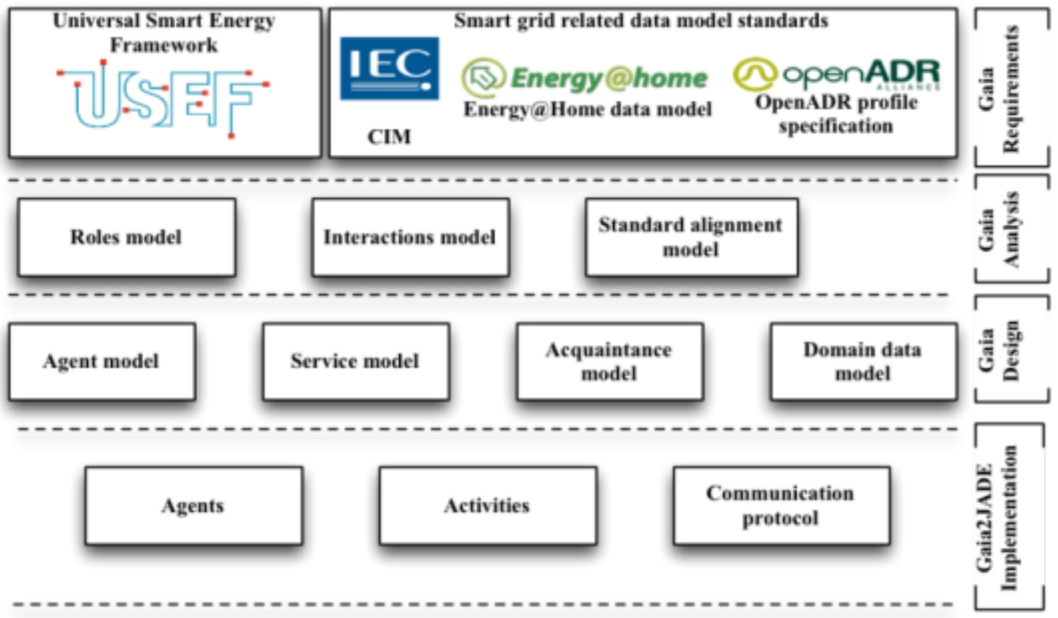


Figure 1: Summary of this deliverable contents

This deliverable aims at providing the specifications and the implementation of the data model, the constraints, the agent model, and the behaviours of the agents. We first provide the background of our specifications by briefly describing the agent types identified based on USEF, the use cases, the followed methodology, and the MAS platform where our solution has been developed. Then, the core of this document provides the smart grid data model, the agents and their behaviours, and the constraints and objectives of the agents and the smart grid devices. The deliverable is completed with deliverable D5.3 and D5.4 where the communication aspects of the agents are studied.

A research paper, based on this deliverable, has been submitted to CASE special session in IEEE Smart Cities conference[1]. The paper also includes some contents regarding communication from D5.4, and is attached as Annex in this deliverable (Annex E.1).

[1] http://events.unitn.it/en/isc2-2016/special-sessions

Document Information

IST Project Number	FP7 – 619682	Acronym	MAS2TERING
Full Title	Multi-Agent Systems and Secured coupling of Telecom and EnErgy gRIds for Next Generation smart grid services		
Project URL	http://www.MAS2TERING.eu/		
Document URL			
EU Project Officer	Patricia Arsene		

Deliverable	Number	D3.1	Title	Multi-agent systems holonic platform generic components
Work Package	Number	WP3	Title	Multi-agent systems and optimisation

Date of Delivery	Contractual	M18	Actual	M21
Status	Version 1.0		final □	
Nature	prototype □ report □ dissemination □			
Dissemination level	public X consortium □			

Authors (Partner)	CEA			
Responsible Author	Name	Hassan Sleiman	E-mail	Hassan.sleiman@cea.fr
	Partner	CEA	Phone	

Abstract (for dissemination)	This deliverable is intended to provide a limited public release of the software components of the multi-agent holonic platform developed in MAS2TERING. This document provides the specifications for the Smart Grid data model for MAS2TERING, the constraints and the objectives, and the agents and their behaviours that will run in the multi-agent system platform. These specifications were obtained from the requirements obtained in D2.1, the use cases that follow the Universal Smart Energy Framework (USEF) framework described in D6.1, and the multi-agent systems (MAS) platform described in D2.2. The GAIA methodology, which provides methodological tools towards successfully and efficiently implementing problem solving MASs and which has been used in D2.2, has also been used in this deliverable. It is completed with the GAIA2JADE, which complements the implementation-independent GAIA methodology to support MAS development using the JADE framework. In addition, this document is accompanied by a software implementation of those components.
Keywords	Multi-agent system, MAS2TERING agents' model, Smart grid model, messages ontology, data model.

Version Log

Issue Date	Rev. No.	Author	Change
03/11/2015	0.1	Meritxell Vinyals	ToC released
09/11/2015	0.2	Meritxell Vinyals	Added Section 1.1 and 1.2
01/12/2015	0.3	Meritxell Vinyals	Added the Section 3.1 agent model.
29/01/2015	0.4	Meritxell Vinyals	Completed Section 3.1
05/02/2015	0.5	Meritxell Vinyals	Completed Section 3.2
10/02/2016	0.6	Sandra Garcia	Completed Section 2.1 and 2.2
18/02/2016	0.7	Sandra Garcia	Completed Section 4
22/02/2016	0.8	Sandra Garcia	Reviewed Section 4. Format to document, table and figures list and legends added.
23/02/2016	0.9	Meritxell Vinyals	Added missing descriptions.
26/02/2016	0.10	Shaun Howell, Jean-Laurent Hippolyte	Completed section Smart Grid data model
29/02/2016	0.11	Hassan Sleiman	Prepare the advanced draft considering the reviewers comments.
15/04/2016	0.12	Loïs Vanhée	Added documentation about agent and behaviour implementation.
15/04/2016	0.13	Hassan Sleiman, Loïs Vanhée, Meritxell Vinyals	Updated specification about agent and behaviours
25/04/2016	0.14	Hassan Sleiman Jean-Laurent Hippolyte	Updated the data model. Review and update the document.
27/04/2016	0.15	Hassan Sleiman	Review and update the document.
30/04/2016	0.16	Hassan Sleiman	Update UML diagrams and Annex.
11/05/2016	0.17a	Juan Manuel Espeche	Deliverable review
12/05/2016	0.17b	Monjur Mourshed	Deliverable review
16/05/2016	0.18	Hassan Sleiman	Deliverable modification addressing reviews
20/05/2016	0.18a	Juan Manuel Espeche Monjur Mourshed	Deliverable review (2nd round)
30/05/2016	1.0	Hassan Sleiman	Deliverable final version.

Table of Contents

List of figures

List of tables

Abbreviations

ACL	Agent Communication Language
AGR	Aggregator
API	Application Programming Interface
BRP	Balance Responsible Party
CEMS	Customer Energy Management System
CIS	Component Interface Standards
CIM	Common Information Model
DER	Distributed Energy Resource
DF	Directory Facilitator
DSO	Distributor System Operator
FIPA	Foundation for Intelligent Physical Agents
FSM	Finite State Machine
HAN	Home Area Network
IRM	Interface Reference Model
JADE	Java Agent Development Framework
LGPL	Lesser General Public License
MAS	Multi-Agent System
MVD	Model View Definition
OMG	Object Management Group
OpenADR	Open Automated Demand Response
OSGi	Open Services Gateway initiative
OWL	Web Ontology Language
Q_f	The flexibility utilised
Q_{tot}	The total energy consumption
RDF	Resource Description Framework
PEV	Plug-in Electric Vehicle
SPEM	Software Process Engineering Meta-model
T_{min}	Minimum amount of time the task requires to be completed.
TSO	The Transmission System Operator
UC	Use case
UML	Unified Modelling Language
URI	Uniform Resource Identifier
USEF	Universal Smart Energy Framework
WG	Working group
WP	Work package

Definitions

FIPA: FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.

JADE framework: JADE (Java Agent Development Framework) is a Java software Framework. It is intended to simplify the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment tasks.

Multi-agent System (MAS): it is a system that models an application as a collection of components, called agents, which are characterised by their autonomy, proactivity and an ability to communicate. Agents in a MAS are considered improving the current methods for conceptualising, designing and implementing software systems, and may also be the solution to the legacy software integration problem.

GAIA: It is a methodology for the analysis and design of agent-based systems. The key concepts in GAIA are roles, which have associated with them responsibilities, permissions, activities, and protocols.

GAIA2JADE: It is a development process for the agents that uses the GAIA models and provides a roadmap for transforming GAIA formulas to Finite State Machine diagrams and then provide some code generation for JADE implementation.

Finite State Machine (FSM): It is a mathematical model that can be used to model systems in different areas, including software applications and communication protocols. It is composed of states, connected by means of transitions, which are run once the transition conditions are fulfilled to pass from one state to another. Please check Annex-A for more details.

CityGML: It is an open XML-based data model and format for the storage and exchange of virtual 3D city models. It defines the classes and relations for topographic objects in cities and regional models.

A-plan: the expected consumption profile during the day of delivery for a given Aggregator portfolio of Prosumers. This concept is aligned with USEF framework.

P-plan: the expected consumption profile during the day of delivery of a Prosumer. This concept is aligned with USEF framework.

D-prognoses: the expected consumption profile during the day of delivery for a given aggregator portfolio of Prosumers including only Prosumers related to a particular congestion point (i.e. the D-prognoses can be derived from the A-plan of an AGR excluding all prosumers not related to the particular Congestion Point). This prognosis is sent by the AGR agent to the DSO agent in order that the latter can perform grid safety analysis. This concept is aligned with USEF framework.

CEMS: the functional role defined in Common Information Model by CEN-CENELEC-ETSI Smart Grid Coordination Group. The CEMS concept is identical to the USEF BEMS (Building Energy Management System) but it operates at home level.

Device Abstraction Layer: a specification from OSGi Residential Group that specifies the set of APIs that next-generation Energy Boxes will expose to provide access to physical devices through a uniform interface.

1 Introduction

This deliverable summarises the design and the implementation of a multi-agent platform for the optimisation and the management of the grid for based on flexibility. It is a first step towards the development of such multi-agent holonic platform for MAS2TERING. Since this deliverable is of Software nature, this document provides the specification details for the implementation of the multi-agent platform and its main components by providing their design and implementation.

A holon is a self-similar or fractal structure that is stable and coherent and that consists of several holons as sub-structures. In a holonic multi-agent system, an agent that appears as a single entity to the outside world may in fact be composed of many sub-agents and conversely, many sub-agents may decide that it is advantageous to join into the coherent structure of a super-agent and thus act as single entity.

Agents are intelligent software components that can connect to hardware in order to implement physical actions. The intelligence is assigned to these agents through models describing the stakeholders' business or interest in terms of: 1) objective function to be maximised or minimised; and 2) constraints that describe their business/physical models. Constraints can either be hard constraints, which set some conditions that must be strictly fulfilled (otherwise the solution is not considered as valid), or soft constraints, which set the costs for some conditions by penalising it in the objective function.

Multi-agent Systems (MAS) have become popular solutions to tackle the complexity of decentralized systems. In a multi-agent approach, each component (physical or abstract) of a system is autonomous and can interact or communicate with their environment and with other agents via predefined interfaces. MAS have proven to bring together many disciplines in an effort to build distributed, intelligent, and robust applications especially for smart grid solutions. A number of prominent agent-oriented design methodologies have been proposed in the literature and applied by practitioners.

The GAIA methodology provides methodological tools towards successfully and efficiently implementing problem-solving MASs [1, 2]. The first phase of GAIA is the analysis, which extracts from the system requirements: (a) the roles of the organization (including an informal description, permissions, activities and protocols to be performed by the role) and (b) the interactions that should be conducted (including the purpose, the initiator, the responder, inputs, outputs and processing to be performed by the interaction). The output of the analysis phase is then used towards producing more concrete artefacts in the design phase, which further describes the agents (types of agents in the system), services (activities to be performed by a role) and acquaintances (describing who is connected to whom). GAIA methodology was explained and followed in deliverable D2.2, where the agents' roles and models have been presented.

GAIA2JADE complements the implementation-independent GAIA methodology to support MAS development using the JADE framework [3, 4]. It adds an additional phase that follows GAIA's design phase, called JADE implementation. GAIA2JADE allows developing real-world MAS that had been analysed and designed using GAIA, and that shall be implemented within JADE framework. The JADE implementation phase provides MAS developers with systematic steps and guidelines to produce the agents' Java code and a repository communication protocols, the implementation of the activities, and agent behaviours. GAIA2JADE process was described using the Software Process Engineering Meta-model (SPEM) proposed by the Object Management Group (OMG).

The JADE implementation process involves the developer and produces two software products: a repository of behaviours (i.e. reusable pieces of code that can be used for devising agents or other behaviours that extend existing ones), and the JAVA code with the agents built using these behaviours.

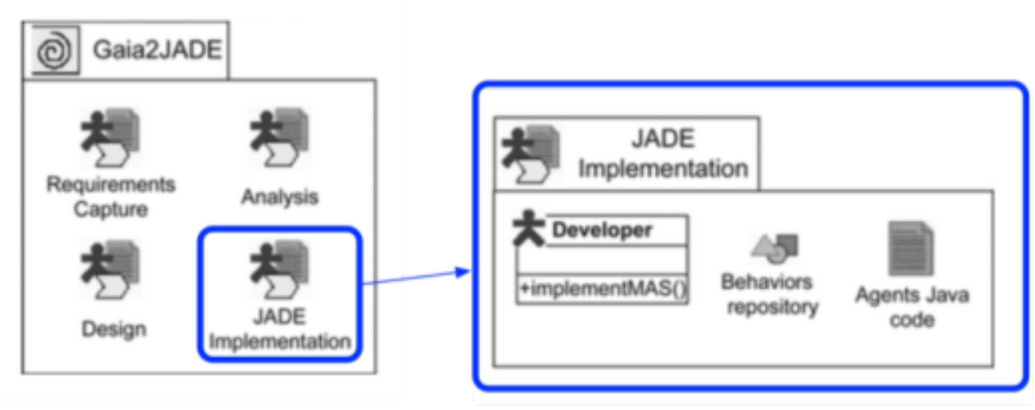


Figure 2: The GAIA2JADE process and the JADE implementation process package

Following the analysis phase carried out in WP2, this deliverable uses the GAIA methodology for the design of the MAS2TERING multi-agent framework and uses the JADE Framework for the implementation. The mapping is performed using the GAIA2JADE methodology. The main outputs of this deliverable are the high-level software components and the recursive and hierarchical structures that implement the holonic approach of the multi agent platform. Mainly, these components are the data model for defining the grid model and the ontology used for message exchange between the agents for the smart grid, the agents and their behaviours, and the constraints.

As shown in Figure 2, from the four processes that compose the GAIA2JADE process, this deliverable focuses on the JADE implementation process. It strongly builds on the deliverables D2.1 and D2.2 by taking as input the requirements and the five GAIA models defined in deliverable D2.2. The output of this deliverable is the implementation of generic high level software components (mainly the data model, the behaviours and the agents), generic interaction protocols and the recursive and hierarchical structures that implement the holonic approach. The software elements provided in this deliverable, which are completed with the communication protocols defined in D5.3 and D5.4, will define the basis for the management and optimisation algorithms developed in deliverable D3.3. This code will be used to extend our platform components described in D2.2

To utilise the application dependent data in the agents, the domain ontology has been developed, based on the USEF framework [5], and existing knowledge models of the domain. The data model, or ontology, formalises the concepts and relationships in the domain; both those which constitute message payloads between agents, and those which describe the agents themselves. This modelling is based on existing standards, extended to the USEF framework and the MAS2TERING use cases. The deliverable also specifies the activities refinement table, which defines the application-dependent data (in UML), their structure and the algorithms that are going to be used by agents in the MAS2TERING solution

Based on the use cases defined in deliverable D6.1, the main implemented agents that are described in this deliverable are as follows: 1) the Distributor System Operator agent (DSO); 2) the Aggregator (AGR) agent; 3) the Customer Energy Management System agent (CEMS); and 4) Device agents (with seven subtypes) that abstract the flexibility provided by the different physical devices. These software elements will define the base where management and optimisation algorithms (see tasks 3.2. and 3.3 in WP3) will be integrated. The particular algorithms and behaviours related to such management (optimisation and prediction) algorithms will be implemented as part of future deliverables D3.2 (i.e. prediction and forecasting algorithms) and D3.3 (i.e. optimisation algorithms).

This document is organised as follows: Chapter 2 details the background from which the specifications for this deliverable has been obtained (USEF and MAS2TERING use cases), and the followed methodology (GAIA and GAIA2JADE), in addition to the MAS2TERING platform, where our solution will be integrated; Chapter 3 describes the Smart Grid model that include the CIM data model, integrated with other standards, and the ACL messages ontology; Chapter 4, provides the specifications for the Agents model and their behaviours; Chapter 5, describes the functions for the physical agents of the different MAS2TERING use cases and provides their objectives and constraints; Chapter 6 maps each agent type and behaviour to the use case in which it will be instantiated; and finally, Chapter 7 concludes the document and briefly describes the upcoming deliverables related to it. This deliverable is also accompanied by the software implementation of the defined extensions of the components.

2 Background

This chapter briefly describes the context from which our specifications have been drawn. We first summarise the agents types identified using the USEF framework (extensively described in deliverable D1.6) and then, we briefly describe the use cases that will be developed in WP6, and where the agents will be validated.

2.1 Agent types and roles based on USEF framework

As described in deliverable D1.6, MAS2TERING aligns with the USEF framework. USEF delivers a common standard for flexibility market solution by defining the different stakeholders and their roles and responsibilities. Furthermore, it ensures that the value of flexibility can be maximised and transferred. USEF proposes the aggregator as the centre of the flexibility value chain, whose services are provided to the Prosumer, the Balance Responsible Party (BRP), The Distribution System Operator (DSO), and the Transmission System Operator (TSO). In MAS2TERING, we focus on the Prosumer, Aggregator, and the DSO, for which we only consider the role of congestion/capacity management.

Following USEF's specifications and the GAIA methodology, we have identified four agent types and roles, namely: Device agent, CEMS agent, AGR agent, and DSO agent. Below we show a short description of such agents with their associated roles, whereas further details are given in Chapter 4.

Agent type	Roles
CEMS	The Consumer Energy Management System (CEMS), monitors, controls and optimises the flexibility of the prosumer.
AGR	The aggregator manages the flexibility produced by a portfolio of prosumers, and tries to provide flexibility to other participants in the flexibility market. It is an intermediate agent between the Prosumer and the DSO.
DSO	The Distribution System Operator (DSO) agent implements the functionality of grid congestion/capacity management (Other DSO roles fall outside the scope of MAS2TERING and will not be considered).
Device	This class of agents represents the controllable and non-controllable energy-consuming and/or producing systems in the grid. Some of them can be actively controlled to provide flexibility.

Table 1 Agent types and their roles

Furthermore, based on the USEF framework and the value chain it proposes, the interactions between agents have been also identified in D2.2. The following table shows for each agent the other agents it communicates with. For instance, an AGR agent may communicate with another AGR agent, a CEMS or a DSO agent. This model can be used for identifying potential communication bottlenecks that may arise at runtime.

Agent	CEMS	AGR	DSO	Device
CEMS		X		X
AGR	X	X	X	
DSO		X		
Device	X			

Table 2 Communications between agents

2.2 MAS2TERING use cases

MAS2TERING relies on three use cases that will be implemented in WP6, to test the solution developed in the project, and assess the achievement of its objectives. In the following, we briefly describe each of the use cases.

The first use case (UC1), focuses on the Home Area Network (HAN) and the interaction with the end-user. The main goal is to prove the interoperability between the HAN management system, the smart meter and a technical interface (gateway), which allows the bi-directional communication between the end user and the rest of the actors of the low voltage grid. MAS implementation at this level focuses on two main agents, namely: Device agent and CEMS agent. The first one is in charge of the controllable and non-controllable devices. The second one performs the in-home optimisation, communicates with the AGR agent and also with the real devices to get their flexibilities and send them the control signals.

The second use case (UC2) deals instead with the local management at the district level. The objective is to demonstrate the effectiveness of balancing and optimising (without considering grid constraints), at local community level, as an alternative to traditional centralised optimisation. This use case receives as input the data coming from the UC1. The local community is considered as a collection of consumption/generation nodes that are managed by a single entity, the aggregator. The AGR agent enables the local flexibility market by negotiating with Prosumers (via their corresponding CEMS agents) in their portfolio regarding the use of their flexibilities to collectively optimise energy flows without considering grid constraints (i.e. the grid capacity/congestion management capacity of the DSO does not form part of this use case).

The third use case (UC3) is an extension of UC2; it handles the entire low voltage grid as the union of many local communities in a given area. This use case involves the DSOs, and intends to demonstrate that the local optimisation enabled in UC2 may be a cost-effective way to deal with local congestions and globally increase the grid performance, its reliability and resilience. The DSO agent is added to MAS system for this use case. In case of expected congestion, the DSO agent initiates negotiations with local AGR in the area to procure flexibility.

2.3 Methodology

One of the main results from WP2, and in particular in deliverable D2.2, is the decision of using GAIA as a methodology for designing MAS2TERING agents and the use of Java Agent Development

Framework (JADE), as a framework for implementing them. In this deliverable, we apply the GAIA2JADE process to convert from the identified models into the real MAS implementation.

GAIA is a methodology for the analysis and the design of the multi-agent systems [2, 1], whereas JADE[2] is an open-source software platform developed by Telecom Italia Lab (TILAB) in Italy, and distributed under the terms of the Lesser General Public License (LGPL). JADE is a middle-ware (written entirely in the JAVA language), which simplifies the implementation of multi agents by providing a set of graphical tools that support the debugging and deployment of agents. More information on JADE can be found in [3].

Another advantage of combining the GAIA methodology and the JADE platform is the large amount of research and work in the literature that focuses on mapping from one to another; i.e., there are few works that studied how to translate the GAIA model to the JADE platform, in the so-called GAIA2JADE process [1, 4]. This is achieved by specifically focusing on the JADE platform in the design phase, whereas the designer can move straight forward towards the implementation, without having to adapt the results of the design phase.

We follow the GAIA2JADE process for implementing the GAIA models, which are defined in D2.2 using the JADE framework that is chosen in MAS2TERING as underlying MAS platform. The GAIA2JADE process aggregates four process packages. In this deliverable, we focus on the JADE implementation process. This process involves the developer role and produces two products (i.e. outputs of this deliverable), namely: the Java code and the repository of behaviours. Notice that JADE behaviours are reusable pieces of code (components) that can be used for building agents or other complex behaviours.

In the GAIA2JADE process, all the GAIA responsibilities (i.e. activities and protocols) that are defined in the role model are transformed into MAS2TERING behaviours[3]. The transformation process is as follows:

1. Define the behaviours corresponding to the activities field in the role model
2. Define the protocol behaviours corresponding to the protocols field in the roles model
3. Link the defined agents with their behaviours. For this purpose, the developer shall use the setup method in the Agent class by invoking all the methods (GAIA activities) that are executed only one time at the beginning of the top responsibility. It also initialises all agent data structures and adds all behaviours of the lower level in the agent scheduler.

[2] http://jade.tilab.com/

[3] Activities and protocols can be translated to JADE behaviours, to action methods (which will be part of finite state machine – FSM like behaviours) or to simple methods of behaviours.

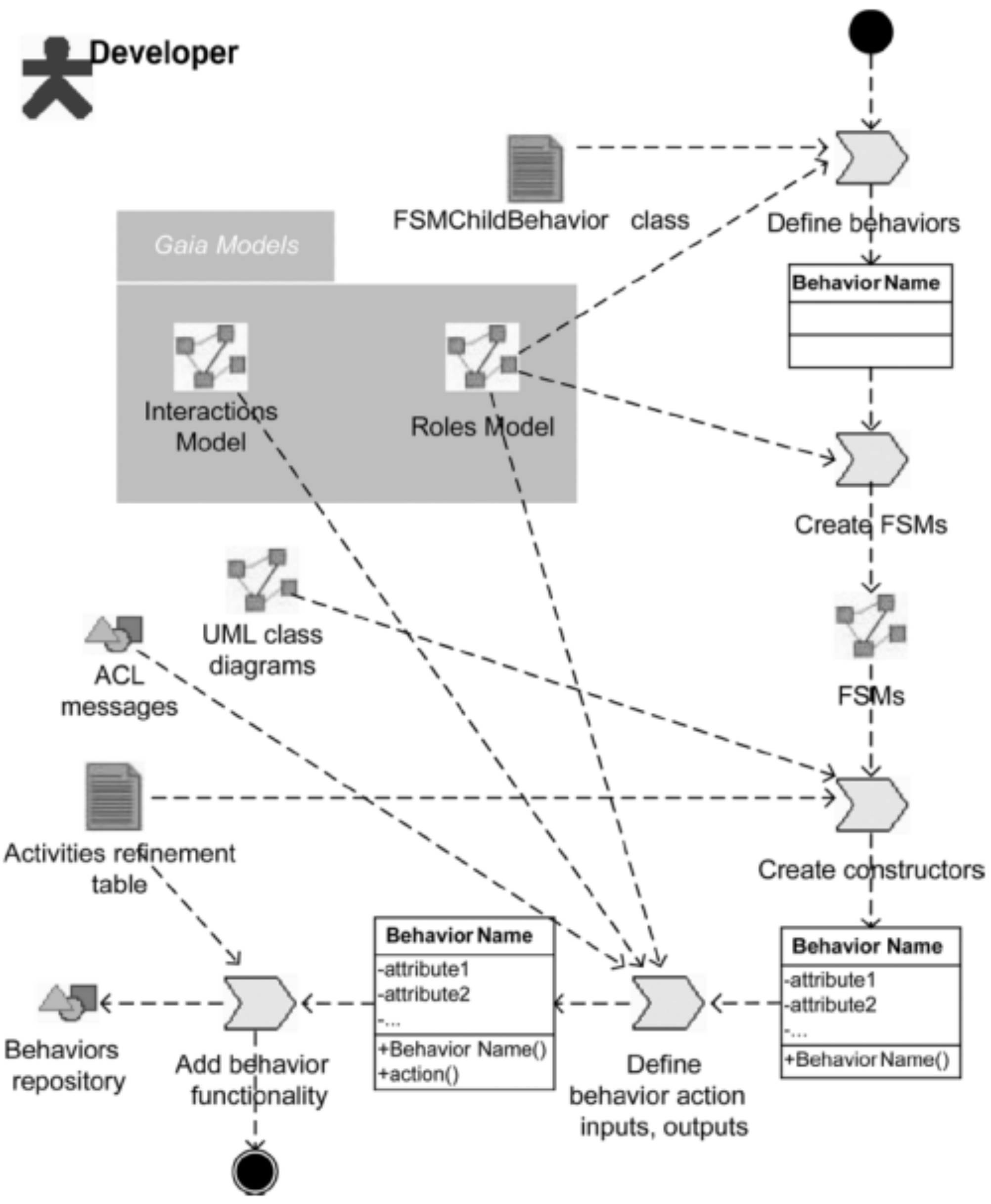


Figure 3 GAIA2JADE definition of JADE behaviours and agents implementation process [1].

This deliverable defines the activities refinement table, including application-dependent data, their structure and the algorithms that are going to be used by the agents are defined. Then, we define the JADE behaviours and the agent classes following the corresponding GAIA2JADE process illustrated in Figure 3.

2.4 MAS platform

The MAS platform, which has been described in the deliverable D2.2, is extended to include the MAS2TERING solution. In this chapter we give a brief description of the main components of the platform and provide more details on the interfaces, classes and relationships that are specifically developed to cope with the MAS2TERING project within each component.

The platform is built upon a component-based architecture to enable clear separation between the different components; i.e., each one represents a reusable component that is composed of a collection of conceptually related classes, which implement specific functionality and provide services to the other ones. These components are illustrated in Figure 4 and are as follows: the user interface, the Smart Grid model component, the agents' model component, the constraints and objectives component, communication protocols component, the security component, and a utility component.

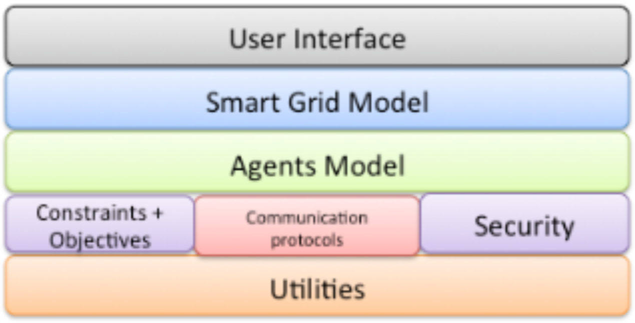


Figure 4: MAS platform architecture

The following subsections briefly describe each of the components of the MAS platform architecture, and focus on the components that have been extended to build the MAS2TERING multi-agent based solution. These components are: the Smart Grid model component, the Agents model, and the Constraints and Objectives component. The extension of the communication protocols is described in D5.3 and implemented in D5.4, whereas the extension of the Security component is described in D4.2.

2.4.1 User interface

This component provides a graphical user interface (GUI) to show a 3D graphical and interactive interface of the grid. It displays CityGML information in 3D, a 2D model of the grid, and real time information from the agents, by means of a multilayer representation. A factory class is used to create the entities that represent the components that are part of the grid, place them on the CityGML map, and to connect them with their agents in the MAS model. Furthermore, it creates an information panel for each of them in order to display the information and events on real time using user-friendly reporting tools. Such tools allow colour-stated visualisation on a map of the entire network, including some metric parameters for statistical and historical reporting.

2.4.2 Smart grid model

As described in D2.2, this component contains all the classes for defining the physical components and characteristics of a power grid (The Smart Grid model) in the scope of the project and the ontology used for message exchange. Based on this component, the user is expected to define the grid model by using the user-defined profile.

2.4.3 Agent model

This component implements the kernel of the platform, the distributed run-time environment that supports the entire platform and its tools. Many of the functionalities of this layer will be relying on the JADE framework. The view of agents in the MAS2TERING solution is based on following definition:

An agent is an intelligent software component that exhibits the following properties:

- Autonomy: each agent is independent of other agents and has (some kind of) control over its actions and internal state to achieve its individual goals without any direct intervention of humans or others.
- Social ability: each agent interacts with other agents (including also humans and other third-party software) via some kind of agent communication language in order to negotiate/cooperate/compete to achieve its goals (i.e. goal-directed behaviours).
- Reactivity: agents perceive their environment and respond in a timely fashion to changes occurring therein (i.e. an agent may be possible connected to hardware in order to implement physical actions).

This particular view of agents is the only assumption for analysis, while the design is specific to the JADE platform, which is a FIPA-compliant realisation of the above vision. To be able to realise that vision, MAS2TERING uses a model for each agent that contains: its goals (via constraints and objectives), its actions (via behaviours) and its interactions (via agent protocols).

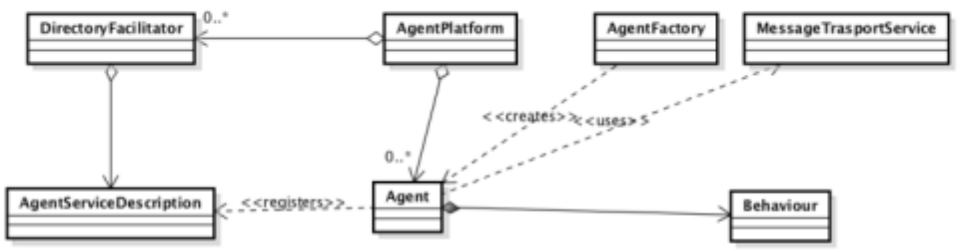


Figure 5 View of core classes of the MAS level

As described in deliverable D2.2, in the core of the MAS component, there is the agent class defined by the JADE framework. An agent is viewed here as a thread associated to a mailbox that you want to enhance with behaviours. A behaviour can be viewed as a method which defines when it should be executed, take the state of the agent as argument, execute some actions and return whether it should be kept running or being stopped (c.f. Behaviours section). When assigned to the agent, the behaviours will have access to certain functionalities of the agent that owns them. Those reflective functionalities include identifying the agent, sending messages and parsing the mailbox.

By using the static methods of the 'AgentFactory', it is possible to build a JADE Agent from an AgentSpecification that contains the set of behaviours that the agent will include. The factory allows starting JADE and launching a MAS system from the Collection of AgentSpecifications objects (c.f. Figure 5).

2.4.4 Constraints and objectives

This component includes the necessary classes for modelling the constraints used to define the restrictions of a grid component, of agents, and of their objectives. Each agent needs a way to represent its goals, which are based on its individual interests (e.g. balance for distribution operators; reduction of generation price for producers; reduction of consumption and comfort maximisation for consumers, etc.).

2.4.5 Communication and Protocols

This package contains all the classes that provide support for implementing standard interaction protocols in MAS2TERING. With exception of some functionalities for which an agent will not require communication with other agents (i.e., the so-called agent activities in the GAIA role model), the rest of agent´s behaviours will be implemented as a part of a multi-agent interaction protocol. Examples of protocols are auctions, subscriptions to receive notifications, negotiations and a large etc.

JADE defines some basic protocols. However, MAS2TERING will need to enhance existing protocols and create new ones to be able to fulfil the projects requirements. In this section we describe the main building blocks necessary to define such protocols, whereas the particular definition of the protocols for MAS2TERING shall be defined in the deliverable D5.3 and implemented in the deliverable D5.4.

When participating in a conversation driven by an interaction protocol, an agent can play either the initiator or the responder role. Consequently, classes in this package are divided into initiators and responders. For instance, following a protocol of subscription we have the SubscriptionInitiator and the SubscriptionResponder and so on. Playing a role in a conversation, no matter if it is the initator or responder role, implies executing a task of some sort and thus all protocol classes (both initiators and responders) are behaviours. Both protocol classes are implemented as subclasses of FMS-Behaviour and each callback method is invoked in a dedicated state of finite state machine. This implies that an agent that is going to execute some protocol (as for example the DistrictManagement agent) will contain the FlexibilityNegotiationInitiator as part of its behaviours.

Complex multi-agent interaction protocols can be built with nested protocols. For each protocol we may need to define a set of specific messages that relate to this protocol, the protocol may be associated to a particular ontology and a particular language. This package will be extended in deliverable D5.4, based on the communication specification in deliverable D5.3.

2.4.6 Security component

This component includes the classes necessary for authentication of the agents in the MAS platform, and the message encryption, necessary for the communication between the agents. The contents of this component are briefly described in D2.2, and are defined in more details in deliverable D4.2.

2.4.7 Utilities component

The utility and services classes that can be used by the other components are included in these components. It is composed of four sub-components, namely: the connecters that include connectors for different toolkits and simulations tools; the loggers that includes the classes necessary for event registration, by means of loggers; the information fusion subcomponent that deals with different data models and allow merging and wrapping data; and the utils, that includes utilities such as file readers and some statistical utilities.

3 Smart Grid Model

As described in D2.2, this component contains all the classes for defining the physical components and characteristics of a power grid (The Smart Grid model) in the scope of the project and the ontology used for message exchange. Based on this component, the user is expected to define the grid model by using the user-defined profile. For this deliverable, we have extended this component by defining the MAS2TERING data model that defines the common expression of information exchange between MAS2TERING agents in the different use cases and the Demand Side Management. This data model follows the elicitation of the domain knowledge through each use case, and it is aligned with relevant standards (in particular with FIPA-ACL, Energy@Home, CIM, IEC 61968 and OpenADR).

A common data model is required in MAS2TERING for the common expression of information exchange between participating entities. Instead of multiple ad-hoc mapping and conversion processes between arbitrary models, participants will either use the common model internally or map their internal model to a common schema. The common ontology will be used for formulating messaging data structures for syntactical and implied semantic compatibility between entities for the support of upper business processes. The common schema will use constructs from current standards. Three primary standards have been identified for their usage, with varying relevance across the use cases, namely: the IEC 61970 standard for modelling the electrical domain, the OpenADR standard for modelling demand response within the Smart Grid, and the Energy@Home standard for domestic conceptual modelling.

Due to the predominance of multi agents in the MAS2TERING platform, the primary usage of the data model is to formulate the contents for those messages, independent of the protocols. As such, the conceptual modelling will result in an ontology suitable for use within JADE. This will extend JADE's Base Ontology Java class to formalise a vocabulary, as well as descriptions of the concepts, predicates, agent actions, and data slots relevant to the domain. In addition, schemas will form the ontology metadata attributes of those messages, and the schemas will be generated from defined custom contexts applied to the base models. Both extending JADE's Base Ontology and defining and registering the schemas can be achieved by using JADE BeanOntology, assuming that concept, predicate and agent action beans have been produced first.

High level descriptions of the main constituent standards used are outlined below together with a description of their scopes, followed by a simple methodology used to derive the implementation artefacts, the results of the use case based elicitation, the resultant ontology, and its alignment with the relevant standards.

3.1 The CIM

The Common Information Model (CIM) consists of three-layered parts published as a reference model by the working group TC57, which is in turn based on the United Nations electronic exchange standards. At the Smart Grid component, there is the UML class model capturing entities and their attributes and relationships between entities, including specialisation and association, together with further constraints such as cardinality in associations. The domain described by the model is wide such that is able to support a diverse range of systems and processes including network management, outage management, work management, compliance checking, asset management, business process, customer information, risk analysis, planning activities among others. The standard therefore models entities such as electrical infrastructures, topology, power grid assets / equipment, geo-spatial / GIS, facility management, cross-cutting those areas with support for engineering, operations, maintenance, quality and operations for a range of stakeholders.

The upper layer information model of CIM is composed of four parts, developed by working groups WG13 (IEC 61970) and WG16 extension, WG14 (IEC 61968) and WG15 (c.f. Figure 6). WG13 contains the model common core and other central packages including those called wires and topology describing the connection of the grid infrastructure, its assets, and measurement and generation concepts. It is extended in WG16 by describing marketing aspects and including support for bidding and security constraints. Figure 4 shows WG14, which covers the functional and operational activities. WG15 contains business oriented models covering market operations, energy scheduling, financial interests.

Regarding interfaces, IEC61970 part 4xx is a series of Component Interface Standards (CIS). The CIS is a functional specification that applications should implement in order to comply with standard messages exchange. A further related standard is part 5, which specifics message realisation while additionally IEC 61968-1 describes an interface reference model (IRM). The scope of those message exchange formulations is likely to cover FIPA-ACL message content in MAS2TERING but the extent of stack layering that will be utilised form the standards is not yet clear.

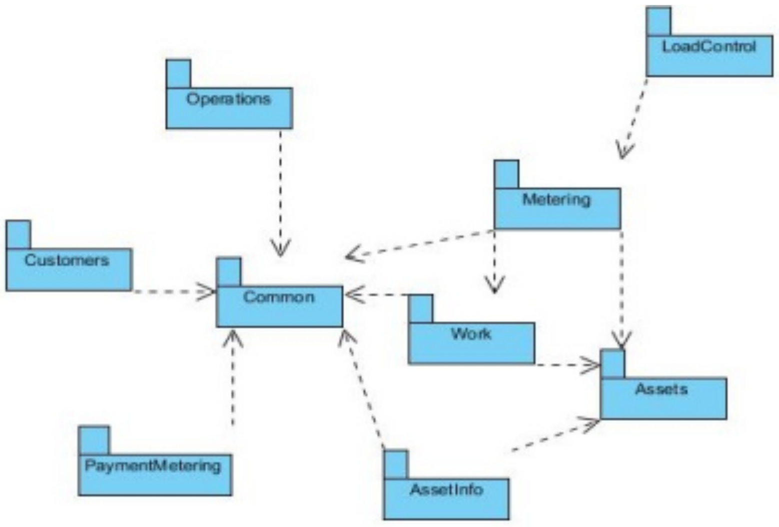


Figure 6: Main WG14 / IEC 61968 CIM packages

The middle layer of the CIM defines the context and subsets of the base model that can be used for message exchange. The profile instances define which parts are mandatory and which are optional, and can specify constraints on the base information but cannot add to the information model (for that purpose the base model has to be extended).

Finally, the lower part of the CIM is an implementation specification that defines the implementation resources. For this work package, the implementation artefacts will contain the specifications for the syntax and the serialisation for the contents ('payload') of the messages used in communications between entities. Possible schemas to be used are the XML Schema (XMLS) and the RDF Schema (RDFS). XMLS offers a simpler implementation, while RDFS gives the potential for a richer capture of explicit semantics. RDFS introduces the constructs: resources, properties and relations as well as namespaces and URIs etc. As well as the use of RDFS for custom implementation, models such as the Web Ontology Language (OWL) could be used for which several reasoners and editing tools are readily available. In addition to message contents schemas, overlapping with the other project work packages, other schemas can be generated, such as the schemas for databases or internal agent models. The transformation of the class models to payload or other contents is the main purpose of this layer resulting in a 'trimmed' and modified class structure. Such transformation can involve the modification of associations to aggregations and the removal of superclass definitions where that information is known to participants.

The derivation of the profile is use-case driven, iterative and incremental as implied in the figure. Typically, the process is driven by UML sequence diagrams generated from use cases, but essentially anything that conveys the nature the interaction to be supported is adequate. That specification is the elaborated and transformed as outlined above to generate a version controlled schema instance. As is typical of iterative software development processes the profile generation methodology includes a 'feedback' loop and this is expected to be a central concern in MAS2TERING as exchange requirements emerge during the project lifecycle.

3.2 The OpenADR

The OpenADR (Open Automated Demand Response) standard targets the interoperation and automation of applications concerned with integrating consumer, supplier and aggregator demand and response (of electrical energy supply) information in order to better manage the resources from several perspectives including economy of cost and resources, business models, availability and other concerns. Its coverage is as well as specifying a data model, describes the communication mechanisms between the participating entities which are servers that publish information (referred to as Virtual Top Node) and consumers that subscribe to information supplies (Virtual End Nodes). However, the data model is independent of the transport specifications. The standard consists of a profile specification and a schema. The data model described by OpenADR is the primary resources of interest to MAS2TERING, primarily focusing on the schemas (although the security framework utilising Public Key Infrastructure certificates may be of interest to other work packages). The model addresses energy reduction and shifting strategies which is a central concern in the project. In particular, the formalisations for the following, a subset of OASIS EI Version 1.0, are likely to be of interest:

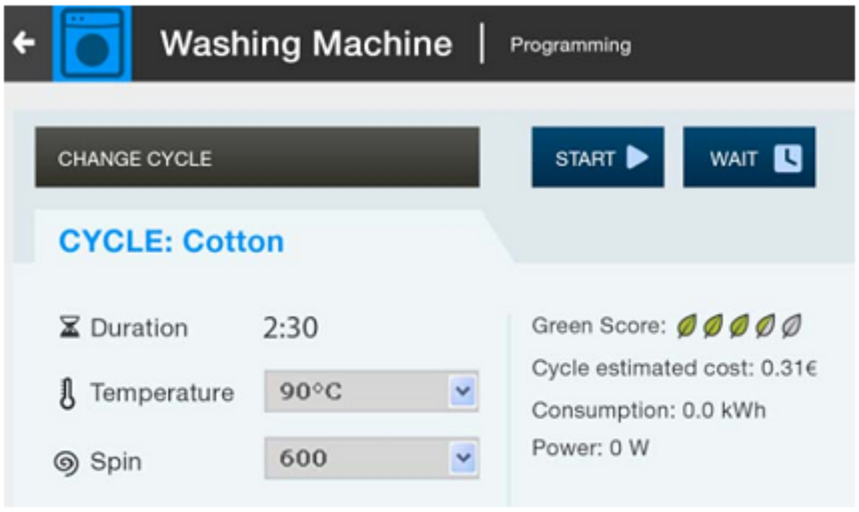
1. market context
2. event descriptions
3. dynamic pricing
4. availability and related constraints
5. pricing strategies
6. opting in and out of schedules

The event model from the OASIS EI v1.0 standard corresponding to the OpenADR event schema is illustrated in *OpenADR 2.0a Profile Specification* document.

Some automated support for MAS2TERING data model merging with the other UML data models may be feasible using XSLT transformation scripts and an appropriate transformation engine.

3.3 Energy@Home data model

The Energy@Home data model specifies a representation model for home area networks, based on the CIM approach (through its evolution into the SEP2 model), and it is broadly aligned with the OpenADR schema. The specification includes the concepts regarding smart appliances, renewable energy generation, smart meters and smart user interfaces. For this reason, the Energy@Home specification is similar in scope to use case 1 of the MAS2TERING project. The data model describes the device properties and the properties of the device functions that are used to provide the user interfaces similar to the one shown in Figure 7.



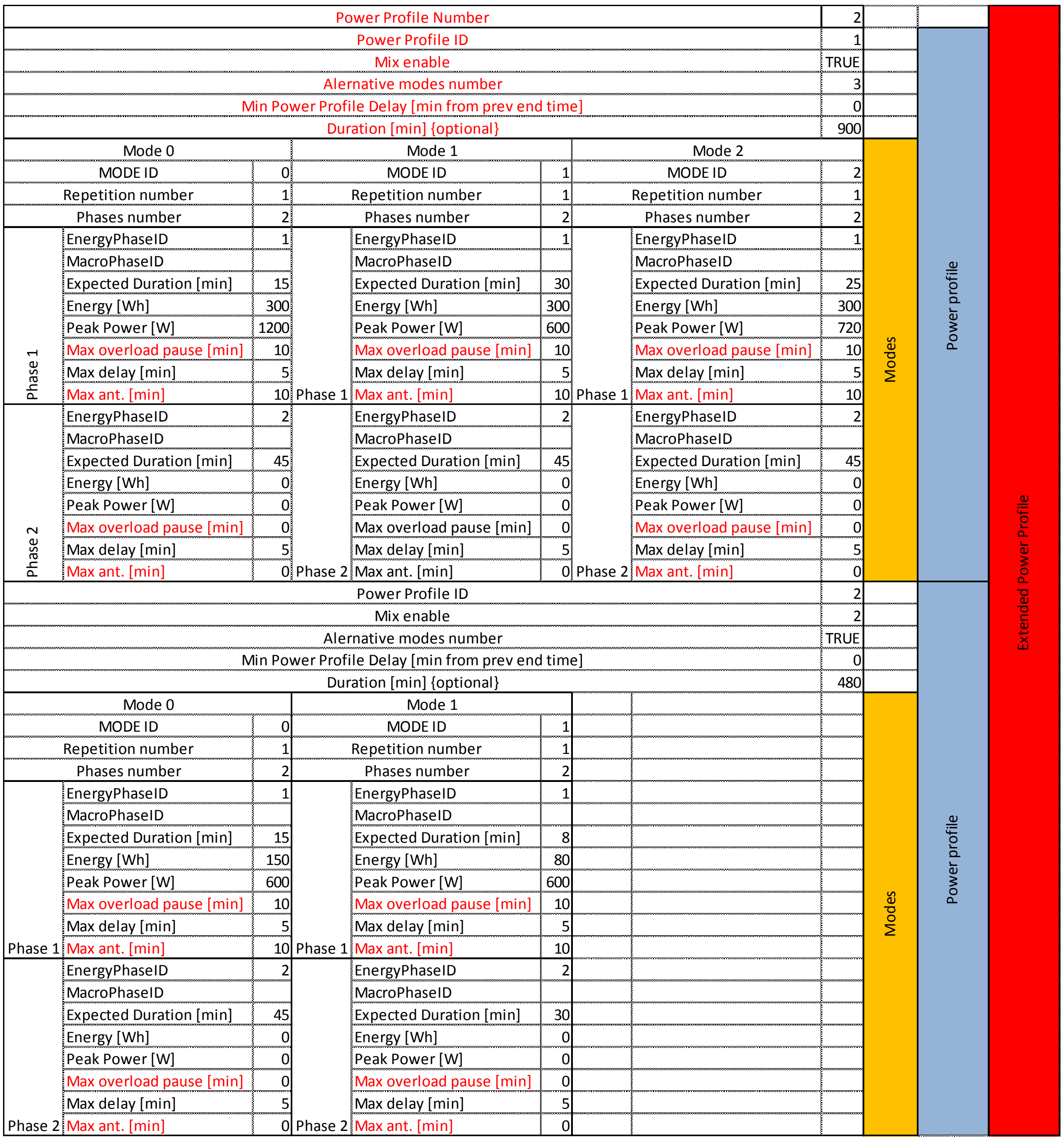
CYCLE: Cotton

Duration 2:30
Temperature 90°C
Spin 600

Green Score:
Cycle estimated cost: 0.31€
Consumption: 0.0 kWh
Power: 0 W

Figure 7: Smart appliance user interface based on the Energy@Home model

As well as the 'static parameters' related to device properties, Energy@Home also formalises a method of describing devices' energy consumption profiles in terms of energy phases, modes, power profiles and extended profiles. This is highly relevant to the MAS2TERING project, as permutations of modes and profiles would facilitate the concept of flexibility by curtailing or deferring load. An energy phase is the atomic component of an energy schedule; whereby a phase represents a sub process performed by an appliance, such as a pre-wash cycle of a washing machine. A mode is then a collection of phases, and represents one method of completing a function of an appliance, such as a wash cycle with a set temperature, for a washing machine. A power profile is then a task that the appliance performs, which can be accomplished through various modes. An extended power profile is then a collection of power profiles, such as a wash-dry program for a washing machine. This is presented visually in Figure 8.



	Power Profile Number							2	
	Power Profile ID							1	Power profile
	Mix enable							TRUE	
	Alernative modes number							3	
	Min Power Profile Delay [min from prev end time]							0	
	Duration [min] {optional}							900	
	Mode 0			Mode 1			Mode 2		Modes
	MODE ID	0		MODE ID	1		MODE ID	2	
	Repetition number	1		Repetition number	1		Repetition number	1	
	Phases number	2		Phases number	2		Phases number	2	
	EnergyPhaseID	1		EnergyPhaseID	1		EnergyPhaseID	1	
	MacroPhaseID			MacroPhaseID			MacroPhaseID		
	Expected Duration [min]	15		Expected Duration [min]	30		Expected Duration [min]	25	
	Energy [Wh]	300		Energy [Wh]	300		Energy [Wh]	300	
	Peak Power [W]	1200		Peak Power [W]	600		Peak Power [W]	720	
	Max overload pause [min]	10		Max overload pause [min]	10		Max overload pause [min]	10	
	Max delay [min]	5		Max delay [min]	5		Max delay [min]	5	
Phase 1	Max ant. [min]	10	Phase 1	Max ant. [min]	10	Phase 1	Max ant. [min]	10	
	EnergyPhaseID	2		EnergyPhaseID	2		EnergyPhaseID	2	
	MacroPhaseID			MacroPhaseID			MacroPhaseID		
	Expected Duration [min]	45		Expected Duration [min]	45		Expected Duration [min]	45	
	Energy [Wh]	0		Energy [Wh]	0		Energy [Wh]	0	
	Peak Power [W]	0		Peak Power [W]	0		Peak Power [W]	0	
	Max overload pause [min]	0		Max overload pause [min]	0		Max overload pause [min]	0	
	Max delay [min]	5		Max delay [min]	5		Max delay [min]	5	
Phase 2	Max ant. [min]	0	Phase 2	Max ant. [min]	0	Phase 2	Max ant. [min]	0	Extended Power Profile
	Power Profile ID							2	Power profile
	Mix enable							2	
	Alernative modes number							TRUE	
	Min Power Profile Delay [min from prev end time]							0	
	Duration [min] {optional}							480	
	Mode 0			Mode 1					Modes
	MODE ID	0		MODE ID	1				
	Repetition number	1		Repetition number	1				
	Phases number	2		Phases number	2				
	EnergyPhaseID	1		EnergyPhaseID	1				
	MacroPhaseID			MacroPhaseID					
	Expected Duration [min]	15		Expected Duration [min]	8				
	Energy [Wh]	150		Energy [Wh]	80				
	Peak Power [W]	600		Peak Power [W]	600				
	Max overload pause [min]	10		Max overload pause [min]	10				
	Max delay [min]	5		Max delay [min]	5				
Phase 1	Max ant. [min]	10	Phase 1	Max ant. [min]	10				
	EnergyPhaseID	2		EnergyPhaseID	2				
	MacroPhaseID			MacroPhaseID					
	Expected Duration [min]	45		Expected Duration [min]	30				
	Energy [Wh]	0		Energy [Wh]	0				
	Peak Power [W]	0		Peak Power [W]	0				
	Max overload pause [min]	0		Max overload pause [min]	0				
	Max delay [min]	5		Max delay [min]	5				
Phase 2	Max ant. [min]	0	Phase 2	Max ant. [min]	0				

Figure 8: Breakdown of energy profile objects and properties in the Energy@Home data model

As outlined, the CIM (and IEC 61968 CIM extension) will be the central resource extended in the framework by other models. The layered structure is shown in Figure 9. The roles of the different layers have been described above and the manifestation of the model and syntax layers is respectively UML class model and XML Schema (XSD format). The context layer manifestation is again a UML model but is the specification of a subset and in canonical form. Additional constraints may be specified using UML constructs.

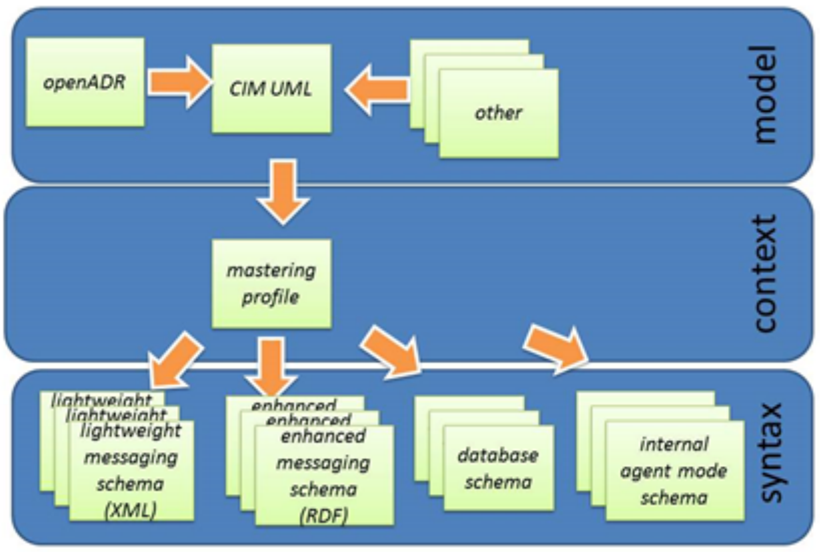

openADR
CIM UML
other
model
mastering profile
context
lightweight messaging schema (XML)
enhanced messaging schema (RDF)
database schema
internal agent mode schema
syntax

Figure 9: MAS2TERING model layers (based on Xtensible Solutions presentation)

In the scope of the base model formulation, when multiple models are integrated, inevitably, overlaps are expected. Ideally, the development of a meta-model would address the formalisation and the mapping of consistent theories and concepts but this is beyond the scope of this project. Instead, the approach here will be the pruning of models in the profile and nomination of resources via multiple name spaces and appropriate translation and mapping if necessary.

For the first iteration of a MAS2TERING data model the following domains have been identified for inclusion:

1. Temporal model, time series
2. Load profile, load descriptions (optimisation and simulation), consumption profile, aggregated profiles
3. Flexibility parameters, flexibility bid description
4. Device characteristics e.g. charge / discharge plan, device availability
5. Demand / response

However as previously described, the ontology model development process is iterative and will evolve, driven by use cases so the conceptualisations and theories modelled, extending the domains above.

Following a thorough analysis of the standards outlined previously, these were then federated manually (and automatically where the normative file formats allowed) into ontological representations using OWL constructs. Subsets of these ontologies were then aligned and extended to fully model the MAS2TERING domain. The main areas of extension regarded optimisation, device types and descriptions, demand response and load control. The results of this use case based elicitation are presented in the following sections followed by the resultant ontology.

3.4 Use case based description logic elicitation

Based on the methodology's use-case driven approach to elicit a lightweight ontology aligned with existing standards, each use case was considered in turn, with concepts and relationships and properties being elicited to satisfy its exchange requirements. These were then compared to existing standards to determine potential alignments before formalising the MAS2TERING ontology. This analysis is now briefly presented for each of the use cases defined in deliverable D6.1.

- Use Case 1 – Domestic demand optimisation

The primary existing standard relevant to this use case was the Energy@Home data model described previously, as both Device and CEMS agents exchanging messages within the context of this use case would be exchanging content regarding the HAN. The CIM specification and IEC 61968 formalize schemas for the exchange of messages regarding the network, and contain few concepts relevant to the HAN. Those concepts which they do model are aligned with in the OpenADR and Energy@Home schemas, the latter of which is also broadly aligned with the former.

- Use Case 2 – Aggregation of Dwellings

Use case 2 aims to utilise the flexibility offered by consumers through possible deferment and curtailment of loads at the multi-building level, by trading this flexibility with an aggregator agent. The aggregator agent receives a P-plan from each CEMS and buys the required flexibility based on user constraints. Aggregator agents are then able to trade flexibility between each other and relay these requests to CEMS agents. Beyond the concepts modelled for use case 1 then, use case 2 requires the modelling of multi-home concepts, and flexibility concepts, in a formal manner. This extends the scope of the ontology to overlap more significantly with those of the OpenADR and CIM standards, and so this overlap and resultant alignment will be relevant in use case 2.

- Use Case 3 - Coordination of aggregator agents

Use case 3 aims to reduce the number and impact of congestion points within a low voltage grid through the introduction of the DSO and using the FIPA's agent called Directory Facilitator (DF), in which agents wishing to advertise their services register (also called yellow pages). These agents exchange knowledge regarding the connections and congestion in the network with the aggregators, who then trade flexibility in order to improve the efficiency and resilience of the grid. Beyond the concepts modelled in use cases 1 and 2 then, use case 3 requires the modelling of connections and congestion points, as well as DSOs and further modelling of flexibility. As the concept of flexibility is central to the approach, a clear understanding of its definition and a thorough formalisation of its nature is required. The following section therefore briefly presents the perspective utilised when trading flexibility.

3.5 Domain perspective of energy flexibility

Load flexibility is here defined as a market commodity of utilised peak load reduction through optional deferment and/or curtailment of consumer demand, expressed as a unit of energy. Deferment is the shifting of a load to a time more favourable to the network operator, where the amount of flexibility is equal to the amount of energy shifted. In this way, the extent of the shift is independent to the flexibility, as the consumer sets a deadline for the task completion. This is represented in Figure 9 below, when Q_{tot} is the total energy consumption of the task, Q_f is the flexibility utilised, t0 is the earliest start time of the task, t_1 is the task completion deadline, and T_{min} is the minimum amount of time the task requires to be completed.

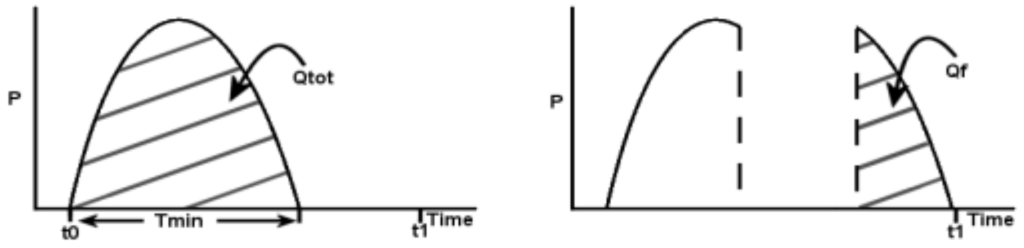


Figure 9 Ontological perspective of load deferment. Left - desired load, right - deferred load

Curtailment of load is then the supply of a quantity of energy over time which is less than the desired quantity. The flexibility is then the difference between the desired quantity and the supplied quantity, again expressed as an amount of energy. This is shown in Figure 9 and Figure 10, where t0 is the earliest start time of the task, t1 is the non-negotiable deadline of the task, Qf is the amount of flexibility utilised, and P_{min} is the minimum amount of energy to be supplied (such as when a heating device must meet a minimum room temperature).

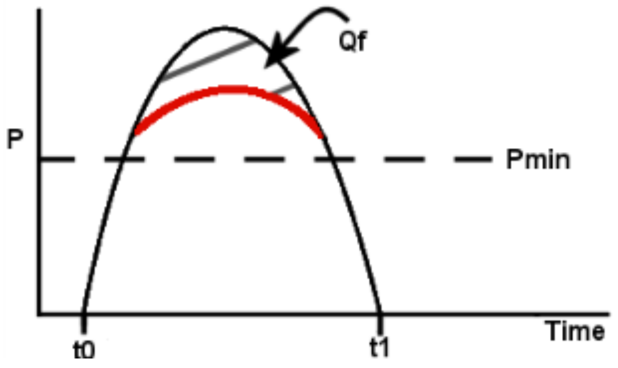


Figure 10 Domain perspective of load curtailment. Black profile - desired load, red line - curtailed load

Based on the use case analyses and the definitions of flexibility presented, devices were then categorised according to their likely flexibilities and types of variability. Of course, the actual flexibility offered for any appliance will be decided by the consumer's preferences, as well as the appliance's technological capabilities.

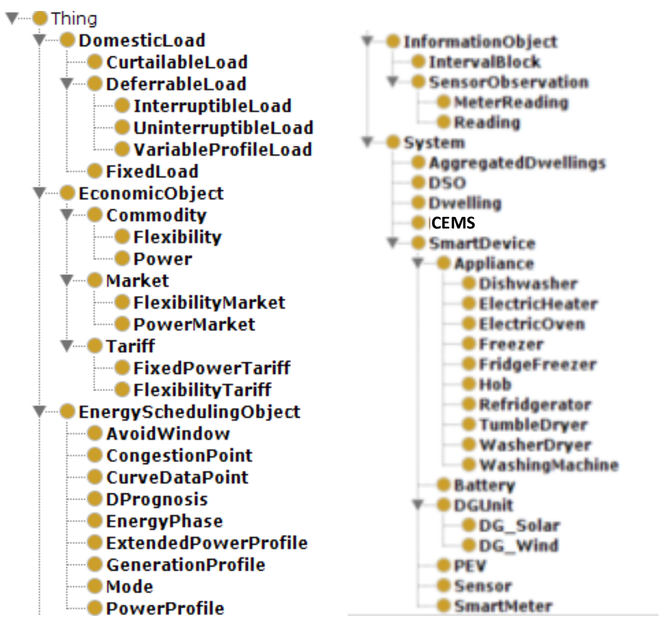
Table 3 Classifications of likely flexibilities of devices

	Non-interruptible	Interruptible	Variable Profile
Curtailable	N/A	N/A	Electric heater

Deferrable	Washing machine	Dishwasher	PEV, electric oven, tumble dryer, kettle
Fixed	Freezer, fridge, lights	N/A	N/A

3.6 Candidate generic domain ontology - OWL constructs

Following the elicitation of domain knowledge through the use-case and the standards-analysis driven process, the resultant knowledge was formalised in description logic using basic OWL constructs so as to produce a candidate ontology, generic across potential implementations. This was conducted in the Protégé software, and serves to produce an ontology with value outside of its MAS application, as it is also suitable for web service deployment, or direct use in a C++ or Java program through a relevant OWL library. This was then extended with more application-specific knowledge, which served to couple the ontology closely with the agents and protocols described in this deliverable, and in D5.3. The generic ontology is presented first using OWL constructs due to their likely familiarity to the intended readership of ontological modellers, then the extended ontology is presented, including its coupling with MAS2TERING protocols, and finally in the JADE format utilised in MAS2TERING. Given the difference in nature between the data schemas which these standards present and that of a JADE ontology, the federation and re-use approach adopted represents a best-case for future compliance with existing standards if they are expressed normatively in an ontological format in the future. Figure 11, Figure 12, Figure 13, Figure 14, and Figure 15 present the main concepts and the relationships formalised in the generic ontology, and the data properties of these concepts.



- Thing
  - DomesticLoad
    - CurtailableLoad
    - DeferrableLoad
      - InterruptibleLoad
      - UninterruptibleLoad
      - VariableProfileLoad
    - FixedLoad
  - EconomicObject
    - Commodity
      - Flexibility
      - Power
    - Market
      - FlexibilityMarket
      - PowerMarket
    - Tariff
      - FixedPowerTariff
      - FlexibilityTariff
  - EnergySchedulingObject
    - AvoidWindow
    - CongestionPoint
    - CurveDataPoint
    - DPrognosis
    - EnergyPhase
    - ExtendedPowerProfile
    - GenerationProfile
    - Mode
    - PowerProfile
  - InformationObject
    - IntervalBlock
    - SensorObservation
      - MeterReading
      - Reading
  - System
    - AggregatedDwellings
    - DSO
    - Dwelling
    - CEMS
    - SmartDevice
      - Appliance
        - Dishwasher
        - ElectricHeater
        - ElectricOven
        - Freezer
        - FridgeFreezer
        - Hob
        - Refridgerator
        - TumbleDryer
        - WasherDryer
        - WashingMachine
      - Battery
      - DGUnit
        - DG_Solar
        - DG_Wind
      - PEV
      - Sensor
      - SmartMeter

Figure 11 Full generic OWL model class list

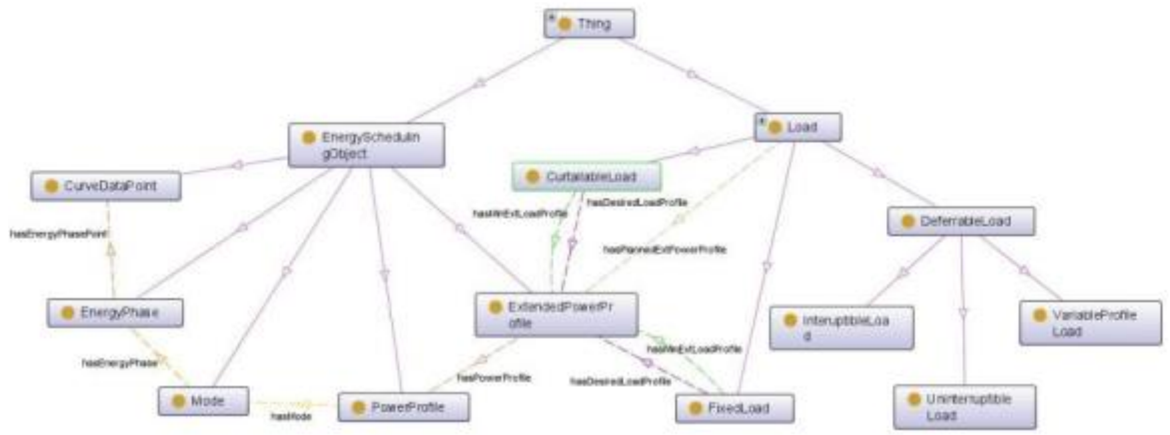


Figure 12 OWL MVD focusing on energy scheduling concepts

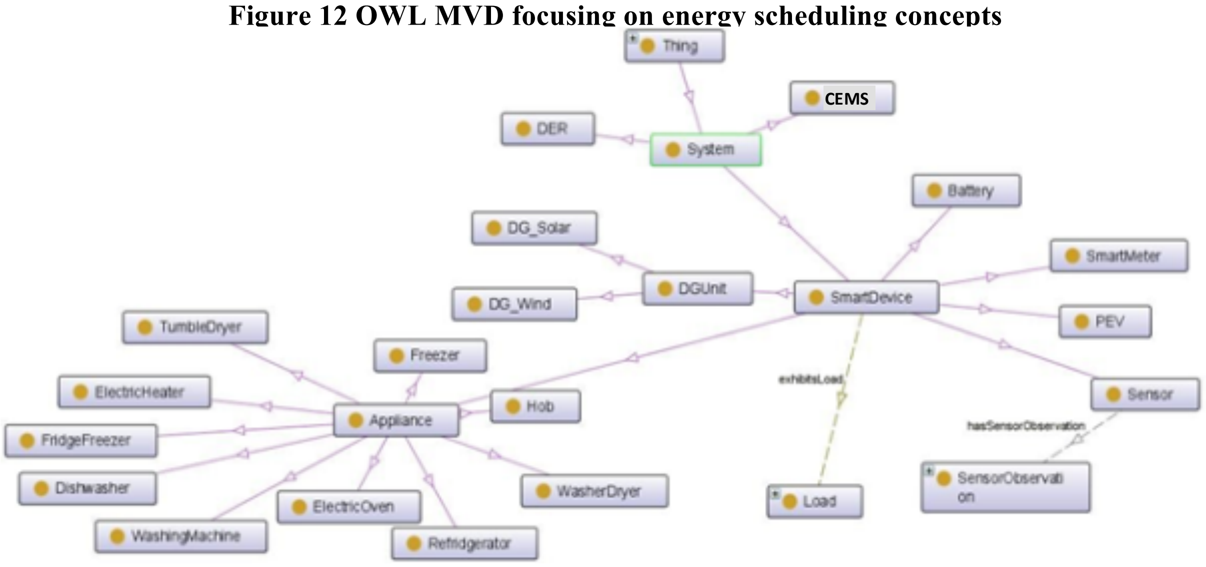


Figure 13 OWL MVD focusing on device concepts

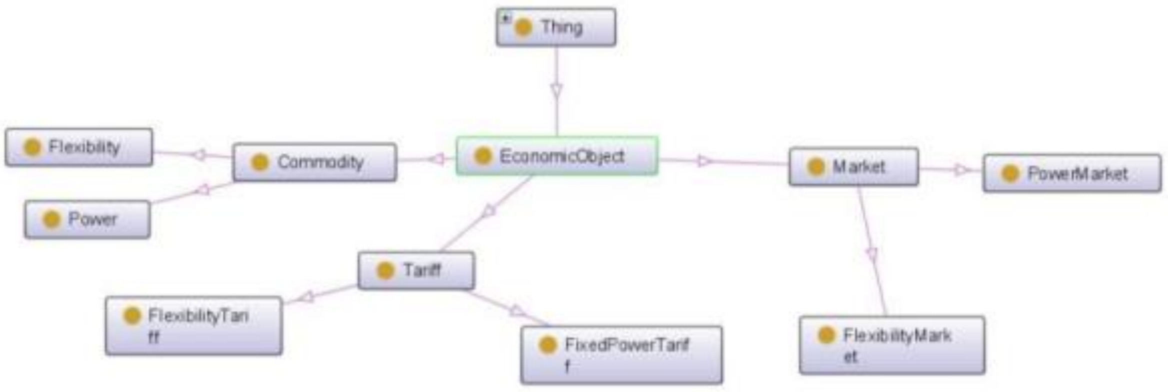
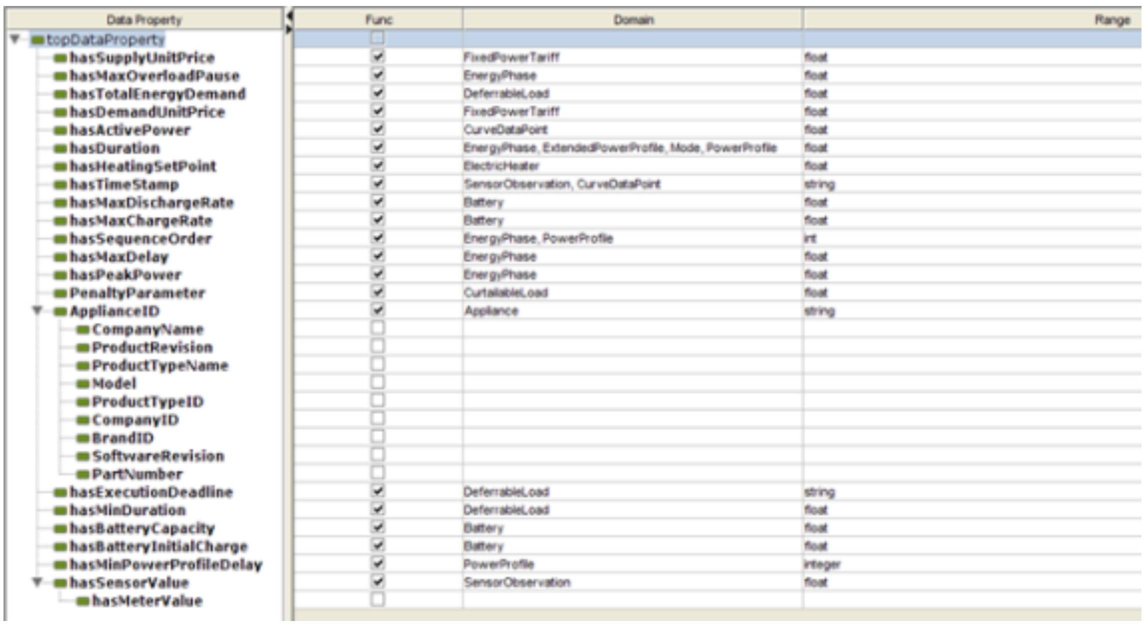


Figure 14 OWL MVD focussing on economic concepts

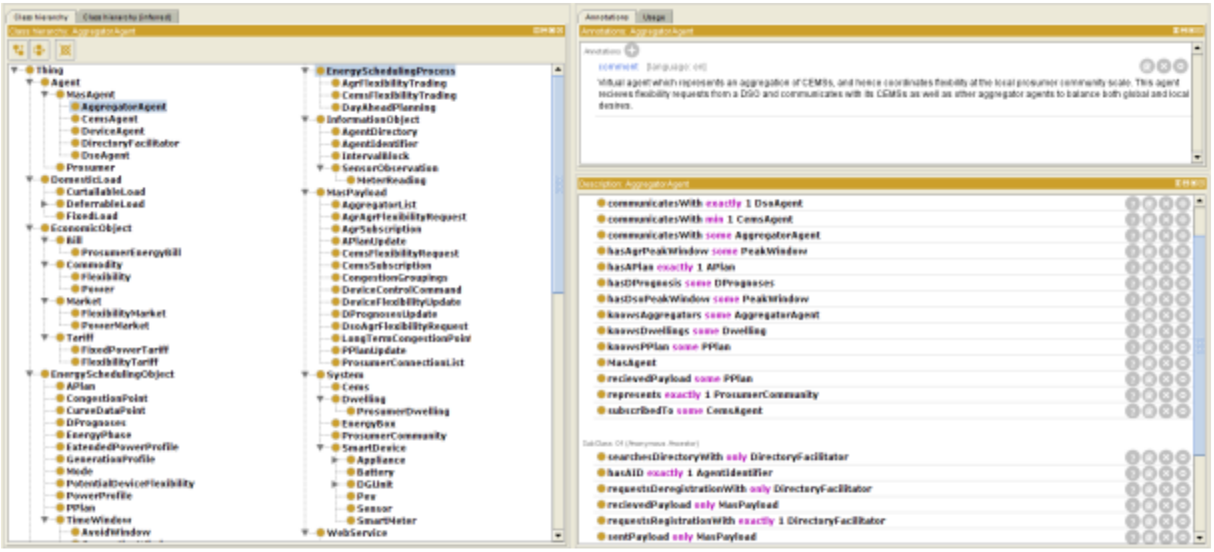


Data Property	Func	Domain	Range
topDataProperty	☐		
hasSupplyUnitPrice	✓	FixedPowerTariff	float
hasMaxOverloadPause	✓	EnergyPhase	float
hasTotalEnergyDemand	✓	DeferrableLoad	float
hasDemandUnitPrice	✓	FixedPowerTariff	float
hasActivePower	✓	CurveDataPoint	float
hasDuration	✓	EnergyPhase, ExtendedPowerProfile, Mode, PowerProfile	float
hasHeatingSetPoint	✓	ElectricHeater	float
hasTimeStamp	✓	SensorObservation, CurveDataPoint	string
hasMaxDischargeRate	✓	Battery	float
hasMaxChargeRate	✓	Battery	float
hasSequenceOrder	✓	EnergyPhase, PowerProfile	int
hasMaxDelay	✓	EnergyPhase	float
hasPeakPower	✓	EnergyPhase	float
PenaltyParameter	✓	CurtailableLoad	float
ApplianceID	✓	Appliance	string
CompanyName	☐		
ProductRevision	☐		
ProductTypeName	☐		
Model	☐		
ProductTypeID	☐		
CompanyID	☐		
BrandID	☐		
SoftwareRevision	☐		
PartNumber	☐		
hasExecutionDeadline	✓	DeferrableLoad	string
hasMinDuration	✓	DeferrableLoad	float
hasBatteryCapacity	✓	Battery	float
hasBatteryInitialCharge	✓	Battery	float
hasMinPowerProfileDelay	✓	PowerProfile	integer
hasSensorValue	✓	SensorObservation	float
hasMeterValue	☐		

Figure 15 Generic OWL model data property specification

3.7 Candidate protocol payload ontology – OWL constructs

Following the process of formalising a generic domain ontology aligned with existing standards, this was then extended to closely couple with the requirements of the MAS2TERING agents and protocols specified. This involved formalising concepts, which would not themselves be included in message payloads, but which would be necessary to contextualise the domain fully from the perspective of the agents. Primarily, the agents themselves, as well as the message payloads, were described as classes with required properties, resulting in 216 named concepts. This full class hierarchy is presented in Figure 16 below, along with an example of the required property descriptions for the new classes.



```
Class hierarchy | Class hierarchy (inferred)
Class hierarchy: AggregatorAgent

▼ Thing
  ▼ Agent
    ▼ MasAgent
        AggregatorAgent
        CemsAgent
        DeviceAgent
        DirectoryFacilitator
        DsoAgent
      Prosumer
  ▼ DomesticLoad
      CurtailableLoad
    ► DeferrableLoad
      FixedLoad
  ▼ EconomicObject
    ▼ Bill
        ProsumerEnergyBill
    ▼ Commodity
        Flexibility
        Power
    ▼ Market
        FlexibilityMarket
        PowerMarket
    ▼ Tariff
        FixedPowerTariff
        FlexibilityTariff
  ▼ EnergySchedulingObject
      APlan
      CongestionPoint
      CurveDataPoint
      DPrognoses
      EnergyPhase
      ExtendedPowerProfile
      GenerationProfile
      Mode
      PotentialDeviceFlexibility
      PowerProfile
      PPlan
    ▼ TimeWindow
        AvoidWindow
  ▼ EnergySchedulingProcess
      AgrFlexibilityTrading
      CemsFlexibilityTrading
      DayAheadPlanning
  ▼ InformationObject
      AgentDirectory
      AgentIdentifier
      IntervalBlock
    ▼ SensorObservation
        MeterReading
  ▼ MasPayload
      AggregatorList
      AgrAgrFlexibilityRequest
      AgrSubscription
      APlanUpdate
      CemsFlexibilityRequest
      CemsSubscription
      CongestionGroupings
      DeviceControlCommand
      DeviceFlexibilityUpdate
      DPrognosesUpdate
      DsoAgrFlexibilityRequest
      LongTermCongestionPoint
      PPlanUpdate
      ProsumerConnectionList
  ▼ System
      Cems
    ▼ Dwelling
        ProsumerDwelling
      EnergyBox
      ProsumerCommunity
    ▼ SmartDevice
      ► Appliance
        Battery
      ► DGUnit
        Pev
        Sensor
        SmartMeter
  ▼ WebService
```

```
Annotations | Usage
Annotations: AggregatorAgent
Annotations +
  comment [language: en]
  Virtual agent which represents an aggregation of CEMSs, and hence coordinates flexibility at the local prosumer community scale. This agent recieves flexibility requests from a DSO and communicates with its CEMSs as well as other aggregator agents to balance both global and local desires.

Description: AggregatorAgent
  communicatesWith exactly 1 DsoAgent
  communicatesWith min 1 CemsAgent
  communicatesWith some AggregatorAgent
  hasAgrPeakWindow some PeakWindow
  hasAPlan exactly 1 APlan
  hasDPrognosis some DPrognoses
  hasDsoPeakWindow some PeakWindow
  knowsAggregators some AggregatorAgent
  knowsDwellings some Dwelling
  knowsPPlan some PPlan
  MasAgent
  recievedPayload some PPlan
  represents exactly 1 ProsumerCommunity
  subscribedTo some CemsAgent

SubClass Of (Anonymous Ancestor)
  searchesDirectoryWith only DirectoryFacilitator
  hasAID exactly 1 AgentIdentifier
  requestsDeregistrationWith only DirectoryFacilitator
  recievedPayload only MasPayload
  requestsRegistrationWith exactly 1 DirectoryFacilitator
  sentPayload only MasPayload
```

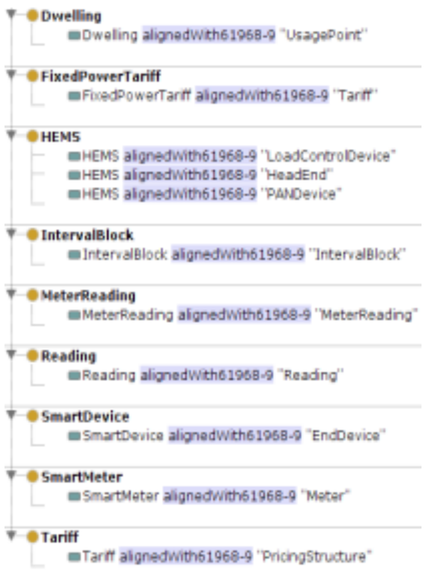
Figure 16: Full class list for MAS-coupled ontology, and example of class property specification

3.8 Candidate ontology - JADE constructs

In order to utilise the ontology to formalise the semantics of FIPA-ACL encoded payloads, the OWL candidate ontology was converted into a set of JADE concepts and predicate bean classes. This process has been automated by combining Apache Jena (to interpret the OWL file expressing the candidate ontology) and Eclipse JDT (to manipulate Java source code). The JADE bean generation process is detailed in MAS2TERING deliverable D5.3. Annex B.1 provides more information on the conversion process.

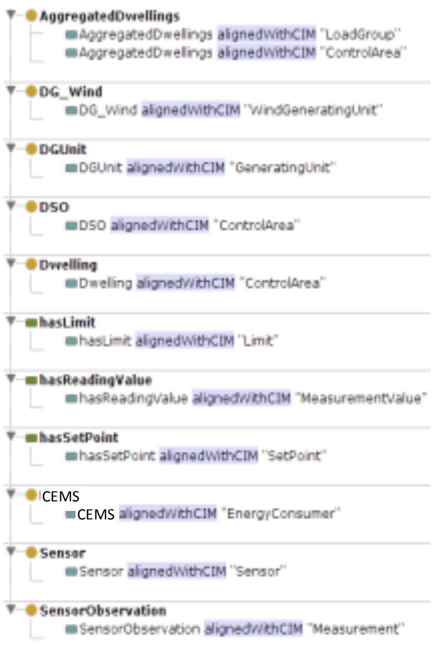
3.9 Alignments with existing standards

Alignments with the existing standards have been formalised as OWL annotations, as presented in Figure 17, Figure 18, and Figure 19 below. Whilst it would be incorrect to state that this represents full compliance or alignment with the standard (as this would require further testing and refinement), it demonstrates broad coherence with the domain perspectives of the existing standards, and paves the way for genuine compliance if the existing standards are developed into full semantic models in the future.



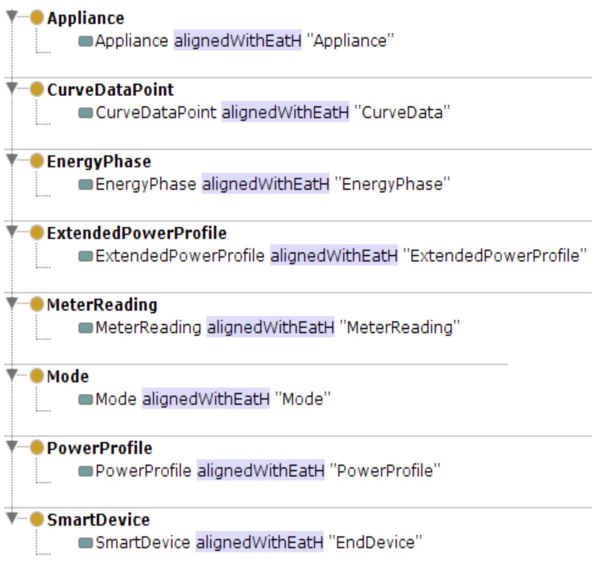
```
Dwelling
    Dwelling alignedWith61968-9 "UsagePoint"
FixedPowerTariff
    FixedPowerTariff alignedWith61968-9 "Tariff"
HEMS
    HEMS alignedWith61968-9 "LoadControlDevice"
    HEMS alignedWith61968-9 "HeadEnd"
    HEMS alignedWith61968-9 "PANDevice"
IntervalBlock
    IntervalBlock alignedWith61968-9 "IntervalBlock"
MeterReading
    MeterReading alignedWith61968-9 "MeterReading"
Reading
    Reading alignedWith61968-9 "Reading"
SmartDevice
    SmartDevice alignedWith61968-9 "EndDevice"
SmartMeter
    SmartMeter alignedWith61968-9 "Meter"
Tariff
    Tariff alignedWith61968-9 "PricingStructure"
```

Figure 17 Alignment of ontological concepts with IEC 61968-9



```
AggregatedDwellings
    AggregatedDwellings alignedWithCIM "LoadGroup"
    AggregatedDwellings alignedWithCIM "ControlArea"
DG_Wind
    DG_Wind alignedWithCIM "WindGeneratingUnit"
DGUnit
    DGUnit alignedWithCIM "GeneratingUnit"
DSO
    DSO alignedWithCIM "ControlArea"
Dwelling
    Dwelling alignedWithCIM "ControlArea"
hasLimit
    hasLimit alignedWithCIM "Limit"
hasReadingValue
    hasReadingValue alignedWithCIM "MeasurementValue"
hasSetPoint
    hasSetPoint alignedWithCIM "SetPoint"
CEMS
    CEMS alignedWithCIM "EnergyConsumer"
Sensor
    Sensor alignedWithCIM "Sensor"
SensorObservation
    SensorObservation alignedWithCIM "Measurement"
```

Figure 18 Alignment of ontological concepts with CIM



Appliance
Appliance alignedWithEatH "Appliance"
CurveDataPoint
CurveDataPoint alignedWithEatH "CurveData"
EnergyPhase
EnergyPhase alignedWithEatH "EnergyPhase"
ExtendedPowerProfile
ExtendedPowerProfile alignedWithEatH "ExtendedPowerProfile"
MeterReading
MeterReading alignedWithEatH "MeterReading"
Mode
Mode alignedWithEatH "Mode"
PowerProfile
PowerProfile alignedWithEatH "PowerProfile"
SmartDevice
SmartDevice alignedWithEatH "EndDevice"

Figure 19 Alignment of ontological concepts with Energy@Home

4 The Agents model

The MAS2TERING platform will perform as follows: (1) the aggregation of atomic 'Behaviours' into the AgentSpecification, (2) using the factory to build the final JADE agent from the Agent specifications, and (3) Launching the JADE Agents objects within the platform with the Factory.

The agent model is extended by modelling the specific requirements extracted from both: the USEF framework and the use cases. We start by defining the types of agents and their roles that will be defined in the management system. The description tables of the four types of agents and the seven subtypes, which will be used for testing the three use cases, are presented following this template:

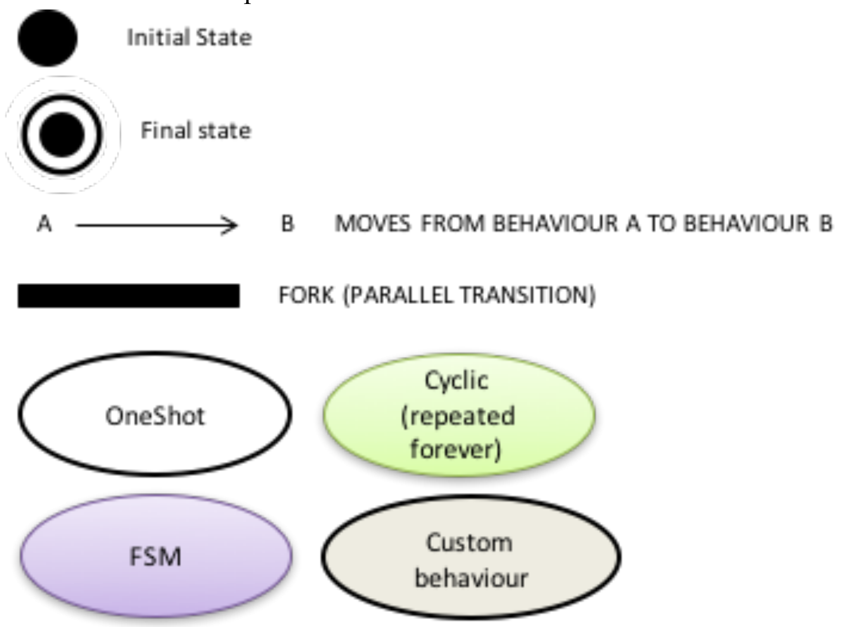
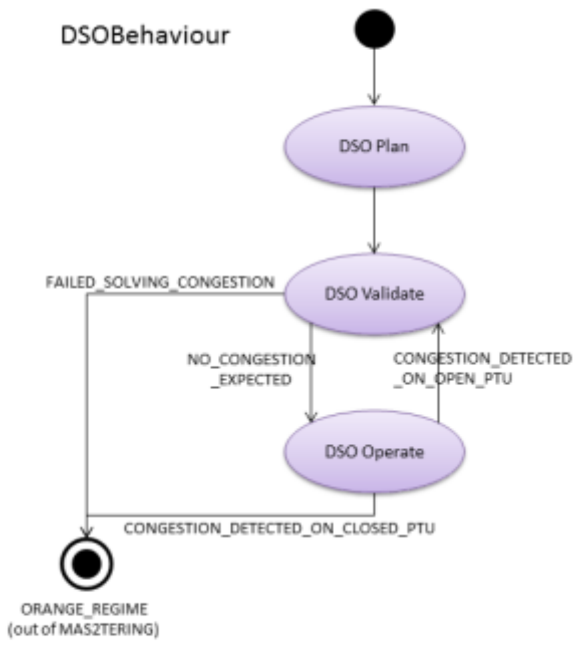
Agent class	Agent type name as specified in D2.2
Subtype	Agent subtypes (if any)
Description	Textual description of the role that the agent plays
Finite State Machine (FSM)	A high-level visual representation of the behaviour of the agent is represented using a Finite State Machine. Figure 20 shows a legend for the created Finite State Machines. More details about how to read and understand FSMs is provided in Annex A.1.  Initial State Final state A B MOVES FROM BEHAVIOUR A TO BEHAVIOUR B FORK (PARALLEL TRANSITION) OneShot Cyclic (repeated forever) FSM Custom behaviour **Figure 20: Overview of the FSM components and used syntaxes**

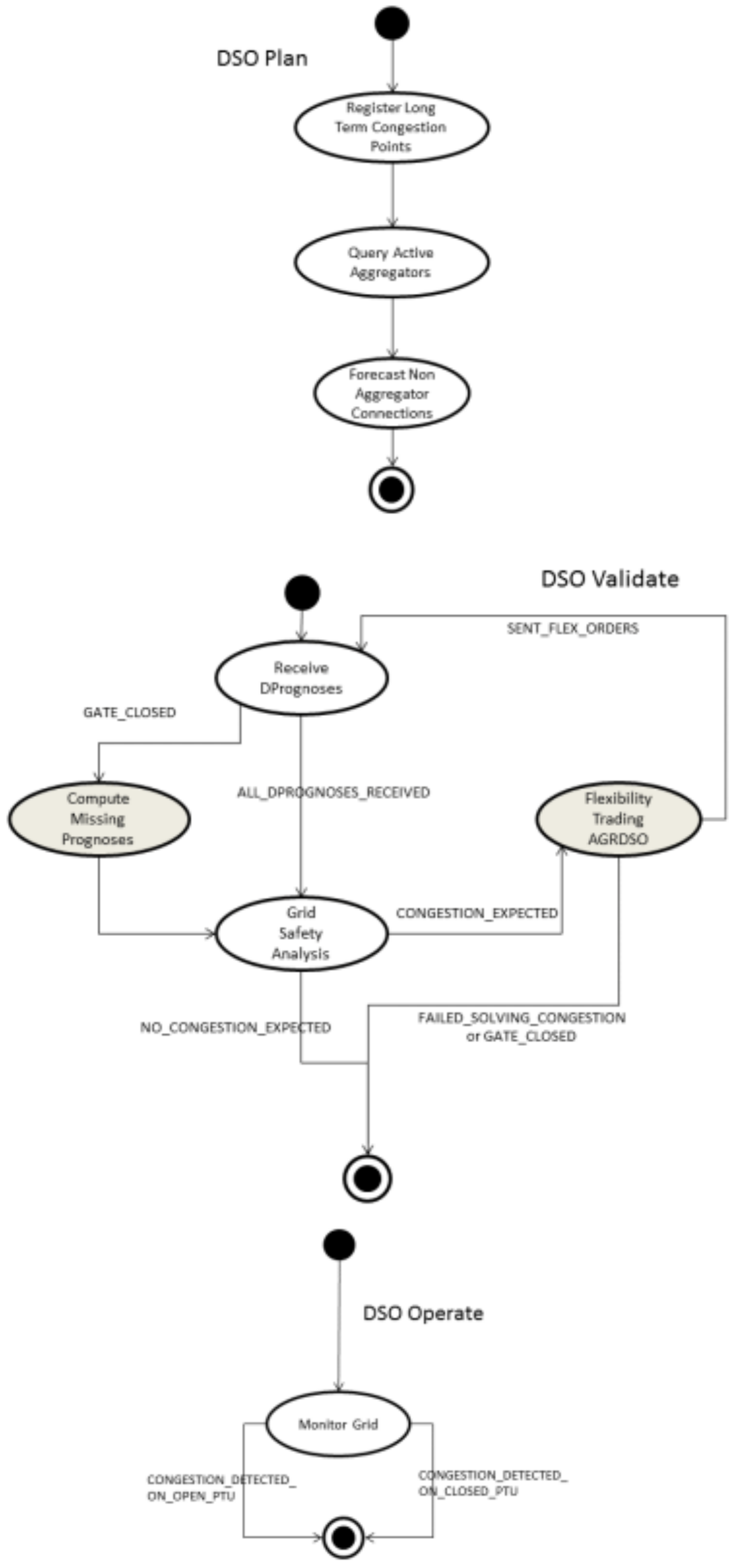
Table 4 Agent type description template

In the following tables, we provide the description of each of the agent types and subtypes.

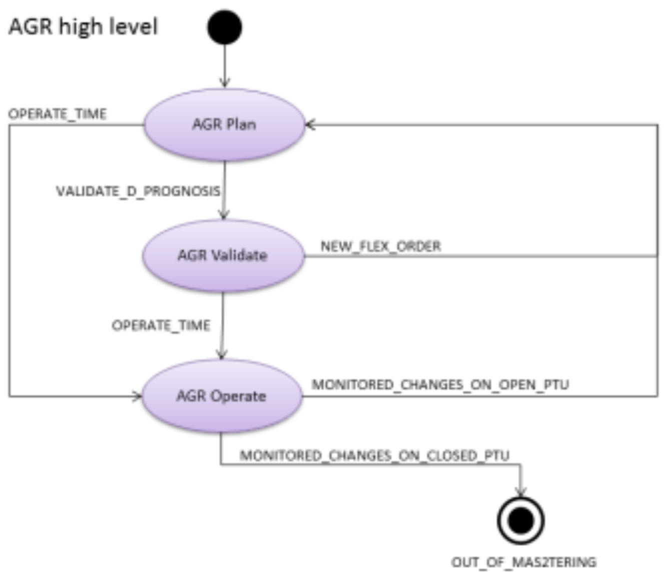
4.1.1 Distribution System Operator (DSO) Agent

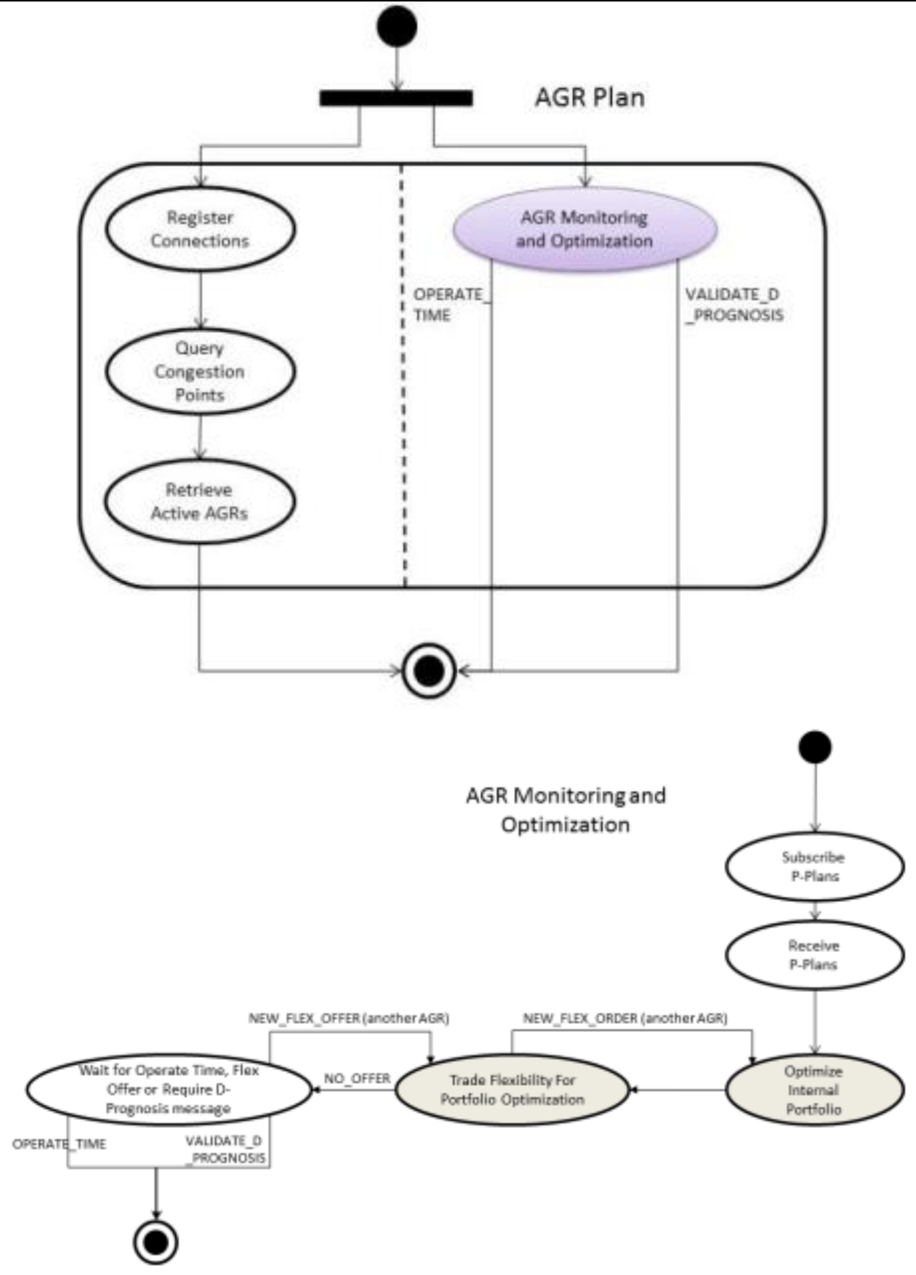
Agent class name	**Distribution System Operator (DSO)**
Subtypes	--
Description	The DSO is responsible of the cost-effective distribution of electricity to end

	consumers within statutory limits for the region of the distribution grid for which it is responsible (See D1.6). However, due to simplicity purposes, the DSO agent of the designed MAS system for MAS2TERING will only provide the functionality for managing the grid congestion. During the planning phase the DSO will publish the locations in the grid where overload may occur (i.e. based on its analysis of the trends in energy flows in its grids). After this, the DSO mainly participates in the validate phase, in which it takes place the alignment Aggregator/DSO. In more detail, the DSO whether the demand and supply of energy can be distributed safely without any limitations based on the received D-prognoses from AGRs. If congestion is expected to occur, the DSO procures flexibility from AGRs to solve the grid capacity issues. The following FSMs show: a high-level FSM (DSOBehaviour), which is composed of the lower-level FSMs illustrated (DSO Plan, DSO Validate, and DSO Operate).
Finite State Machine	 DSOBehaviour DSO Plan DSO Validate FAILED_SOLVING_CONGESTION NO_CONGESTION _EXPECTED CONGESTION_DETECTED _ON_OPEN_PTU DSO Operate CONGESTION_DETECTED_ON_CLOSED_PTU ORANGE_REGIME (out of MAS2TERING)

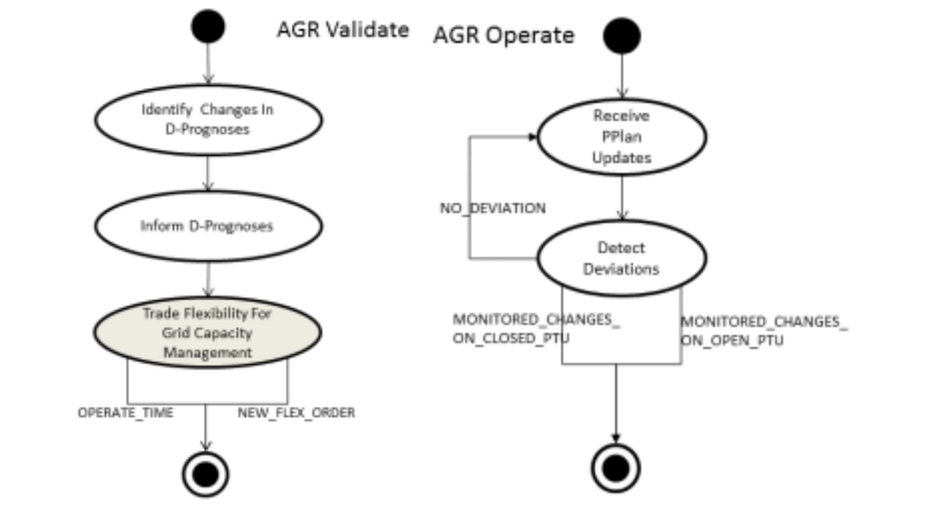
	 DSO Plan Register Long Term Congestion Points Query Active Aggregators Forecast Non Aggregator Connections DSO Validate SENT_FLEX_ORDERS Receive DPrognoses GATE_CLOSED Compute Missing Prognoses ALL_DPROGNOSES_RECEIVED Flexibility Trading AGRDSO Grid Safety Analysis CONGESTION_EXPECTED NO_CONGESTION_EXPECTED FAILED_SOLVING_CONGESTION or GATE_CLOSED DSO Operate Monitor Grid CONGESTION_DETECTED_ ON_OPEN_PTU CONGESTION_DETECTED_ ON_CLOSED_PTU

4.1.2 Aggregator Agent

Agent class name	**Aggregator (AGR)**
Subtypes	--
Description	The role of the aggregator is to manage the flexibility produced by a portfolio of prosumers. Aggregators compete to each other to provide flexibility to other participants in the flexibility market (i.e. DSO, other aggregators) (see D1.6). To do this, the aggregator participates in the following phases: Plan: The plan phase starts when the aggregator collects P-plans from the prosumers it is representing. Then, the aggregator optimises its portfolio based on its client needs and provides an A-plan which is the expected consumption profile during the day of delivery of the portfolio of prosumers of the aggregator. Validate: The aggregator sends/updates a D-prognoses per congestion point to the DSO. The aggregator handles flexibility requests from the DSO entering into negotiations to provide flexibility from its portfolio. Operate: The aggregator adheres to its D-prognoses and A-plans. To do this, the aggregator needs to monitor the P-plans of their prosumers, and if any change in the forecast is detected, it has to process the corresponding affected plans. In the following, a high-level FSM shows the aggregator phases (AGR high level FSM). Each hierarchical FSM is then illustrated (AGR plan FSM, AGR Validate FSM, and AGR Operate FSM) .
Finite State Machine	 AGR high level OPERATE_TIME AGR Plan VALIDATE_D_PROGNOSIS AGR Validate NEW_FLEX_ORDER OPERATE_TIME AGR Operate MONITORED_CHANGES_ON_OPEN_PTU MONITORED_CHANGES_ON_CLOSED_PTU OUT_OF_MAS2TERING

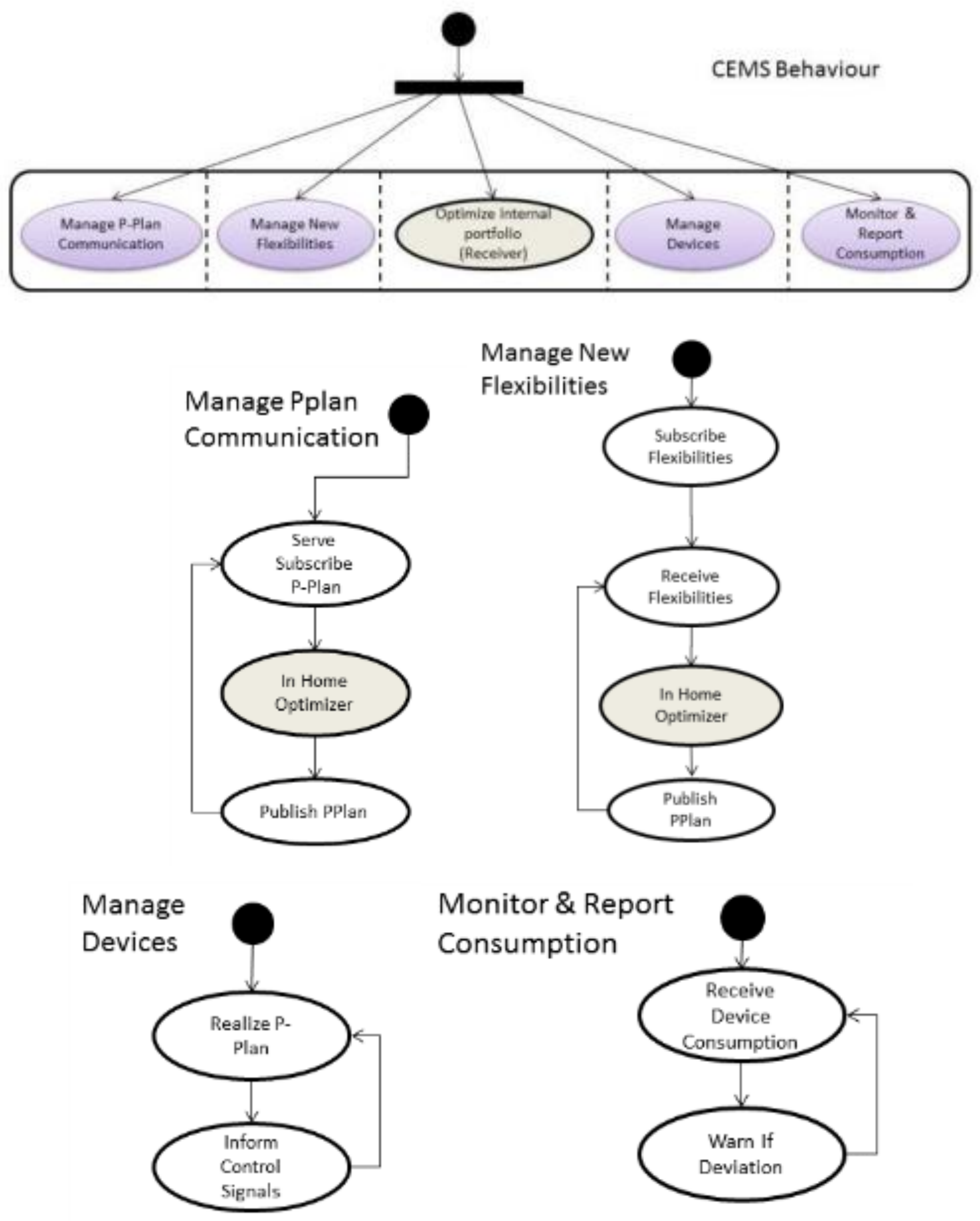
	

AGR Plan
Register Connections
Query Congestion Points
Retrieve Active AGRs
AGR Monitoring and Optimization
OPERATE_TIME
VALIDATE_D_PROGNOSIS
AGR Monitoring and Optimization
Subscribe P-Plans
Receive P-Plans
Optimize Internal Portfolio
NEW_FLEX_ORDER (another AGR)
Trade Flexibility For Portfolio Optimization
NEW_FLEX_OFFER (another AGR)
NO_OFFER
Wait for Operate Time, Flex Offer or Require D-Prognosis message
OPERATE_TIME
VALIDATE_D_PROGNOSIS

	 AGR Validate Identify Changes In D-Prognoses Inform D-Prognoses Trade Flexibility For Grid Capacity Management OPERATE_TIME NEW_FLEX_ORDER AGR Operate Receive PPlan Updates NO_DEVIATION Detect Deviations MONITORED_CHANGES_ON_CLOSED_PTU MONITORED_CHANGES_ON_OPEN_PTU

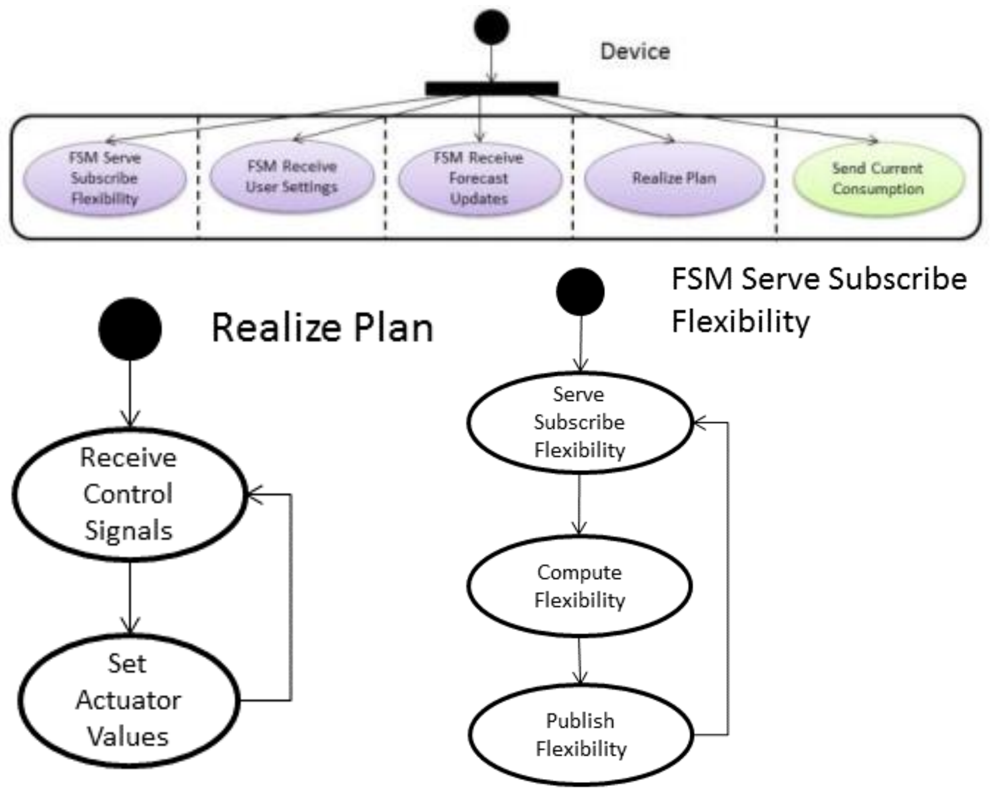
4.1.3 Consumer Energy Management System (CEMS)

Agent class name	Consumer Energy Management System (CEMS)
Subtypes	--
Description	The CEMS controls and optimises the flexibility of the prosumer. It aims at minimising the corresponding prosumer energy bill (i.e. it carries out an internal optimisation behind the meter). To do this, the CEMS has access to all the devices of the corresponding prosumer as well as to the tariff that the prosumer contracted with the supplier. The outcome of this is twofold: the configuration of the controllable devices and the prosumer energy consumption plan (the P-plan). The in-home optimiser can minimise the bill by using flexibilities to perform TOU optimisation, controlling the maximum load or apply a self-balancing. In case that the prosumer subscribes to an AGR, the CEMS serves as a first aggregation level being in charge of communicating the P-plan to the AGR and handle messages of offers for flexibility from the AGR in order to consider them in the in-home optimisation when updating P-plans. A CEMS is a logical entity and not necessarily a physical device. The CEMS is also in charge of realising the agreed P-plan when entering the corresponding Operate phase by sending the corresponding control signals to the in-home controllable devices (i.e. executing all the corresponding flexibility orders).

Finite State Machine	 CEMS Behaviour Manage P-Plan Communication Manage New Flexibilities Optimize internal portfolio (Receiver) Manage Devices Monitor & Report Consumption Manage Pplan Communication Serve Subscribe P-Plan In Home Optimizer Publish PPlan Manage New Flexibilities Subscribe Flexibilities Receive Flexibilities In Home Optimizer Publish PPlan Manage Devices Realize P-Plan Inform Control Signals Monitor & Report Consumption Receive Device Consumption Warn If Deviation **Figure 21. FSM CEMS agent**

4.1.4 Device Agent

Agent class name	Device
Subtypes	Curtailable Load, Deferrable Load, Fixed Load, Generator, Transmission Line, Battery, External Tie

Description	It represents the energy-consuming and/or producing/storage end-systems that can be actively controlled or not. This agent defines a set of flexibilities by using its device model constraints, user settings and the forecasts (when applicable). Devices interact with CEMS to provide it their flexibilities. Flexibility of a non-controllable device is assumed as a fixed power demand curve subject to some constraints. In case of self-controlled devices, the flexibility may be a flexible regime in which the device can work, for instance to shift, increase or decrease their energy consumption or production. They also present another interaction in which CEMS sends the devices the control signals for setting up their actuators.
Finite State Machine	 Device FSM Serve Subscribe Flexibility FSM Receive User Settings FSM Receive Forecast Updates Realize Plan Send Current Consumption Realize Plan Receive Control Signals Set Actuator Values FSM Serve Subscribe Flexibility Serve Subscribe Flexibility Compute Flexibility Publish Flexibility

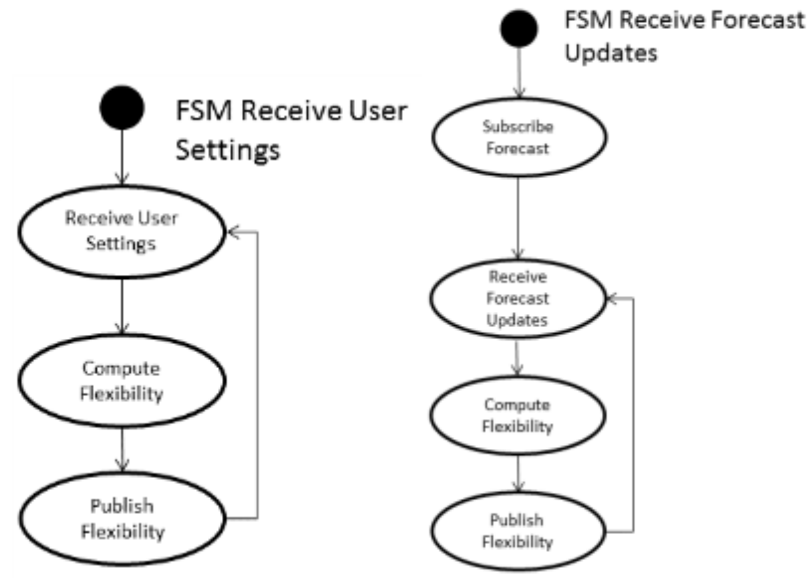


Figure 22. FSM Device agent

The Device Agent is the only one that needs a more specific definition based on each kind of device. Since it represents the physical devices of the power grid, each of them with its own specification and constraints, several agents' subtypes that extend the Device agent must be defined. We identified and modelled one agent per physical device of the grid. More precisely, seven agents are needed for testing the three use cases defined for MAS2TERING project. Figure 23 shows the UML diagram of these agents:

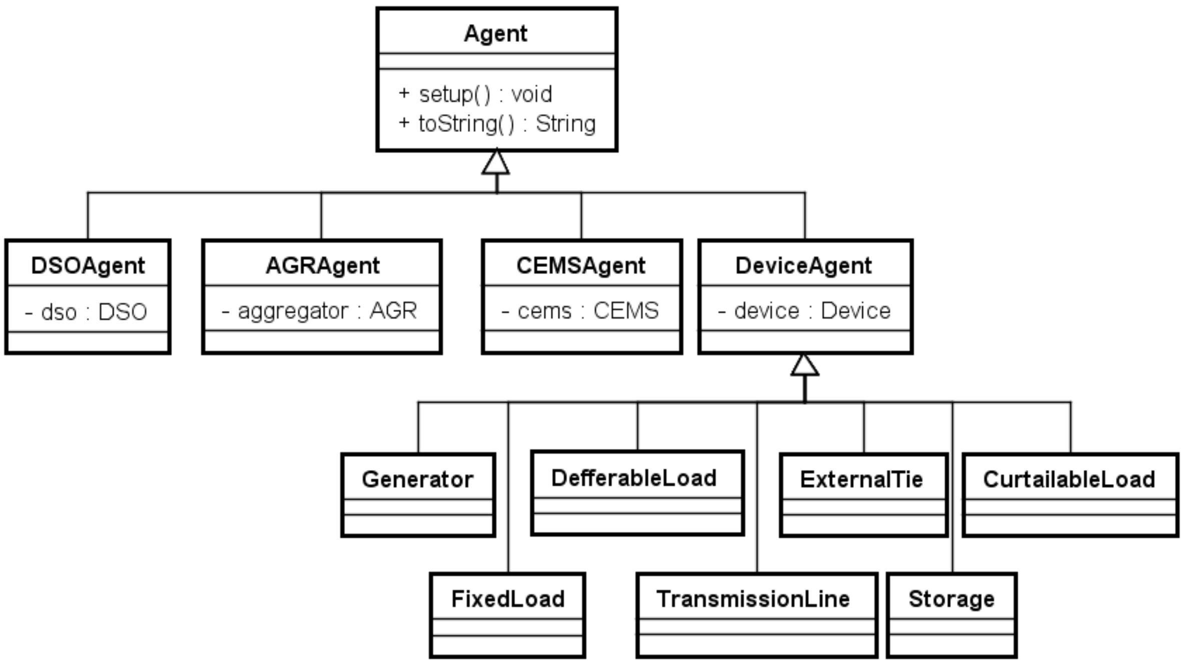


Figure 23 Agents hierarchy in JADE

Next subsections provide a short definition for each of the subtypes of Device Agents that can be extracted from the Smart Grid model defined for MAS2TERING project and that will be used in the use cases. The constrains of each device subclass are described inside the constraints and objectives component of the MAS platform (Chapter 5).

4.1.4.1 Generator

A generator is a single-terminal device that supplies power to the grid. It works with a power schedule and generates power within a range and may have a limit for changing power levels from one period to the next.

4.1.4.2 Curtailable Load

A curtailable load is a single-terminal device that has a desirable load provide and a real power load. It penalises the energy shortfall between a desired load profile and the delivered power. Some device extensions may include time-varying and nonlinear penalties on the energy shortfall.

4.1.4.3 Deferrable load

A deferrable load is a single terminal device that requires its execution to be done by a certain time specified by the user. MAS2TERING defines different subtypes of deferrable loads depending on: (i) if the corresponding appliance can be stopped once started; and (ii) the way that it is specified the appliance power consumption profile (i.e. if it needs to match particular power consumption profile or it just must consume a minimum amount of power over the given interval of time) .

4.1.4.4 Storage

A battery is a single terminal device with power schedule, which can take in or deliver energy, depending on whether it is charging or discharging.

4.1.4.5 Fixed load

A fixed energy load is a single terminal device with zero cost function which consists of a desired consumption profile that must be satisfied in each period. It does not provide any flexibility and cannot be controlled.

4.1.4.6 External tie

An external tie is a single terminal device that represents a connection to an external source of power. In MAS2TERING these external ties are used to represent the individual tariffs that each home owner has with the utility. Such tariffs are represented as external ties since the price of energy is fixed for the tariff contract and independent of the prices given to participate in the local flexibility market.

4.1.4.7 Transmission line

A transmission line is a device with two terminals (i.e. two power schedules) that transports power across some distance. It works at the distribution level and may carry some energy losses.

4.1.5 Behaviours

In the following, we provide a list of behaviours that are used by the agents in this component of the MAS platform. The behaviours are defined by extending the existing behaviours in JADE, which are shown in Figure 24. The figure shows the main classes in the Behaviour package: each behaviour can be viewed as a coherent, potentially complex ability that provides the agent with a certain functionality (e.g. deliberating for solving a given type of problem, interacting with other agents using a given protocol, using external application Matlab, etc.). As far as possible, a behaviour is defined relatively independently of the other behaviours. If dependencies are needed, dependent behaviours are preferably being related by a composite behaviour (e.g. first receive new flexibility, then perform local optimization), by exchanging information through agent memory (e.g. compute a PPlan in a first behaviour and then send it in another) and possibly by "causing" external influences (e.g. send request for PPlan in a first behaviour and then receive incoming PPlans in a second) (complex behaviour). Thus, each behaviour can be designed while requiring only a very limited understanding of the whole application it may be used for. In addition to simplifying behaviour design, this last point makes possible to reuse the same behaviour multiple times.

As we can see in Figure 24, there are two main types of behaviours: simple and composite. Those behaviours take the same meaning as in JADE: a composite behaviour is a finite state machine in which each state represents a micro-behaviour and a transition represents the flow of execution from one micro-behaviour to the next.

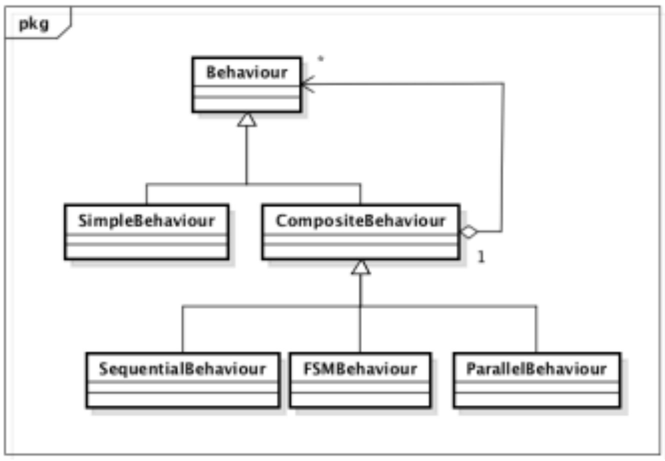


Figure 24: Main classes in the Behaviour package.

The MAS2TERING-specific behaviours are described following this template:

Table 5 Behaviours description template

Behaviour name	***Behaviour name (as specified in D2.2)***
Behaviour type	*Type of JADE behaviour*
Description	*Textual description of the behaviour*
Inputs	*Inputs that the behaviour receives (from agent memory)*
Outputs	*Outputs that the behaviour provide (influencing agent memory or the*

	execution of other behaviours connected by a composite behaviour)

In the following tables, we specify each of the identified behaviours for the four agent classes described in the previous section. Given that no behaviour is used by multiple agents, we sorted these behaviours by agents.

Some of these behaviours are "final/working" behaviours (describing actions for the agents) some others are composite (describing a composition of behaviours) and some others are undefined (to be defined at a later time of MAS2TERING.

4.1.5.1 DSO-Level Behaviours

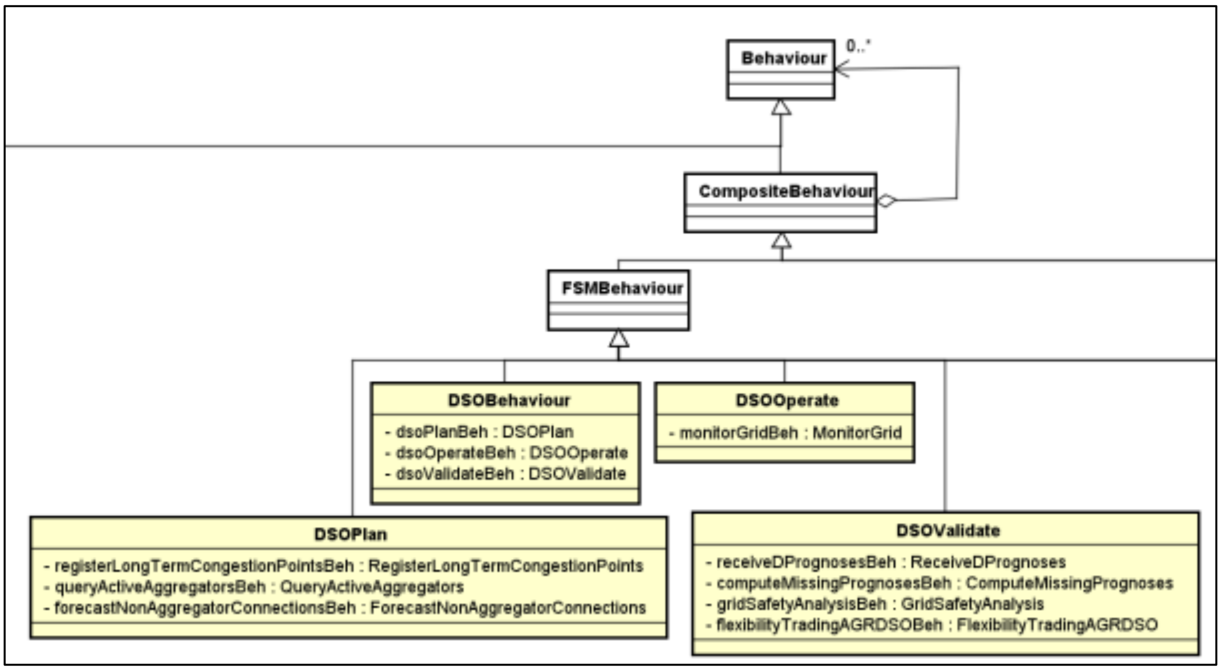

Behaviour
0..*
CompositeBehaviour
FSMBehaviour
DSOBehaviour
- dsoPlanBeh : DSOPlan
- dsoOperateBeh : DSOOperate
- dsoValidateBeh : DSOValidate
DSOOperate
- monitorGridBeh : MonitorGrid
DSOPlan
- registerLongTermCongestionPointsBeh : RegisterLongTermCongestionPoints
- queryActiveAggregatorsBeh : QueryActiveAggregators
- forecastNonAggregatorConnectionsBeh : ForecastNonAggregatorConnections
DSOValidate
- receiveDPrognosesBeh : ReceiveDPrognoses
- computeMissingPrognosesBeh : ComputeMissingPrognoses
- gridSafetyAnalysisBeh : GridSafetyAnalysis
- flexibilityTradingAGRDSOBeh : FlexibilityTradingAGRDSO

Figure 25: DSO FSM behaviours UML diagram

Figure 25 illustrates a general view of the DSO behaviours that are explained in the following subsections. The whole UML diagram for the DSO behaviours is included in the Annex of this deliverable.

Behaviour name	**DSOBehaviour**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the DSOPlan, DSOValidate and DSOOperate phases.
Inputs	--
Outputs	--/ ORANGE_REGIME (out of MAS2TERING scope)

4.1.5.1.1 DSO Plan

Behaviour name	**DSOPlan**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the working behaviours the DSO performs at Plan time, as introduced in Section 4.1.1. This behaviour combines

	RegisterLongTermCongestionPoints, QueryActiveAggregators and ForecastNonAggregatorConnections behaviours.
Inputs	--
Outputs	--

Behaviour name	**RegisterLongTermCongestionPoints**
Behaviour type	*OneShotBehaviour*
Description	This behaviour publishes the Congestion Points to the Directory Facilitator. This behaviour aims, in a longer term, at initiating a contact procedure with active AGRs that are related to a congestion point (cf. RegisterLongTermCongestionPoints in D5.3).
Inputs	Long-term congestion points
Outputs	--

Behaviour name	**QueryActiveAggregators**
Behaviour type	*OneShotBehaviour*
Description	This behaviour seeks the identity of AGRs that are connected to a long-term congestion point as well as the long-term congestion points they are related to. More details are provided in the RegisterLongTermCongestionPoint protocol in D5.3.
Inputs	--
Outputs	Active aggregators and the long-term congestion points they are related to.

Behaviour name	**ForecastNonAggregatorConnections**
Behaviour type	*OneShotBehaviour*
Description	This behaviour forecasts the energy consumption for the parts of the grid that are related to a congestion point and not monitored by an AGR.
Inputs	Grid model, active aggregators, their connections
Outputs	Forecasted consumption plan for connections that are not monitored by an AGR.

4.1.5.1.2 DSO Validate

Behaviour name	**DSOValidate**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the working behaviours performed by the DSOAgent at the Validate time, as introduced in Section 4.1.1. This behaviour combines the ReceiveDPrognosis, ComputeMissingPrognoses, GridSafetyAnalysis, FlexibilityTradingAGRDSO behaviours
Inputs	--
Outputs	--/NO_CONGESTION_EXPECTED, FAILED_SOLVING_CONGESTION

Behaviour name	ReceiveDPrognoses
Behaviour type	*OneShotBehaviour*
Description	This behaviour receives expected D-Prognoses from AGRs and record them. This behaviour stops either when all expected D-prognoses are received or when the day-ahead gate closure has passed.
Inputs	AGRs expected to send D-Prognoses. This set encompasses AGRs that are related to a congestion point and for which (1) no D-Prognosis is known or (2) updated D-Prognoses are expected after having sent a FlexOrder.
Outputs	Updated D-Prognoses/ALL_DPROGNOSES_RECEIVED, GATE_CLOSED

Behaviour name	ComputeMissingPrognoses
Behaviour type	*CustomBehaviour (to be implemented at a later time of the project)*
Description	This behaviour evaluates the consumption of the parts of the grid that are monitored by an AGR but for which no D-prognoses have been provided.
Inputs	Set of missing D-Prognoses, grid-model
Outputs	Updated D-Prognoses (all D-Prognoses should be completed).

Behaviour name	GridSafetyAnalysis
Behaviour type	*OneShotBehaviour*
Description	Analyses the grid configuration in order to detect any congestion at the level of transformers or lines.
Inputs	Grid model, D-Prognoses, consumption forecast of non-aggregator connections
Outputs	AGRs involved in congestion / CONGESTION_EXPECTED, NO_CONGESTION_EXPECTED

Behaviour name	FlexibilityTradingAGRDSO
Behaviour type	*CustomBehaviour (blank for now, to be implemented at a later time of the project)*
Description	This behaviour initiates trading operations with AGRs in order to avoid congestion.
Inputs	Active AGRs, grid-model, D-prognoses, long-term congestion points
Outputs	--/FAILED_SOLVING_CONGESTION, SENT_FLEX_ORDERS

4.1.5.1.3 DSO Operate

Behaviour name	DSOOperate
Behaviour type	*FSMBehaviour*

Description	This behaviour combines the working behaviours that the DSO has to perform at the Operate time, as introduced in the DSO FSM. This behaviour consists of the MonitorGrid behaviour.
Inputs	--
Outputs	--/CONGESTION_DETECTED_ON_OPEN_PTU, CONGESTION_DETECTED_ON_CLOSED_PTU

Behaviour name	**MonitorGrid**
Behaviour type	*OneShotBehaviour*
Description	This behaviour monitors the energy consumption in the grid and receives new D-Prognoses. It stops when a new D-prognosis causes a congestion.
Inputs	Grid model, D-prognoses, incoming D-Prognoses
Outputs	--/CONGESTION_EXPECTED_ON_OPEN_PTU, CONGESTION_EXPECTED_ON_CLOSED_PTU

4.1.5.2 AGR-level behaviours

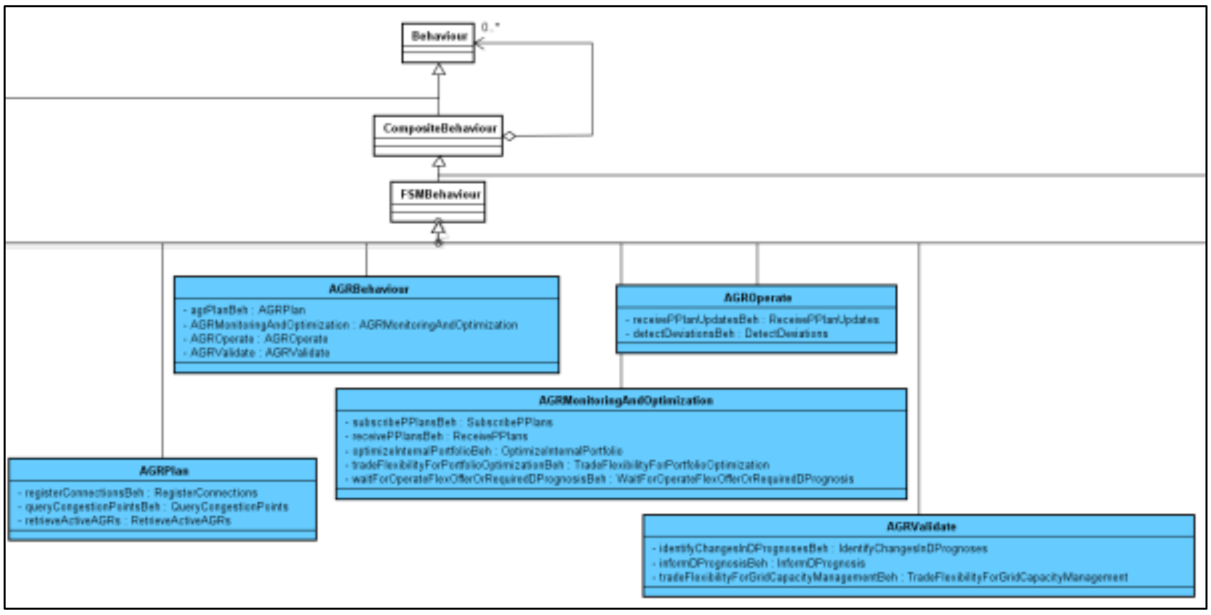

Behaviour
0..*
CompositeBehaviour
FSMBehaviour
AGRBehaviour
- agrPlanBeh : AGRPlan
- AGRMonitoringAndOptimization : AGRMonitoringAndOptimization
- AGROperate : AGROperate
- AGRValidate : AGRValidate
AGROperate
- receivePPlanUpdatesBeh : ReceivePPlanUpdates
- detectDeviationsBeh : DetectDeviations
AGRMonitoringAndOptimization
- subscribePPlansBeh : SubscribePPlans
- receivePPlansBeh : ReceivePPlans
- optimizeInternalPortfolioBeh : OptimizeInternalPortfolio
- tradeFlexibilityForPortfolioOptimizationBeh : TradeFlexibilityForPortfolioOptimization
- waitForOperateFlexOfferOrRequiredDPrognosisBeh : WaitForOperateFlexOfferOrRequiredDPrognosis
AGRPlan
- registerConnectionsBeh : RegisterConnections
- queryCongestionPointsBeh : QueryCongestionPoints
- retrieveActiveAGRs : RetrieveActiveAGRs
AGRValidate
- identifyChangesInDPrognosesBeh : IdentifyChangesInDPrognoses
- informDPrognosisBeh : InformDPrognosis
- tradeFlexibilityForGridCapacityManagementBeh : TradeFlexibilityForGridCapacityManagement

Figure 26 AGR FSM behaviours UML diagram

Figure 26 illustrates a general view of the AGR behaviours that are explained in the following subsections. The whole UML diagram for the AGR behaviours is included in the Annex of this deliverable.

4.1.5.2.1 AGR Plan

Behaviour name	AGRPlan
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the RegisterConnections, QueryCongestionPoints and RetrieveActiveAGRs and AGRMonitoringAndOptimization according to Section 4.1.2..
Inputs	--
Outputs	--/ out of MAS2TERING

Behaviour name	RegisterConnections
Behaviour type	*OneShotBehaviour*
Description	This behaviour sends to the DF the connections (the parts of the grid) in which the agent has active prosumers. See protocol RegisterConnections in D5.3.
Inputs	Connections, grid-model
Outputs	--

Behaviour name	QueryCongestionPoints
Behaviour type	*OneShotBehaviour*
Description	Requests from the DF the set of congestion points and related DSO to which the agent is related to. See QueryCongestionPoints in D5.3.
Inputs	--
Outputs	Congestion-points related to the agent

Behaviour name	RetrieveActiveAGRs
Behaviour type	*OneShotBehaviour*
Description	Requests the set of active AGRs from the DF.
Inputs	--
Outputs	Set of active AGRs

4.1.5.2.1.1. AGR Monitoring and Optimization

Behaviour name	AGRMonitoringAndOptimization
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the SubscribePPlans, ReceivePPlans, OptimizeInternalPortfolio, TradeFlexibilityForPortfolioOptimization, WaitForOperateFlexOfferOrDPrognosis, as depicted in the FSMs illustrated in Section 4.1.2.
Inputs	--
Outputs	OPERATE_TIME, VALIDATE_D_PROGNOSIS

Behaviour name	**SubscribePPlans**
Behaviour type	*OneShotBehaviour*
Description	The AGR agent requests PPlans from the set of CEMS agents monitored by the AGR. See the SubscribePPlans protocol in D5.3.
Inputs	Monitored CEMSs
Outputs	--

Behaviour name	**ReceivePPlans**
Behaviour type	*OneShotBehaviour*
Description	Wait and receives PPlans until having received all CEMS that are expected to send PPlans have done so. See SubscribePPlans protocol in D5.3.
Inputs	CEMSs expected to submit PPlans, new PPlan updates
Outputs	Updated PPlans

Behaviour name	**OptimizeInternalPortfolio**
Behaviour type	*CustomBehaviour*
Description	This behaviour internally optimizes the portfolio of CEMS agents (e.g. opening the local flexibility market, providing FlexOffers to CEMS). See protocol OptimizeInternalPortfolio in D5.3 for more.
Inputs	To be defined in D3.2/D3.3
Outputs	To be defined in D3.2/D3.3, updated Aplan

Behaviour name	**TradeFlexibilityForPortfolioOptimization**
Behaviour type	*CustomBehaviour*
Description	This behaviour performs AGR-to-AGR interactions in order to optimize costs by selling flexibility
Inputs	To be defined in D3.2/D3.3
Outputs	To be defined in D3.2/D3.3 / NEW_FLEX_ORDER (arising from another AGR), NO_OFFER

Behaviour name	**WaitForOperateFlexOfferOrRequiredDPrognosis**
Behaviour type	*OneShotBehaviour*
Description	This behaviour waits for either incoming messages from an external actor (a flex offer from another AGR or a DPrognosis from a DSO) or for operate time. Note: counter-intuitively but according to the USEF framework, AGRs do not react to PPlans at this time. Reacting to PPlan updates is done only after the operate phase.
Inputs	--
Outputs	NEW_FLEX_OFFER, OPERATE_TIME, VALIDATE_D_PROGNOSIS

4.1.5.2.2 AGR Validate

Behaviour name	AGRValidate
Behaviour type	*FSMBehaviour*
Description	Combines the behaviours to be performed during the validation phase for the AGR. These behaviours are: IdentifyChangesInDPrognoses, InformDPrognoses and TradeFlexibilityForGridCapacityManagement.
Inputs	--
Outputs	OPERATE_TIME, NEW_FLEX_ORDER

Behaviour name	IdentifyChangesInDPrognoses
Behaviour type	*OneShotBehaviour*
Description	This behaviour compares the former D-Prognoses with the new one and evaluates the presence of D-prognosis modifications (i.e. whether the consumption related to a congestion point has changed)
Inputs	Aplan, Congestion points
Outputs	Updated D-prognoses

Behaviour name	InformDPrognosis
Behaviour type	*OneShotBehaviour*
Description	This behaviour creates a D-Prognosis per congestion point based on the current Aplan and sends it to the DSO.
Inputs	Congestion points, D-prognoses
Outputs	--

Behaviour name	TradeFlexibilityForGridCapacityManagement
Behaviour type	*CustomBehaviour*
Description	This behaviour handles the flexibility trading between the AGR and the DSO in order to avoid congestion points.
Inputs	To be defined in D3.2 and D3.3
Outputs	To be defined in D3.2 and D3.3. OPERATE_TIME or NEW_FLEX_ORDER

4.1.5.2.3 AGR Operate

Behaviour name	AGROperate
Behaviour type	*FSMBehaviour*
Description	Combines the behaviours to be performed during the operate phase for the AGR. These behaviours are: ReceivePPlanUpdates and DetectDeviations as detailed in the FSM figure in Section 4.1.2.

Inputs	--
Outputs	MONITORED_CHANGES_ON_OPEN_PTU, MONITORED_CHANGES_ON_CLOSED_PTU (NO_DEVIATION : stay in this phase)

Behaviour name	**ReceivePPlanUpdates**
Behaviour type	*OneShotBehaviour*
Description	This behaviour waits for new PPlan updates to be received
Inputs	--
Outputs	New PPlan Updates

Behaviour name	**DetectDeviations**
Behaviour type	*OneShotBehaviour*
Description	Checks whether PPlan updates caused deviations in APlan and D-prognoses
Inputs	New PPlan Updates, PPlan, DPrognoses
Outputs	NO_DEVIATION, MONITORED_CHANGES_ON_OPEN_PTU, MONITORED_CHANGES_ON_CLOSED_PTU

Behaviour name	**AGRBehaviour**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the AGRPlan, AGRValidate and AGROperate phases according to Section 4.1.2.
Inputs	--
Outputs	--/ (out of MAS2TERING)

4.1.5.3 CEMS-level behaviours

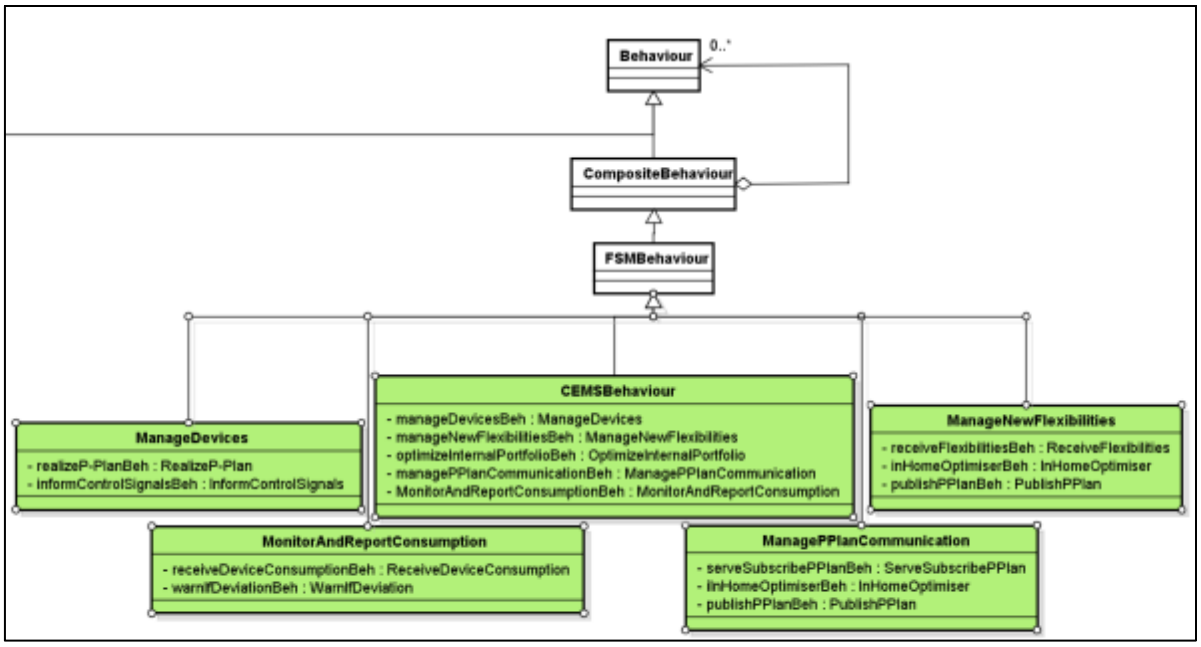

Behaviour
0..*
CompositeBehaviour
FSMBehaviour
CEMSBehaviour
- manageDevicesBeh : ManageDevices
- manageNewFlexibilitiesBeh : ManageNewFlexibilities
- optimizeInternalPortfolioBeh : OptimizeInternalPortfolio
- managePPlanCommunicationBeh : ManagePPlanCommunication
- MonitorAndReportConsumptionBeh : MonitorAndReportConsumption
ManageDevices
- realizeP-PlanBeh : RealizeP-Plan
- informControlSignalsBeh : InformControlSignals
ManageNewFlexibilities
- receiveFlexibilitiesBeh : ReceiveFlexibilities
- inHomeOptimiserBeh : InHomeOptimiser
- publishPPlanBeh : PublishPPlan
MonitorAndReportConsumption
- receiveDeviceConsumptionBeh : ReceiveDeviceConsumption
- warnIfDeviationBeh : WarnIfDeviation
ManagePPlanCommunication
- serveSubscribePPlanBeh : ServeSubscribePPlan
- iInHomeOptimiserBeh : InHomeOptimiser
- publishPPlanBeh : PublishPPlan

Figure 27: CEMS FSM behaviours UML diagram

Figure 27 illustrates a general view of the CEMS behaviours that are explained in the following subsections. The whole UML diagram for the CEMS behaviours is included in the Annex of this deliverable

Behaviour name	**CEMS Behaviour**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours of the CEMS agent: ManagePPlanCommunication, ManageNewFlexibilities, OptimizeInternalPortfolio, ManageDevices and MonitorAndReportConsumption, as described in Figure []
Inputs	--
Outputs	--

Behaviour name	**ManagePPlanCommunication**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours of the CEMS agent related to the management of requests for communicating PPlans. Combined behaviours are SeverSubscribePPlans, InHomeOptimizer and PublishPPlan, as described in the FSM figures in Section 4.1.3
Inputs	--
Outputs	--

Behaviour name	**ServeSubscribePPlan**

Behaviour type	*CustomBehaviour*
Description	This behaviour handles “SubscribePPlan” requests: the agent waits for new “SubscribePPlan” messages, decides whether to comply with this request and sends its acceptance or rejection back to the initiator. This behaviour implements the receiving side for the first exchange of the SubscribePPlan protocol (D5.3).
Inputs	Means for checking whether the sender is legit or not (to be defined in a later deliverable)
Outputs	Updated set of agents to keep informed about changes in PPlans

Behaviour name	**InHomeOptimiser**
Behaviour type	*CustomBehaviour*
Description	This behaviour builds a PPlan that makes uses of flexibility and energy trade offers in order to minimize energy-related costs while keeping sufficient satisfaction from prosumers.
Inputs	Flexibilities provided by devices. Additional optimization-related information to be defined in D3.2/D3.3.
Outputs	Updated PPlan

Behaviour name	**PublishPPlan**
Behaviour type	*OneShotBehaviour*
Description	This behaviour sends PPlans to agents to be kept informed about PPlan changes
Inputs	PPlan, agents to be kept informed about PPlan changes
Outputs	--

Behaviour name	**ManageNewFlexibilities**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours of the CEMS agent related to the management of the evolution of flexibilities from devices. Combined behaviours are SubscribeFlexibilities, ReceiveFlexibility, InHomeOptimizer and PublishPPlan, as described the FSM figures in Section 4.1.3
Inputs	--
Outputs	--

Behaviour name	**SubscribeFlexibilities**
Behaviour type	*ProposeInitiator*
Description	Register to devices agents in order to be informed when the flexibility of a device changes. See the SubscribeFlexibilities protocol in D5.3.
Inputs	Device Agents to be registered to.

Outputs	--

Behaviour name	**ReceiveFlexibilities**
Behaviour type	*OneShotBehaviour*
Description	This behaviour handles messages from Device Agents indicating a change of available flexibilities.
Inputs	--
Outputs	Updated flexibilities

Behaviour name	**OptimizeInternalPortfolio (Receiver)**
Behaviour type	*CustomBehaviour*
Description	Handles the negotiation in the flexibility market, from the CEMS side. This behaviour will be defined in D3.2 and D3.3.
Inputs	To be defined, FlexRequests, Flexibilities
Outputs	Updated FlexOrders

Behaviour name	**ManageDevices**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the plans related to the management of devices. This behaviour combines RealizePPlan and InformControlSignal.
Inputs	--
Outputs	--

Behaviour name	**RealizeP-Plan**
Behaviour type	*OneShotBehaviour*
Description	This behaviour realises its P-plan for a PTU by sending control signals to Device Agents.
Inputs	P-Plan
Outputs	--

Behaviour name	**InformControlSignals**
Behaviour type	*AchieveREInitiator*
Description	This behaviour sends control signals (indicating the activity to be performed by a device) to the corresponding in-home controllable devices.
Inputs	Control signals
Outputs	--

Behaviour name	**MonitorAndReportConsumption**
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours related to the monitoring of the

	consumption. This behaviour combines ReceiveDeviceConsumption and WarnIfDeviation.
Inputs	--
Outputs	--

Behaviour name	**ReceiveDeviceConsumption**
Behaviour type	*OneShotBehaviour*
Description	This behaviour handles the current and expected consumption messages from device agents
Inputs	--
Outputs	Updated consumption profile

Behaviour name	**WarnIfDeviation**
Behaviour type	*OneShotBehaviour*
Description	This behaviour warns the AGR agent in case of deviation from the proposed PPlan.
Inputs	PPlan, current consumption per device
Outputs	--

4.1.5.4 Device-level behaviour

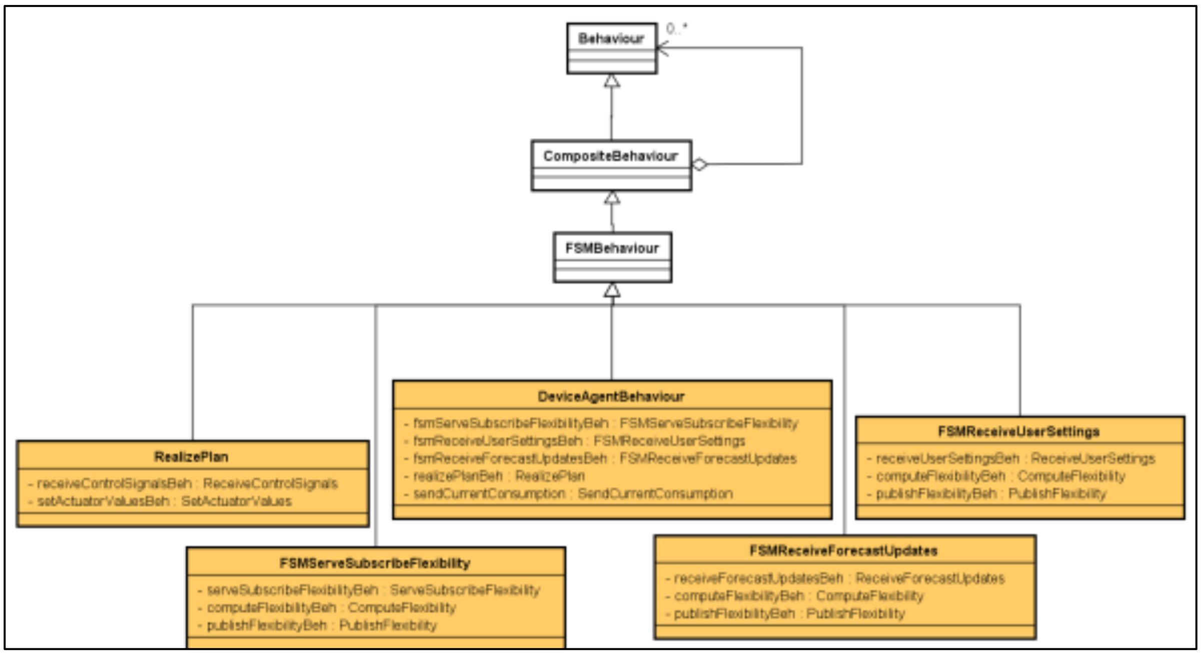

Behaviour
0..*
CompositeBehaviour
FSMBehaviour
DeviceAgentBehaviour
- fsmServeSubscribeFlexibilityBeh : FSMServeSubscribeFlexibility
- fsmReceiveUserSettingsBeh : FSMReceiveUserSettings
- fsmReceiveForecastUpdatesBeh : FSMReceiveForecastUpdates
- realizePlanBeh : RealizePlan
- sendCurrentConsumption : SendCurrentConsumption
RealizePlan
- receiveControlSignalsBeh : ReceiveControlSignals
- setActuatorValuesBeh : SetActuatorValues
FSMReceiveUserSettings
- receiveUserSettingsBeh : ReceiveUserSettings
- computeFlexibilityBeh : ComputeFlexibility
- publishFlexibilityBeh : PublishFlexibility
FSMServeSubscribeFlexibility
- serveSubscribeFlexibilityBeh : ServeSubscribeFlexibility
- computeFlexibilityBeh : ComputeFlexibility
- publishFlexibilityBeh : PublishFlexibility
FSMReceiveForecastUpdates
- receiveForecastUpdatesBeh : ReceiveForecastUpdates
- computeFlexibilityBeh : ComputeFlexibility
- publishFlexibilityBeh : PublishFlexibility

Figure 28: Device agent FSM behaviours UML diagram

Figure 28 illustrates a general view of the CEMS behaviours that are explained in the following subsections. The whole UML diagram for the CEMS behaviours is included in the Annex of this deliverable

Behaviour name	DeviceAgentBehaviour
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours related to the monitoring of the consumption. This behaviour combines FSMServeSubscribeFlexibility, FSMReceiveUserSettings, FSMReceiveUpdateForecast, RealizePlan and SendCurrentConsumption as detailed in Figure [].
Inputs	--
Outputs	--

Behaviour name	FSMServeSubscribeFlexibility
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours related to consumption monitoring. This behaviour combines ServeSubscribeFlexibility, ComputeFlexibility and PublishFlexibility.
Inputs	--
Outputs	--

Behaviour name	ServeSubscribeFlexibility
Behaviour type	*OneShotBehaviour*
Description	This behaviour checks external requests for being informed about flexibilities. If this request is granted, the requester is added to the list of agents to inform when flexibilities are updated.
Inputs	--
Outputs	Updated list of agents to be informed when flexibility changes.

Behaviour name	ComputeFlexibility
Behaviour type	*OneShotBehaviour*
Description	This behaviour evaluates the flexibilities and expected consumption of a device based on user settings and forecasts. To be completed in D3.2 and D3.3.
Inputs	Forecasts, user settings, consumption profile generator
Outputs	Updated flexibilities

Behaviour name	PublishFlexibility
Behaviour type	*OneShotBehaviour*
Description	This behaviour send flexibilities to agents to be kept informed about the device's flexibilities
Inputs	Flexibilities, agents to be informed about device's flexibilities
Outputs	--

Behaviour name	FSMReceiveUserSettings
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours related to the monitoring of the consumption. This behaviour combines ReceiveUserSettings, ComputeFlexibility and PublishFlexibility as detailed in Figure [].
Inputs	--
Outputs	--

Behaviour name	ReceiveUserSettings
Behaviour type	*CyclicBehaviour*
Description	Handle the insertion of user settings.
Inputs	--
Outputs	Updated user settings

Behaviour name	FSMReceiveForecastUpdates
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours related to the monitoring of the consumption. This behaviour combines ReceiveForecastUpdates, ComputeFlexibility and PublishFlexibility as detailed in Figure [].
Inputs	--
Outputs	--

Behaviour name	SubscribeForecast
Behaviour type	*CustomBehaviour*
Description	Register the CEMS agent in the forecast service in order to be notified when forecasts are updated. To be implemented in D3.2/D3.3
Inputs	Forecast services
Outputs	--

Behaviour name	ReceiveForecastUpdates
Behaviour type	*OneShotBehaviour*
Description	The device agent receives via web services the new forecasted values
Inputs	Forecasts services
Outputs	Updated forecasts

Behaviour name	RealizePlan
Behaviour type	*FSMBehaviour*
Description	This behaviour combines the behaviours related to the realization of the plan. This behaviour combines ReceiveControlSignals and

	SetActuatorValues as detailed in Figure [].
Inputs	--
Outputs	--

Behaviour name	**ReceiveControlSignals**
Behaviour type	*OneShotBehavior*
Description	Receive the control signals from the CEMS agent
Inputs	--
Outputs	Updated control signals

Behaviour name	**SetActuatorValues**
Behaviour type	*OneShotBehaviour*
Description	Set up the actuators of the controllable device with the values to be read from device configuration.
Inputs	Control signals
Outputs	--

Behaviour name	**SendCurrentConsumption**
Behaviour type	*OneShotBehaviour*
Description	Report the current energy consumption to agents to keep informed about the consumption of energy
Inputs	Monitored consumption, agents to keep informed
Outputs	--

5 Constraints and objectives

MAS2TERING takes a pragmatic, practical approach when choosing the agent model in order to maintain the internal agent architecture simple and the system scalable. Each agent is modelled using a constraint programming approach in which goals and preferences of the agent are modelled using a mathematical model based on variables and (hard and soft) constraints. More complex agent models such as Belief-Desired-Intention (BDI) models are not expected [6].

This module implements two main concepts:

1. Variables: A variable is a mathematical object whose value can vary across a set called the variable´s domain. Variables can be given a value as a result of an observation (e.g. the external temperature) or as a result of a decision taken by an agent (i.e. a decision variable). Each variable is associated to a domain and a domain can be discrete or continuous. Another related class is the *VariablesAssignment* class that allows assigning specific values (i.e. contained in their domain) to variables. The set of classes corresponding to these variable related concepts are illustrated in Figure 29.

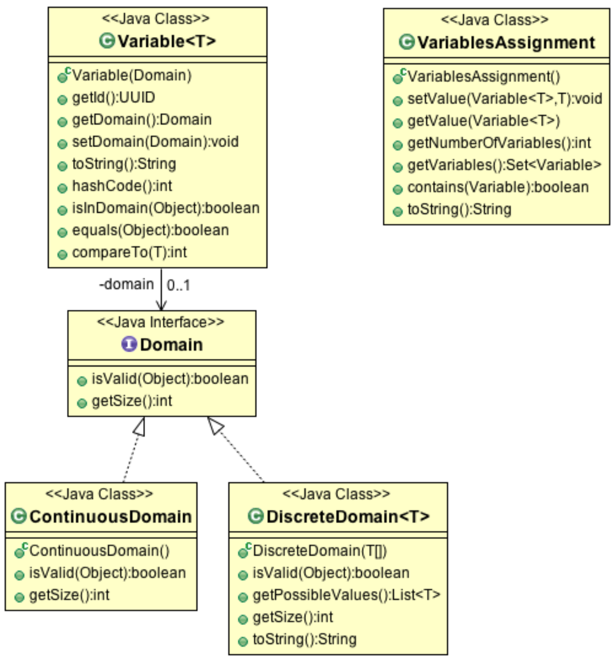
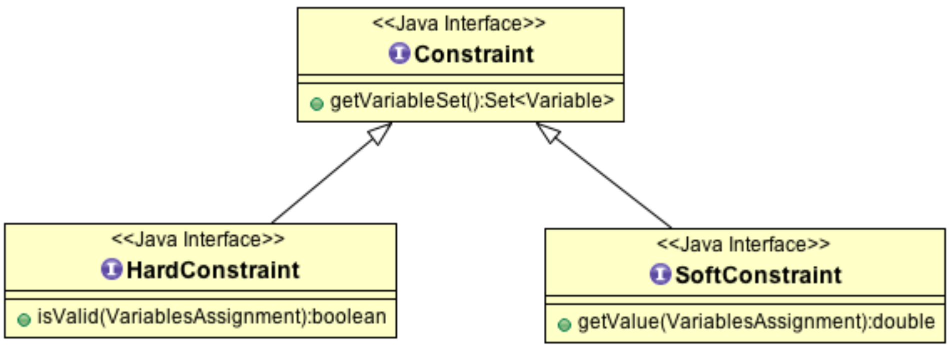


Figure 29 Classes in the variable package

2. Constraints: Each constraint is defined over a set of variables. As such all constraints need to implement the method *getVariableSet()* that returns the set of variables over which is defined the constraint. As shown in Figure 30, there two main types of constraints:

1. Soft: A soft constraint returns some cost for some assignment of variables. As such a soft constraint is required to implement the method *getValue* that given an assignment for variables returns the corresponding cost[4];
2. Hard: A hard constraint must not be violated otherwise the solution is not valid. As such a hard constraint is required to implement the method *isValid* that given a variable assignment returns a boolean indicating if the constraint is satisfied or not.

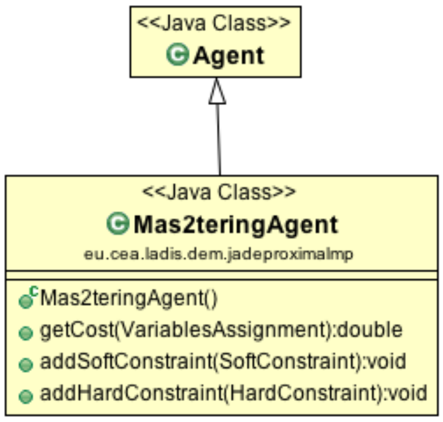


<<Java Interface>>
Constraint
getVariableSet():Set<Variable>
<<Java Interface>>
HardConstraint
isValid(VariablesAssignment):boolean
<<Java Interface>>
SoftConstraint
getValue(VariablesAssignment):double

Figure 30: Classes in the constraint package.

Classes in the package behaviour allow associating to each agent with a set of objectives and constraints (i.e. goals for socio-economic agents or operation constraints for physical resource agents).

Let A be the set of agents. Then, each agent $a_i \in A$ in MAS2TERING has a set of soft constraints, namely S_i, and a set of hard constraints, namely H_i. These constraints are added by the methods *addSoftConstraint* and *addHardConstraint.*



<<Java Class>>
Agent
<<Java Class>>
Mas2teringAgent
eu.cea.ladis.dem.jadeproximalmp
Mas2teringAgent()
getCost(VariablesAssignment):double
addSoftConstraint(SoftConstraint):void
addHardConstraint(HardConstraint):void

Figure 31: MAS2TERING agent class

[4] Notice that since the cost is not restricted to be positive, this method also allows returning some reward expressed as negative cost.

Then, the set of goals of an agent is modelled as a constraint optimisation problem in which each agent aims to optimise an objective function with respect to some variables (i.e. X_i) in the presence of constraint on those variables as follows:

$$\min \sum_{s \in S_i} s(X_i)$$
$$\text{subject to: } h(X_i),\ \forall h \in H_i$$

Note that since the objective function is actually the sum of the costs returned by individual soft constraints, it is meant to be minimised. Finally, each agent has also a *getCost()* function that given an assignment of values to variables (i.e. a *VariableAssignment*) returns the cost of this agent with respect to this state.

In the following, we define the constraints and objectives for each of the specific device agents that will be instantiated in the MAS2TERING use cases.

5.1 Generator

Next tables show the physical definition in terms of soft and hard constraints that the power schedule must meet:

Hard constraints:

$P^{min} \leq -p_{gen} \leq P^{max}$	The generator supplies power between a minimum and a maximum depending on the generator's specifications.
$R^{min} \leq -Dp_{gen} \leq R^{max}$	R is ramp rate limit; this constraint limits the change of power levels from one period to the next of a generator.

Soft constraints:

$\text{Cost}(p_{gen}) = \sum_{t=1}^{T} [c * -p_{gen}(t)]$ **$c = \beta x$** **$c = \alpha x^2 + \beta x$**	Cost of operating the generator at a given power level over a single time period. It can be: • linear • quadratic

where

p_{gen}	Generator power schedule
D	Difference power schedule between the current time step and the previous one.
$\alpha,\beta > 0$	Cost coefficients.

5.2 Curtailable Load

The soft and hard constraints for a curtailable load are listed below:

Hard constraints:

$p_{load}(\tau) \geq 0, \quad \forall \tau = 1, \ldots, T$	The curtailable load is restricted to consume

	(i.e. not to produce) energy.
$p_{load}(\tau) \leq d_{load}(\tau), \quad \forall \tau = 1, \ldots, T$	The amount of energy delivered is always less than the desired.

Soft constraints:

$\alpha \cdot (d_{load}(\tau) - p_{load}(\tau)), \quad \forall \tau = 1 \ldots T$	A linear penalty is applied on the difference between the desired and the delivered power (i.e the energy shortfall.)

Where

p_{load}	Deferrable load power schedule
d_{load}	Desired load power schedule
α	Linear penalty parameter. Greater than zero.

5.3 Deferrable load

The deferrable load does not have soft constraints. However, the energy consumption in each period of time is constrained by E. In some cases, the load can only be turned on or off in each period of time. Therefore, two hard constraints can be extracted:

$\sum_{\tau=A}^{D} p_{load}(\tau) = E$	After the starting time slot, the device needs to charge the needed amount of energy before the final time slot.
$0 \leq P_{load} \leq L^{max}$	The deferrable load is restricted to consume (i.e. not to produce) energy (between 0 and L^{max}).

Where:

p_{load}	**Deferrable load power schedule**
A	Minimum time slot
D	Maximum time slot
E	Total energy required for the time interval A … D
L^{max}	Maximum energy consumption of the device for each time slot

5.4 Storage

Similar to the deferrable load, a storage device is only defined by hard constraints:

$-D^{max} \leq p_{bat}(\tau) \leq C^{max}, \quad \tau = 1, \ldots, T$	The power transmitted must be between the discharging and charging rates limits.
$0 \leq q(\tau) \leq Q^{max}, \tau = 1, \ldots T$ **where:** $q(\tau) = q^{init} + \sum_{t=1}^{\tau} p_{bat}(t)$	The charge level must not exceed the battery capacity

Where:

p_{bat}	Battery power schedule
D^{max}	Maximum discharging rate
C^{max}	Maximum charging rate
Q^{max}	Battery capacity
q^{init}	Battery initial charge

5.5 Fixed load

A Fixed load has a unique hard constraint, which is as follows:

$p_{load}(\tau) = d_{load}(\tau), \forall\tau = 1, ..., T$	The desired consumption of a fixed load must be satisfied in each period.

Where:

Table 6 Fixed Load parameters

p_{load}	Fixed load power schedule
d_{load}	Desired load power schedule

5.6 External tie

The soft constraints of an external tie vary depending if the external tie allows pulling electricity from the utility, injecting electricity to the utility or both.

Case pulling from:	$\sum_{\tau=1}^{T} -P^{in}(\tau) \cdot p_{ex}(\tau)$
Case injecting to:	$\sum_{\tau=1}^{T} -P^{out}(\tau) \cdot p_{ex}(\tau)$
Case pulling from/injecting to:	$-c(\tau) \cdot p_{ex} + \gamma(\tau) \cdot \lvert p_{ex}\rvert \quad \forall\tau = 1 \ldots T$ where: $\gamma(\tau) = \frac{P^{in}(\tau) - P^{out}(\tau)}{2}$

In the same way, this device presents some hard constraints as well:

Case pulling from:	$-E^{max}(\tau) \leq p_{ex}(\tau) \leq 0, \quad \forall\tau = 1, ..., T$
Case injecting to:	$0 \leq p_{ex}(\tau) \leq E^{max}(\tau), \quad \forall\tau = 1, ..., T$
Case pulling from/injecting to:	$\lvert p_{ex}(\tau)\rvert \leq E^{max}(\tau), \quad \forall\tau = 1, ..., T$

where

Table 7 External Tie parameters

p_{ex}	External tie power schedule
P^{in}	Price per unit of energy pulled from the source (Default value)

p^{out}	Price per unit of energy injected to the source (Default value)
E^{max}	Maximum transaction of electricity (Default value: Infinity)

5.7 Transmission line

A transmission line works at the distribution level and may carry some energy losses that are considered as hard constraints:

$\frac{\lvert p_{L1}(\tau) - p_{L2}(\tau)\rvert}{2} \leq C^{max}, \quad \tau = 1, \ldots, T$	The power transmitted must respect the maximum capacity.
$p_{L1} + p_{L2} - \delta(p_{L1}, p_{L2}) = 0$ **where:** $\delta(p_{L1}, p_{L2}) : R^T \times R^T \rightarrow R^T_+$	The power that gets in the line must be the same that the power that gets out (power conservation) taking into account losses.

where:

p_{L1}	First power schedule of the line
p_{L2}	Second power schedule of the line
C^{max}	Maximum capacity

6 Behaviours and agents involved in the use cases

We include a short description of the project's use cases from an agent perspective by specifying which agents and which behaviours are involved in each use case. For this purpose, we provide a traceability matrix, in which a cell marked with a "X" denotes that the behaviour/agent is involved in a given use case

In UC1, the distribution level of a Smart Grid will include various types of active dynamic devices, such as distributed generators based on solar and wind, batteries, deferrable loads, curtailable loads, and electric vehicles, whose control and scheduling amount to a very complex management problem.

UC1 concerns the Prosumer in-home optimisation, the interoperability and the connection to handle requests/connections to the flexibility market via the aggregator. Agents involved in this UC are:

1. CEMS
2. Device that represents the devices installed at home level. Within these devices many subtypes are involved.

The CEMS agent is owner of all the in-home devices and the Distributed Energy Resources (DERs) inside its home. Each of those physical devices has a cost function and hence the cost function of the prosumer is the sum of costs functions of its devices. When deploying the system many Device agents but just one CEMS may be instantiated.

UC2 deals instead with local management at the district level. Since the local community is considered as a collection of consumption/generation nodes that are managed by a single entity, the aggregator. Therefore a new agent takes place in the system, the Aggregator. It expected to communicate with the houses (CEMS agents) that belong to this district. Several Device and CEMS agents can be involved in this use case which would be associated to one Aggregator agent.

UC3 is considered as an extension of UC2 because it takes the entire low voltage power grid as the union of many local communities in a given area: the concept of DSO emerges. The use case may involve one or more DSOs which communicate with the aggregators in order to negotiate the power plans and inform the congestion points of the power grid. Furthermore, each aggregator exchange messages with CEMS agents which are also linked to Device agents.

Agents	UC1	UC2	UC3
DSO			X
Aggregator	X	X	X
CEMS	X	X	X
Device (subtypes)	DeferrableLoad, CurtailableLoad, FixedLoad, Battery	DeferrableLoad, CurtailableLoad, FixedLoad, Battery	DeferrableLoad, CurtailableLoad, FixedLoad, Battery, TransmissionLine

Table 8 Agents used in use cases

Table 8 and Table 9 show all the agent types and behaviours implemented in the platform. As we can see, they can be assigned to one or more use cases.

Behaviours	UC1	UC2	UC3
PublishP-Plan	X	X	X
InHomeOptimiser	X	X	X
ReceiveFlexibility	X	X	X
HandleFlexibilityRequest	X	X	X
ServeSubscribePPlan		X	X
SubscribeFlexibilities	X	X	X
RealisePlan	X	X	X
InformControlSignals	X	X	X
SubscribeForecast	X	X	X
ServeSubscribeFlexibility	X	X	X
ReceiveUserSettings	X	X	X
ReceiveUpdatedForecast	X	X	X
ComputeFlexibility	X	X	X
PublishFlexibility	X	X	X
ReceiveControlSignals	X	X	X
SetActuatorValues	X	X	X
RegisterConnections		X	X
QueryCongestionPoints		X	X
SubscribePPlans		X	X
ReceivePPlan		X	X
InformDPrognosis			X
IdentifyChangesInAPlan		X	X
IdentifyChangesInDPrognoses			X
OptimiseInternalPortfolio		X	X
TradeFlexibilityForPortfolioOptimisation		X	X
FlexibilityTradingAGRDSO		X	X
ForecastNonAggregatorConnections			X
ComputeMissingPrognoses			X
GridSafetyAnalysis			X
RegisterLongTermCongestionPoints			X
ReceiveDPrognoses			X
QueryActiveAggregators			X

Table 9 Behaviours used in use cases

7 Conclusions and next steps

This deliverable provides a first version of the implementation of the multi-agent and holonic platform for MAS2TERING. To extract the requirements and for the analysis phase, our deliverable was based on the USEF framework described in deliverable D1.6, with which MAS2TERING aligns, and the use cases defined in deliverable D6.1, which will be used to validate our proposal.

GAIA methodology, which has been adopted in deliverable D2.2 as agent development methodology, has been also used in this deliverable. Furthermore, the MAS platform, defined in deliverable D2.2 has been also used as the base for the implementation of the identified agents, their behaviours, and their constraints.

As for the Smart grid model, this deliverable presents a MAS2TERING common data model in order to allow the common expression of information exchange between agents. It also details the alignment of this common data model with the three identified standards for use, namely the CIM standard for modelling the electrical domain, the OpenADR standard for modelling demand response within the Smart Grid, and the Energy@Home standard for domestic conceptual modelling and their relevance for the different use cases.

As for the agents, this deliverable details the implementation of the four types of agents that have been defined inside the Agents Model component, and that will be instantiated in the project use cases that are specified in the deliverable D6.1. These agents are the Device agent (with its seven subtypes depending on the type of flexibility provided), the CEMS agent, the AGR agent, and the DSO agent. Each of the subtypes of the Device agent has its constraints and objectives, which have been specified and implemented in this deliverable. The communication protocols between the agents are defined in deliverable D5.3, and will be implemented inside the Communication and protocol component (as part of D5.4), whereas the security aspects studied in deliverable D4.2 will be implemented inside the Security component of the MAS platform.

This deliverable is accompanied by the first version software implementation of the specified agents and their behaviours in this document. The following deliverables in this work package shall focus on designing and implementing the Forecasting algorithms and the optimisation protocols in order to be integrated into the agents, completing the MAS implementation phase.

References

[1] E. P. N. I. S. Pavlos Moraitis, “Engineering JADE Agents with the Gaia Methodology,” *Agent Technologies, Infrastructures, Tools, and Applications for E-Services ,* 2002.

[2] M. Wooldridge, D. Kinny and N. R. Jennings, “"The Gaia methodology for agent-oriented analysis and design,” *Autonomous Agents and multi-agent systems,* vol. 3.3, pp. 285-312, 2000.

[3] F. Bellifemine, G. Caire and D. Greenwood, Developing multi-agent systems with JADE, Wiley, 2007.

[4] N. I. S. Pavlos Moraitis, “The Gaia2Jade process for multi-agent systems development,” *Applied Artificial Intelligence ,* vol. 20, no. 2-4, pp. 251-273, 2006.

[5] USEF, “USEF: The framework explained,” USEF, 2015.

[6] A. S. Georgeff, “DI Agents: from theory to practice,” in *B. First International Conference on Multiagent Systems* , San Francisco, California, USA, 1995.

[7] D. -. D. o. E. [online], “Title XIII Smart Grid,” [Online]. Available: http://www.oe.energy.gov/DocumentsandMedia/EISA_Title_XIII_Smart_Grid.pdf.

[8] M. McGranaghan and B. Deaver, “Sensors and Monitoring Challanges in the Smart Grid,” in *Future of Instrumentation International Workshop (FIIW)*, 2012.

[9] E. C. J. L. S. B. M Kraning, “Dynamic Network Energy Management via Proximal Message Passing,” *Foundations and Trends in Optimization,* vol. 1, no. 2, pp. 73-126, 2014.

[10] P. S. a. S. Thiébaux, “Distributed Multi-Period Optimal Power Flow for Demand Response in Microgrids,” in *e-Energy*, Canberra, 2015.

[11] R. Segovia and M. Sanchez, “Set of common functional requirements of the Smart Meter,” European Commission, 2011.

[12] EUROPEAN COMMISSION, “Report from the Commission: Benchmarking smart metering deployment in the EU-27 with a focus on electricity,” Brussels, 2014.

[13] M. Pau, A. Pegoraro and S. Sulis, “Branch current state estimator for distribution system based on synchronised measurements,” *IEEE International Workshop on Appled Measurements for Power Systems (AMPS),* pp. 53-58, 2012.

[14] M. Pau, P. Pegoraro and S. Sulis, “Efficient branch Current based Distribution System State Estimator in- luding Synchronised Measurements,” *IEEE Transactions on Instrumentation and Measurements,* 2013.

[15] M. Pau, P. Pegoraro and S. Sulis, “WLS Distribution System State Estimator Based on Voltages or Branch Currents: Accuracy and Performance Comparison,” *IEEE Instrumentation and Measurement Technology Conference I2MTC ,* pp. 493-498, 2013.

[16] J. Liu, F. Ponci, A. Monti, C. Muscas and P. Pegoraro, “Trade-Offs in PMU Deployment for State Estimation in Active Distribution Grids,” *IEEE Transactions on Smart Grids,* vol. 3, no. 2, pp. 915-924, 2012.

[17] J. Liu, F. Ponci, A. Monti, C. Muscas, P. Pegoraro and S. Sulis, “Optimal Placement for Robust Distributed Measurement Systems in Active Distribution Grids,” *IEEE Instrumentation and Measurement Technology Conference I2MTC 2013 Minneapolis,* pp. 206-211, 2013.

[18] K. D. McBee, “Benefits of Utilizing a Smart Grid Monitoring System to Improve Feeder Voltage,” in *North America Power Symposium (NAPS)*, 2009.

[19] L. Kumar, “A literature review on Distribution System State Estimation,” *SMART GRID Technologies,* 2015.

[20] a. L. C. L. Oliva, “REST Web Services for Collaborative Work Environments. In: Frontiers in Artificial Intelligence and Applications,” in *Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence.*, Amsterdam, 2009.

[21] E. W. a. R. A. C. Pautasso, “REST: Advanced Research Topics and Practical Applications,” in *47-48*, NewYork, 2014.

[22] SOAPUI, “Best Practices: Understanding REST Headers and Parameters,” [Online]. Available: https://www.soapui.org/testing-dojo/best-practices/understanding-rest-headers-and-parameters.html. [Accessed 23 Feb 2016].

Annex A

A.1 Finite State Machines

This section introduces further information for understanding the meaning of FSMs described in this document. FSMs model the possible evolutions of the states of a system. Basically, a system is assumed to be in a given state. The system can change from one state to another due to the occurrence of certain events (or actions), depending on the current state.
Slightly more formally, a FSM is composed of a set of states (represented by nodes) and transitions (represented by arrows). One of these states is referred to as the initial state (represented by a simple black dot) and another state is referred to as the final state. Each transition is related to an event (represented by labels near the arrows). When there is only one possible transition, we hide the event for sake of simplicity.

The system is run as follows: system "starts" at the initial state. When an event occurs, the system evolves from the current state to another, following the transition that possesses the adequate event name. When a final state is reached, the system is stopped. Note that final states are not necessarily reachable nor reached: the system can run forever.

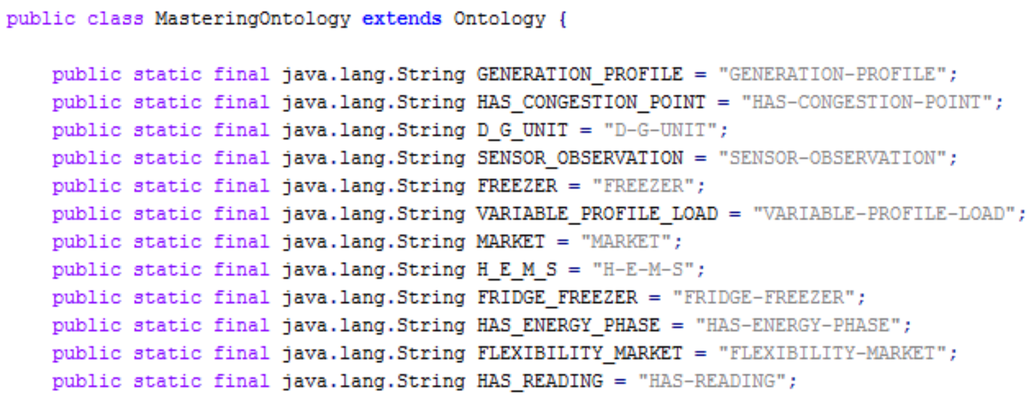
FSMs are used in MAS2TERING for describing agent behaviours as such. Each state is a behaviour. In other words, when the system is in a given state, then the agent is performing the related behaviour (in practice, this behaviour consists of a procedure that is executed until a function indicates that this behaviour is terminated). An event is raised when the behaviour is completed. The nature of the event is determined by how this behaviour was performed (e.g. if congestion was discovered or not). In overall, the agent is performing a given behaviour until this behaviour is completed. Then the agent triggers a transition and starts performing another behaviour, until a final state is reached.

In addition to the basic FSM constructs, we rely on two more elaborated FSM constructs: forks (or parallel executions) and hierarchical FSMs. Forks are a special form of transition. When firing such a transition, the system goes into not one but a set of states. Basically, the system "runs" multiple FSMs in parallel. In our case, parallelism consists of a round-robin: each FSM is run in turn, one after the other
Hierarchical FSMs are a specific way for representing states. Basically, this technique consists in representing the internal details of what happens in a state as a FSM. In other words, a state/behaviour of a FSM is represented in using another FSM. The event arising from a FSM-based state matches the last event raised within this state. In other words, a state is, instead of a piece of code, another FSM. When firing such a state, a user can "zoom in" this state and find another FSM describing how this state is being executed. This representation is very common for modelling MAS agents. Further details about the FSM formalism and their use for building MAS agents in [3].

Annex B

B.1 Conversion process from ontologies to JADE

Through this conversion process, the ontology's axioms were formalised using JADE constructs, as shown in Figure 32, Figure 33, and Figure 34, which show the formalisation of the vocabulary, a class (including inheritance and data properties) and object properties (referred to as predicates in JADE documentation) respectively.



```
public class MasteringOntology extends Ontology {

    public static final java.lang.String GENERATION_PROFILE = "GENERATION-PROFILE";
    public static final java.lang.String HAS_CONGESTION_POINT = "HAS-CONGESTION-POINT";
    public static final java.lang.String D_G_UNIT = "D-G-UNIT";
    public static final java.lang.String SENSOR_OBSERVATION = "SENSOR-OBSERVATION";
    public static final java.lang.String FREEZER = "FREEZER";
    public static final java.lang.String VARIABLE_PROFILE_LOAD = "VARIABLE-PROFILE-LOAD";
    public static final java.lang.String MARKET = "MARKET";
    public static final java.lang.String H_E_M_S = "H-E-M-S";
    public static final java.lang.String FRIDGE_FREEZER = "FRIDGE-FREEZER";
    public static final java.lang.String HAS_ENERGY_PHASE = "HAS-ENERGY-PHASE";
    public static final java.lang.String FLEXIBILITY_MARKET = "FLEXIBILITY-MARKET";
    public static final java.lang.String HAS_READING = "HAS-READING";
```

Figure 32 Except of JADE ACL vocabulary definition

```java
public class DeferrableLoad extends DomesticLoad {

    public void setHasTotalEnergyDemand(java.lang.Float HasTotalEnergyDemand) {
        this.HasTotalEnergyDemand = HasTotalEnergyDemand ;
    }
    public java.lang.Float getHasTotalEnergyDemand(){
        return this.HasTotalEnergyDemand;
    }
    private java.lang.Float _hasTotalEnergyDemand;
    public void setHasMinDuration(java.lang.Float HasMinDuration) {
        this.HasMinDuration = HasMinDuration ;
    }
    public java.lang.Float getHasMinDuration(){
        return this.HasMinDuration;
    }
    private java.lang.Float _hasMinDuration;
    public void setHasPlannedDeferment(java.lang.Float HasPlannedDeferment) {
        this.HasPlannedDeferment = HasPlannedDeferment ;
    }
    public java.lang.Float getHasPlannedDeferment(){
        return this.HasPlannedDeferment;
    }
    private java.lang.Float _hasPlannedDeferment;
    public void setHasExecutionDeadline(java.lang.String HasExecutionDeadline) {
        this.HasExecutionDeadline = HasExecutionDeadline ;
    }
    public java.lang.String getHasExecutionDeadline(){
        return this.HasExecutionDeadline;
    }
    private java.lang.String _hasExecutionDeadline;
}
```

Figure 33 Example of JADE ACL ontology class definition

```java
public class HasEnergyPhase extends Predicate {

    public void setEnergyPhase(EnergyPhase EnergyPhase) {
        this.EnergyPhase = EnergyPhase ;
    }
    public void setMode(Mode Mode) {
        this.Mode = Mode ;
    }
    public EnergyPhase getEnergyPhase(){
        return this.EnergyPhase;
    }
    public Mode getMode(){
        return this.Mode;
    }
    private EnergyPhase _energyPhase;
    private Mode _mode;
}
```

Figure 34 Example of JADE ACL ontology predicate definition