



**Intelligent Assistants for Flexibility Management  
(Grant Agreement No 957670)**

**D3.9 Final automated flexibility management module**

**Date: 2023-10-31**

**Version 1.0**

Published by the iFLEX Consortium

Dissemination Level: PU - Public



Co-funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation  
under Grant Agreement No 957670

## Document control page

**Document file:** D3\_9\_Final\_automated\_flexibility\_management\_module.docx  
**Document version:** 1.0  
**Document owner:** VTT

**Work package:** WP3 Artificial Intelligence for forecasting and automated flexibility management  
**Deliverable type:** DEM - Demonstrator, pilot, prototype

**Document status:**  Approved by the document owner for internal review  
 Approved for submission to the EC

## Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Jussi Kiljander (VTT)	2023-08-15	Initial ToC based on the D3.8
0.5	Jussi Kiljander (VTT), Janne Takalo-Mattila (VTT), Juho Kivelä (VTT)	2023-10-01	Updates to section 5.
0.9	Jussi Kiljander (VTT)	2023-10-25	Internal review ready version
1.0	Jussi Kiljander (VTT)		

## Internal review history:

Reviewed by	Date	Summary of comments
Athanasios Papakonstantinou (HERON)	2023-11-08	Accepted with minor changes.
Dušan Gabrijelčič	2023-11-07	Accepted with minor changes.

### Legal Notice

The information in this document is subject to change without notice.

The Members of the iFLEX Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the iFLEX Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Possible inaccuracies of information are under the responsibility of the project. This report reflects solely the views of its authors. The European Commission is not liable for any use that may be made of the information contained therein.

## Index:

<b>Abbreviations and acronyms</b> .....	<b>4</b>
<b>1 Executive summary</b> .....	<b>5</b>
<b>2 Introduction</b> .....	<b>6</b>
2.1 Purpose, context and scope .....	6
2.2 Main changes compared to D3.8 .....	6
2.3 Content and structure .....	6
<b>3 Overview</b> .....	<b>7</b>
3.1 Relation to use cases .....	7
3.2 Relation to the functional architecture of the iFLEX Framework .....	8
3.3 Third phase focus .....	9
<b>4 Methodology and approach</b> .....	<b>10</b>
4.1 Functional architecture .....	10
4.2 Monte-Carlo Three Search for optimal control with hybrid models .....	12
<b>5 Implementation</b> .....	<b>15</b>
5.1 Overview .....	15
5.1.1 Software architecture .....	15
5.1.2 Use cases .....	20
5.1.3 Interface provided by the Automated Flexibility Management module .....	22
5.2 Instantiations .....	26
5.2.1 Instantiation in the Finnish pilot .....	26
5.2.2 Instantiation in the Greek pilot .....	29
5.2.3 Instantiation in the Slovenian pilot .....	30
<b>6 Conclusions</b> .....	<b>33</b>
<b>List of figures and tables</b> .....	<b>34</b>
6.1 Figures .....	34
6.2 Tables .....	34
<b>7 References</b> .....	<b>35</b>
<b>8 Appendix: Jira requirements</b> .....	<b>36</b>
8.1.1 [IF-107] FN-AFM-05 Optimize flexibility locally (self-consumption, consumer load reduction) Created: 30/Jul/22 Updated: 31/Jul/22 .....	36
8.1.2 [IF-72] FN-AFM-04 Optimize flexibility based on prices (implicit demand response) Created: 15/Jun/21 Updated: 31/Jul/22 .....	37
8.1.3 [IF-71] FN-AFM-03 Activate offered flexibility Created: 15/Jun/21 Updated: 14/Feb/22 Resolved: 14/Feb/22 .....	38
8.1.4 [IF-70] FN-AFM-02 Flexibility potential Created: 15/Jun/21 Updated: 14/Feb/22 Resolved: 14/Feb/22 .....	39
8.1.5 [IF-69] FN-AFM-01 Provide baseline forecasts Created: 15/Jun/21 Updated: 14/Feb/22 Resolved: 14/Feb/22 .....	40

## Abbreviations and acronyms

AI	Artificial Intelligence
AFM	Automated Flexibility Manager
ANN	Artificial Neural Networks
API	Application Programming Interface
BEMS	Building Energy Management Systems
CEM	Customer Energy Manager
DR	Demand Response
DRL	Deep Reinforcement Learning
DRMS	Demand Response Management System
DSO	Distribution System Operator
ESCO	Energy Service Company
HEMS	Home Energy Management System
HVAC	Heating, Ventilation and Air Conditioning
ICT	Information and Communications Technology
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MCTS	Monte Carlo Tree Search
ML	Machine Learning
MPC	Model Predictive Control
MLP	Multilayer Perceptron
MQTT	Publish-subscribe network protocol
MTC	Machine Type Communication
MV	Medium Voltage
MVP	Minimum Viable Product
RES	Renewable Energy Sources
RL	Reinforcement Learning
UML	Unified Modelling Language
VPP	Virtual Power Plant
WP	Work Package

## 1 Executive summary

This deliverable presents the final iteration of the Automated Flexibility Management (AFM) module. The AFM module is a key component of the iFLEX Assistant. It is responsible for providing the *Aggregator and Market Interface* module and optimizing energy management within the consumer premises. The goal for this deliverable is to document the third specification and implementation of the AFM module, its interfaces with other modules, and insights into different optimal control approaches.

The approach and methodology for implementing the AFM module is based on applying Artificial Intelligence (AI) technologies together with optimal control methods. Instead of typical AI-based decision-making approaches that learn control policies directly the idea is to apply model-based planning and control where machine learning is applied together with physics-based modelling to learn accurate and robust models of the consumer (includes people and their premises). These models form a digital twin of the consumer that can be used to plan and optimize flexibility management.

The AFM control architecture is divided into two levels: Energy Planner and Controller(s). The Energy Planner is similar to the Customer Energy Manager (CEM) introduced in the EN 50491-12-x standard series. It is responsible for aggregating and optimizing the energy management at the consumer/building level without a need for flexible asset specific details. To realize this the Energy Planner interacts with Controllers that correspond to Resource Managers (RM) in the EN 50491-12-2 standard. Each Controller is responsible for a logical group of devices providing a logical functionality (e.g., heating system). In contrast to the Energy Planner that optimizes energy management at the whole building level, each Controller just follows the load profile assigned to it by the Energy Planner. Section 4 describes the approach in more detail with the Monte Carlo Three Search Algorithm that is used for searching optimal control policies with the models.

The AFM prototype for the third phase is implemented with Python programming language. The implementation consists of four main classes: AFM, EnergyPlanner, Resource and MqttInterface. The AFM class is the one that needs to be initialized to run the AFM module. It aggregates all the other classes and provided the main run loop for the AFM module. The EnergyPlanner implements the Energy Planner component of the AFM architecture. The Resource class implements Controller component. It also provides a uniform interface to the Digital Twin Repository module enabling the EnergyPlanner to access baseline and flexibility of the consumer. It should be noted that the Resource class implementation depends on the consumer and assets available in the consumer premises. Therefore, the main module provides only an abstract class that needs to be implemented and customized for different types of consumers. The MqttInterface class implements the MQTT interface of the AFM module. MQTT based interface was selected to allow lightweight communication protocol based on publish and subscribe communication pattern. The AFM module is customized for three different pilot sites in Finland, Greek and Slovenia. These instances and their differences are also briefly documented in this deliverable.

## 2 Introduction

### 2.1 Purpose, context and scope

The purpose of this deliverable is to document the third phase version of the Automated Flexibility Management (AFM) module developed in the Task 3.4 - *Automated decision-making and energy optimization*. The role of the AFM module is to provide automated and optimal flexibility management to maximize consumer benefits. This end, it utilizes digital twins of the consumer to plan and optimize control policies in the digital world and then select the best actions to be executed in the real world. This is achieved by using model predictive control with the digital twins created using the hybrid modelling approach combining machine learning with physics-based modelling (see D3.2 for more details). Furthermore, the control strategy involves multiple layers of control, including rule-based control to ensure safe and robust control.

The initial version of the AFM module, documented in D3.7, focused on explicit demand response functionality. The first version included the initial implementations of flexibility evaluation, control and interface specifications with other software modules. Evaluating flexibility and control was done via heuristic approach whereas the energy consumption optimization and implicit demand response were not considered at the phase one. Moreover, the first version was only customized and later deployed into the Finnish pilot focusing on apartment building flexibility management.

In the phase two implementation, documented in D3.8, we extended the initial prototype of the AFM module with new features including implicit demand response and local control. The architecture is also updated to better match with the new Customer Energy Manager Standard (i.e., EN 50491-12-x standard series). Additionally, the first AFM-implementations tailored for the assets in Greek and Slovenian pilot were implemented.

In the third and final version of the AFM, documented in this deliverable, we have tailored the implementation for new sites, and implemented a few small changes to the AFM interface based on the feedback obtained in phase 2. The main changes, compared to D3.8, are presented in section 2.2.

### 2.2 Main changes compared to D3.8

The main changes in D3.9 compared to the previous version of the deliverable are listed below:

- Section 3: The use case and requirement status has been updated. A third phase focus section has been added.
- Section 5.1: The flexibility management interface of the AFM module has been updated.
- Section 5: The AFM instantiations section has been updated with new instantiation for a supermarket in the Finnish pilot (section 5.2.1.2). Additionally, the implementation for the Greek and Slovenian pilots (sections 5.2.2 and 5.2.3) have been updated based on the experiences and limitations encountered during the pre-piloting phase.

### 2.3 Content and structure

This deliverable is structured as follows:

- Section 3 provides an overview, mapping the contents of the deliverable to the use cases and to the iFLEX architecture.
- Section 4 introduces the main methods and approaches applied in the work and presents the logical control architecture for the AFM module.
- Section 5 details the implementation of the Automated Flexibility Management module, outlining the software implementation, interfaces and real-world instantiations of the AFM module in the pilots.
- Section 6 concludes the deliverable.

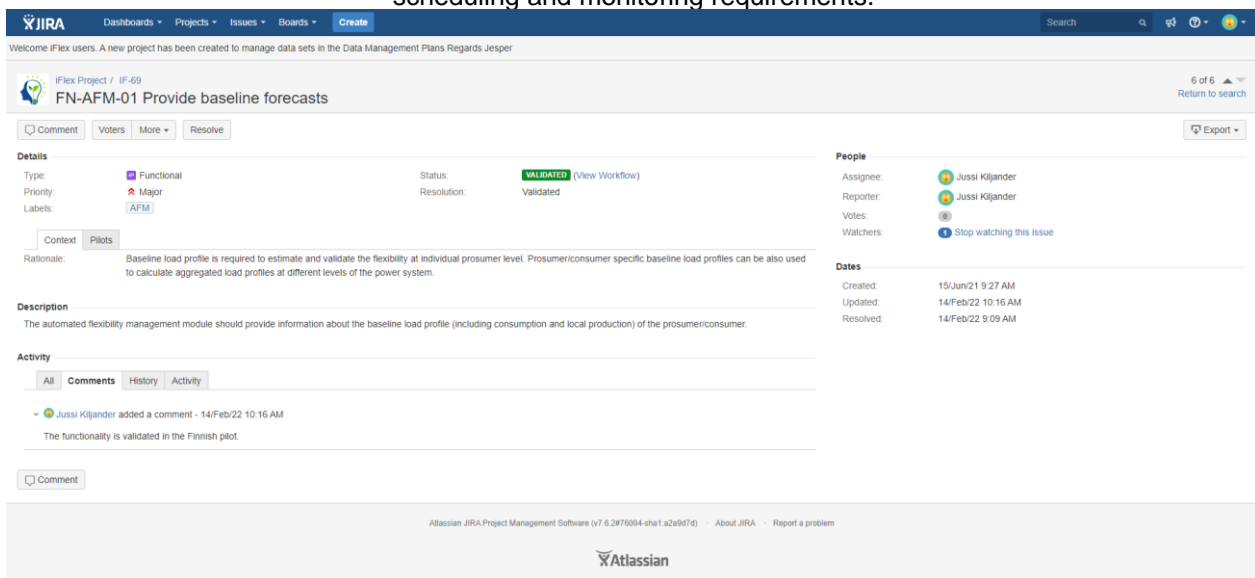
### 3 Overview

#### 3.1 Relation to use cases

The use cases of the iFLEX project are documented in D2.1 - Use cases and requirements. Based on the use cases and system level requirements, component specific requirements have been defined for different functional components of the iFLEX Framework. Four high level requirements have been specified for the Automated Flexibility Management (AFM) component documented in this deliverable. The requirements are related to all three of the high-level requirements. The following list introduces the AFM specific requirements and maps them to the Primary Use Cases (PUC):

- FN-AFM-01 - Provide baseline forecasts, related to PUC-8
- FN-AFM-02 - Flexibility potential, related to PUC-8
- FN-AFM-03 - Activate offered flexibility, related to PUC-9
- FN-AFM-04 - Optimize flexibility based on prices (implicit demand response), related to PUC-9
- FN-AFM-05 - Optimize flexibility locally (maximize self-consumption), related to PUC-10.
- FN-AFM-06 - Provide schedule information for the End-user Interface

The requirements in iFLEX are managed via Jira tool that provides methods for creating, prioritising, scheduling and monitoring requirements.



The screenshot shows a Jira issue page for 'FN-AFM-01 Provide baseline forecasts'. The issue is in the 'VALIDATED' status. The details section shows the type as 'Functional', priority as 'Major', and label as 'AFM'. The rationale states: 'Baseline load profile is required to estimate and validate the flexibility at individual prosumer level. Prosumer/consumer specific baseline load profiles can be also used to calculate aggregated load profiles at different levels of the power system.' The description states: 'The automated flexibility management module should provide information about the baseline load profile (including consumption and local production) of the prosumer/consumer.' The activity section shows a comment by Jussi Kiljander: 'The functionality is validated in the Finnish pilot.'

Figure 1 illustrates how the requirements are managed in Jira. The full list of requirements is presented in the Appendix. All the requirements have been implemented and most of them also validated in phase 2. In phase 3 pilots the remaining requirements will be validated with pilots in operational environment.

The screenshot displays the Jira interface for the 'iFlex Project'. The top section shows a search results table with the following data:

Summary	Assignee	Reporter	P	Status	Resolution	Created	Updated	Source
FN-AFM-06 Provide schedule information for the End-user Interface	Jussi Kiljander	Jussi Kiljander	🔴	IMPLEMENTED	Unresolved	09/Oct/23	09/Oct/23	
FN-AFM-05 Optimize flexibility locally (self-consumption, consumer load reduction)	Jussi Kiljander	Jussi Kiljander	🔴	VALIDATED	Validated	30/Jul/22	14/Apr/23	
FN-AFM-04 Optimize flexibility based on prices (e.g. implicit demand response)	Jussi Kiljander	Jussi Kiljander	🔴	IMPLEMENTED	Unresolved	15/Jan/21	09/Oct/23	
FN-AFM-03 Activate offered flexibility	Jussi Kiljander	Jussi Kiljander	🔴	VALIDATED	Validated	15/Jan/21	14/Feb/22	...
FN-AFM-02 Flexibility potential	Jussi Kiljander	Jussi Kiljander	🔴	VALIDATED	Validated	15/Jan/21	14/Feb/22	
FN-AFM-01 Provide baseline forecasts	Jussi Kiljander	Jussi Kiljander	🔴	VALIDATED	Validated	15/Jan/21	14/Feb/22	

The bottom section shows the details of issue 'FN-AFM-01 Provide baseline forecasts'. Key information includes:

- Type:** Functional
- Priority:** Major
- Status:** VALIDATED (view Workflow)
- Resolution:** Validated
- Labels:** AFM
- Rationale:** Baseline load profile is required to estimate and validate the flexibility at individual prosumer level. Prosumer/consumer specific baseline load profiles can be also used to calculate aggregated load profiles at different levels of the power system.
- Description:** The automated flexibility management module should provide information about the baseline load profile (including consumption and local production) of the prosumer/consumer.
- Activity:** Jussi Kiljander added a comment - 14/Feb/22 10:16 AM: The functionality is validated in the Finnish pilot.

Figure 1: AFM requirements updated in the project's Jira tool.

### 3.2 Relation to the functional architecture of the iFLEX Framework

Automated Flexibility Management module is at the centre of the iFLEX assistant framework, as is depicted in Figure 2. The main responsibilities of the module are:

- Forecasting on status and energy consumption of the building.
- Evaluation potential flexibilities on both electricity and district heat vectors.
- Optimization building demand-response with respect to energy price, CO2 emissions etc.
- Production of control signals to BEMS/HEMS according to the activated flexibility or optimized load plan.

Forecasting is done by utilizing Digital Twin repository, which contains, as the name implies, all models related to the system. These models provide forecasts on energy loads, flexibility and response of flexible assets with respect to various control inputs. The models are documented in detail in *D3.3 - Final Hybrid Modelling Module*. Communication with BEMS/HEMS is done via Resource Abstraction Interface (RAI), which is documented in *D4.3 - Final Resource Abstraction Interface*. This interface also provides access to external data sources, such as weather and CO2 emissions data. Additionally, the AFM module communicates with the end-user (through end-user interface, see *D3.6 - Final Natural User Interfaces*) for receiving user-defined comfort and/or consumption preferences as well as request approval on changes to baseline consumption. Finally, activation of flexibilities is received from Aggregator and market interface, documented in *D4.6 - Final Market and Aggregation Interface Module*.



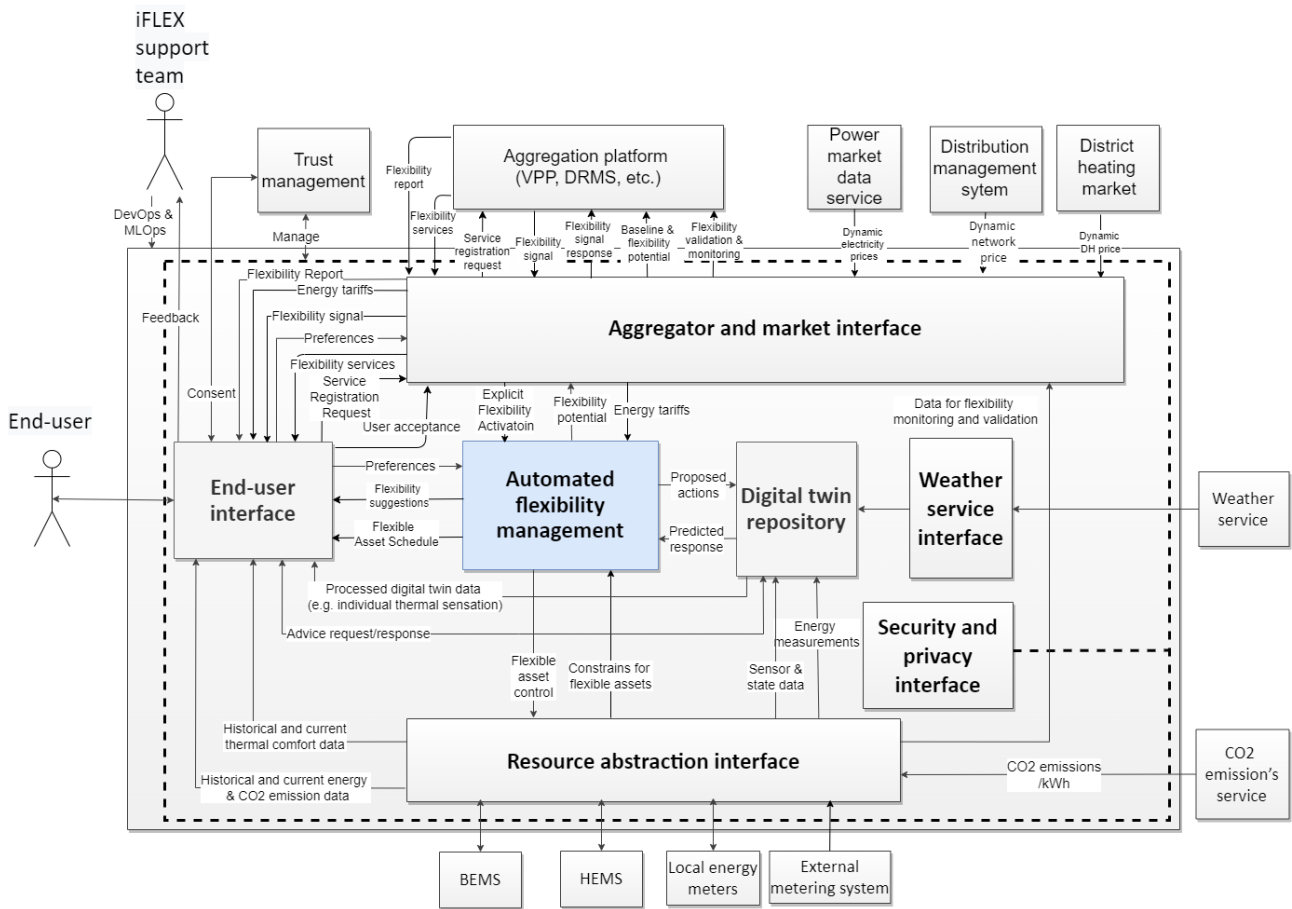


Figure 2. Functional view of the iFLEX assistant with Automated Flexibility Management module highlighted.

### 3.3 Third phase focus

The third phase focused on replicating the AFM implementations into new sites in the Greek, Slovenian, and Finnish pilots. To this end, the work focused on implementation and configuration of controllers (also called Resource Managers) for assets in those pilots. In the Greek pilot, we revised the controller implementation for boilers based on the feedback obtained from phase 2. Furthermore, we integrated additional sub-metering devices (smart plugs) in order to offer more detailed monitoring capabilities. For the Slovenian pilot the work focused on controller customization for household heating systems supplied by air-to-water heat pumps. In the Finnish pilot the work focused on controller implementation for a supermarket's integrated heating system. In addition to the instantiations, the MQTT interface of the AFM was updated for phase 3 based on the lessons learned in phase 2.

## 4 Methodology and approach

### 4.1 Functional architecture

The functional architecture of the Automated Flexibility Manager is depicted in Figure 3. The figure presents the internal architecture of the AFM module and the other iFA functional components interacting with the AFM module. The AFM consist of two types of components: Energy Planner and Controller.

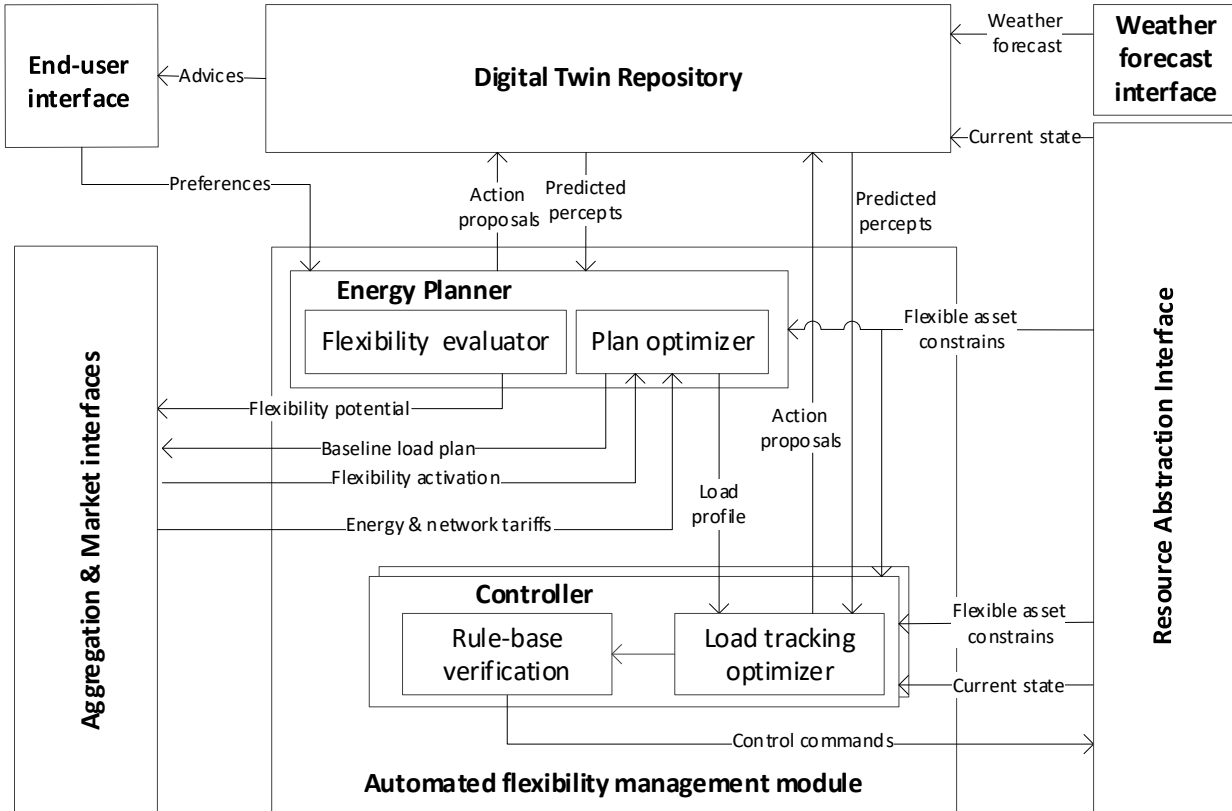


Figure 3: Functional view of the Automated Flexibility Management module.

There is a single Energy Planner component for each AFM/iFA. It is responsible for:

- 1) Aggregating and optimizing flexible assets within the consumer premises.
- 2) Delivering baseline and flexibility forecast to Aggregation and Market interface.
- 3) Reacting to price, incentive, and explicit control signals to maximize consumer benefits while meeting their preferences.
- 4) Informing each Controller about the flexible asset specific load profiles the asset should follow.

The Energy Planner utilizes MPC-based (Kiljander et al., 2021) approach for optimizing the energy and flexibility management within consumer premises. The Digital Twin Repository (final version documented in D3.3) provided the models utilized for searching optimal load profile for each flexible asset. The optimality of the load plan is measured by a reward (or cost function), which depends on the market setting and end-user preferences. Additionally, the Energy Planner can utilize models for inflexible loads and local generation from RES generation to optimize the load profiles. A formal representation of the Energy Planner's objective is presented in (3). The functions  $f_f$ ,  $f_g$  and  $f_d$  are represented with the hybrid models provided by the Digital Twin Repository.

There is a single Controller for each flexible asset (or logical group of flexible assets) within the consumer premises. Each controller is responsible for following the individual load plan provided by the Energy Planner. The optimization problem presented here can be labelled as a load following problem, which in standard optimization problem form can be presented as

$$\begin{aligned} \min_{a_1, \dots, a_T} \sum_{t=1}^T (E_t - \hat{E}_t)^2 \quad (1) \\ \text{s. t. } s_t = f_f(s_{t-1}, a_{t-1}) \\ s_{min} \leq s_T \leq s_{max} \\ \hat{E}_t \in s_t, \end{aligned}$$

where  $E_t$  is the energy in the load plan,  $\hat{E}_t$  is the energy consumption predicted by the model  $f_f$ , and  $s_t$  is the state of the system including the energy consumption and user comfort. This can be solved in many ways, for example if  $f_f$  is a continuous function, convex optimization algorithms such as quadratic programming can be used (Bianchini et al., 2016; West et al., 2014). However, if the control horizon is short and control space limited, a brute force (guess and check) method will suffice.

Depending on control frequency and resource dynamics the Controller can be implemented either with traditional reactive control mechanisms such as Proportional-Integrative-Derivative (PID) control or with more complex predictive control approach utilizing the models provided by the Digital Twin Repository. Important part of each Controller is also a rule-based control system that ensures safe and reliable operation of the system in case AI-based control is unsure of its decision or is otherwise compromised.

In addition to these two components the energy management system (EMS), interfaced via the Resource Abstraction Interface of the consumer premises, is an important part of the control architecture. The EMS is responsible for ensuring the normal operation, i.e. operating range, of the system and also translates more high-level control (e.g. temperature set point) signals into actuator control signals. An example of this could be the controller inside a heat pump that turns the pump on and off while maintaining a desired temperature. In addition, it will maintain its parameters in operational range even by overriding incoming control signals if necessary. Typically, a Proportional-Integral-Derivative (PID) controller is used, which uses feedback mechanism to minimize error between desired set point and measured process variable.

Two fundamental approaches can be recognized for Artificial Intelligence based optimal control, namely *model-free* and *model-based* optimal control. In model-free control, as the name implies, there is no dynamics model to simulate states from. A popular technique is to use *reinforcement learning* in which the agent learns to control the system through interaction with the environment, bearing similarities with human learning. With the rise of neural networks, the agent used today is typically a neural network, in which case the technique is labelled as *deep reinforcement learning (DRL)*. Among different DRL algorithms, the most popular for DR control is *Q-learning* (Sutton & Barto, 1998), which has been battle-tested in many scenarios (Chen et al., 2018; Patyn et al., 2018; Ruelens et al., 2017; Ruelens et al., 2015). In Q-learning, the agent learns a function, labelled as Q-function, that estimates the *value* of an action in a given state (Figure 4). It does this by exploring the system by doing different actions and observing different reward-action pairs and then updating the value function using the following equation iteratively:

$$Q^{new}(s_t, a_t) \leftarrow Q^{old}(s_t, a_t) + \alpha \left( r_t + \gamma \max_a Q(s_{t+1}, a) - Q^{old}(s_t, a_t) \right), \quad (2)$$

where  $s_t$  is the state,  $a_t$  is the action,  $\alpha$  is the learning rate and  $\gamma$  is the future rewards discount factor. In essence, the new value of Q-function is obtained by adding the total net reward scaled by the learning rate to the previous value.

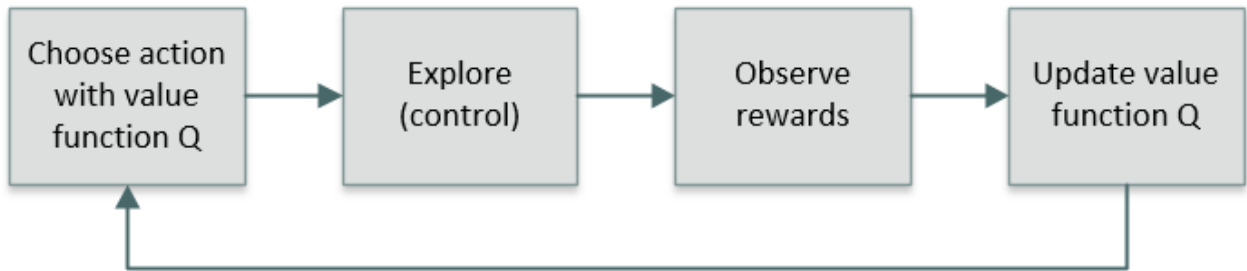


Figure 4. Q-learning algorithm.

In contrast to model-free algorithms, model-based control techniques rely on dynamics model, which is used to simulate different control scenarios. This optimal control problem for DR can be presented in general level as (Kiljander et al., 2021):

$$\begin{aligned} & \max_{a_1, \dots, a_T} \sum_{t=1}^T r(s_t, a_t) \\ \text{s. t. } & s_t = f_f(s_{t-1}, a_{t-1}) + f_g(s_{t-1}) + f_d(s_{t-1}) \\ & s_{\min} \leq s_t \leq s_{\max}, \end{aligned} \quad (3)$$

where  $r$  is the reward function,  $s_t$  is the state of the system, and  $a_t$  is the action. The  $s_{\max}$  and  $s_{\min}$  represent possible constraints such as the minimum and maximum values for indoor temperature. The  $f_f$ ,  $f_g$  and  $f_d$  represent models for flexible resources, power generation and inflexible demands, respectively. The dynamics models  $f_f, f_g, f_d$  can be any functions, from simple heuristics to deep neural networks. However, the choice of the models also dictates the possible algorithms that can be used to solve the problem. Generally, the optimization algorithms can be divided into *gradient-based* and *gradient-free* methods. Gradient-based methods are usually faster and also can (mathematically) guarantee an optimal solution but are notoriously difficult to use with neural network models. This is mainly due to the exploding and vanishing gradient problem (Pascanu et al., 2013). Gradient-free methods, such as genetic algorithms and particle swarm optimization (Kusiak et al., 2014; Ma & Wang, 2011; Tang & Xu, 2011), do work well with neural networks and are used extensively with them but lag on accuracy and efficiency compared to gradient-based methods. In addition, model-based optimal control with neural networks typically employs a *Nonlinear Model Predictive Control (NMPC)* approach, a form of closed-loop control, where the control algorithm uses feedback from the system when making new decisions, as is shown in Figure 5 (Kiljander et al., 2021). In practice, this means that a new optimized plan is done at every time step with the latest information available, compared to doing it only once per period. An upside of this approach is that it leads to more optimal control with a trade-off in plan predictability.

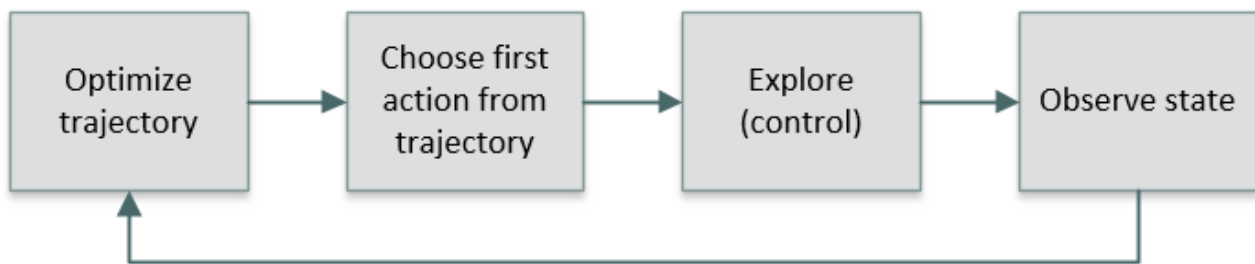


Figure 5. Model predictive control (MPC) method for model-based optimal control.

## 4.2 Monte-Carlo Tree Search for optimal control with hybrid models

The model predictive control withing the AFM is realized with the Monte Carlo Tree Search (MCTS) algorithm (Browne et al., 2012). MCTS is a simulation-based tree search algorithm that has become popular in Artificial Intelligence (AI) solutions for game playing such as the AlphaGo (Silver, 2016), and AlphaZero (Silver et al., 2018) developed by the DeepMind. These solutions combine MCTS with deep reinforcement learning (DRL) to achieve state-of-the-art performance in Shogi, Go, and Chess.

To the best of our knowledge, the optimal control solution developed for AFM module is the first that applies MCTS for demand-side flexibility management and building energy optimization. MCTS provides a lightweight optimization framework that is a natural selection over derivative-based methods when the control inputs are not continuous (e.g., the heat pump setpoints in this case). Furthermore, the main reason for using MCTS is that it is suitable for the hybrid-models (documented in D3.1, D3.2 and D3.3) that do not provide gradients to be used for optimization.

In MCTS the search space of possible trajectories (alternative control actions at different time periods) is represented as a tree. The goal is to find the most optimal set of actions that maximizes the reward. Each execution is an iterative process that simulates many trajectories starting from the current state and running to a terminal state (i.e., end of the optimization window). MCTS uses Monte Carlo simulation to estimate the value of different states to guide the search toward optimal trajectories in the search space. In addition to exploiting the trajectory that is currently perceived as the optimal one, MCTS continues to evaluate other alternatives periodically. This balancing between the optimal strategy and search for alternative trajectories is known as the *"exploration-exploitation trade-off"*. There are many variants of MCTS proposed in the literature. The current implementation of the algorithm for the AFM module comprises of following four steps: *selection, expansion, simulation, and backpropagation*. (Sutton & Barto, 2018)

In the **selection step** the algorithm selects an optimal path in the search tree using a tree policy. The path represents the optimal actions (e.g. HVAC set points) at different time periods. The search starts from the root node (i.e., current state of the system) and continues until a leaf node is reached. A leaf is any node that has a child node from which no simulation has been executed. An important part of the tree policy is to balance the exploration and exploitation (i.e., how much to emphasize the most promising parts of the search space versus exploring the tree for new regions which could provide even more value). The tree policy implemented for the AFM modules uses the Upper Confidence Bound applied to Trees (UCT) (Kocsis & Szepesvári, 2006) selection rule to balance exploration-exploitation trade off. The equation below specifies how the value for each node is calculated. The node with the highest estimated value is selected.

$$V_i = x_i + C \sqrt{\frac{\ln(n_p)}{n_i}} \tag{4}$$

$V_i$  is the estimated value of the node  $i$ ,  $x_i$  is the empirical mean value of the node  $i$ ,  $C$  is a constant used for balancing between exploitation and exploration, and  $n_i$  and  $n_p$  are the number of times the node  $i$  and its parent have been visited, respectively. A typical value for the  $C$  in game playing is  $\frac{1}{\sqrt{2}}$ . This value is found to be optimal when the rewards remain within the range  $[-1, 1]$ . To avoid the hyperparameter search for this variable in the demand-side flexibility management we adopted the approach where the rewards are scaled so that they remain within the range  $[-1, 1]$  as much as possible. The scaling factors were estimated by running a fixed setpoint policy with historical data.

In the **expansion step** the search space (tree) is expanded with new node. This step is executed unless the leaf node is the last time step in the optimization window. The new node is created by randomly choosing an action (i.e., set point) and sampling a valid action with the hybrid models.

The purpose of the **simulation step** is to evaluate the value of the current state (and consequently its parents states). To this end, a rollout from the new node, expanded in the previous step, is completed. The rollout consist of sampling new states, using a simulation policy, until the end of the optimization window is reached. Instead of using the typical random policy as the simulation policy, the AFM modules simulates with a fixed policy based on the default control strategy (e.g. fixed set point for temperature). The fixed setpoint policy is used, because it is more likely policy than random sampling and provides thus more accurate estimate for the state values.

In the **backpropagation step** the empirical value ( $x_i$ ) of the nodes are updated or initialized based on the results obtained from the simulation step. Only values for the tree are updated (i.e., no values for states beyond the three that were visited during the simulations step).

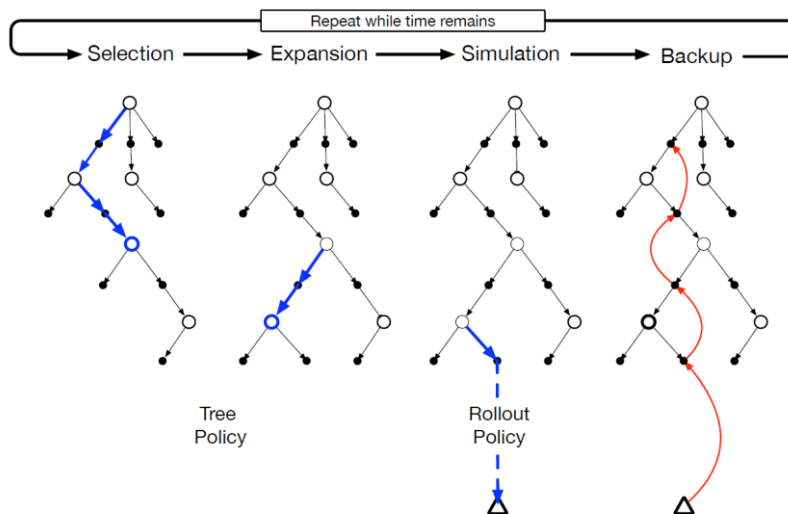


Figure 6. Monte Carlo Tree Search. Reprinted from Image from An Introduction: Reinforcement Learning by Sutton & Barto (Sutton & Barto, 2018)

MCTS executes iteratively the abovementioned steps until the time allocated for optimization runs out. Every iteration is started from the root node (i.e., current state of the system). Finally, the action with highest empirical value to be executed from the root node is selected. After the environment transitions to a new state, MCTS is

run again starting with a tree containing any descendants of the new state (i.e., node) left over from the tree constructed by the previous execution of MCTS.

## 5 Implementation

### 5.1 Overview

The AFM module is implemented with Python programming language. Code implementation follows object-oriented programming paradigm and is overall designed with expandability in mind. Interfacing with other modules is done via MQTT protocol, using the Eclipse Paho MQTT Python client library. Data between software modules is transferred in Pandas DataFrame format, serialized to JSON notation. Various run parameters are specified in a separate configuration file, where data polling, flexibility evaluation, planning and control intervals are set as well as paths and MQTT topics are configured.

#### 5.1.1 Software architecture

A Unified Modelling Language (UML) class diagram of the AFM implementation is depicted Figure 7. The implementation consists of five classes: AFM, EnergyPlanner, Resource, MqttInterface and OperationTimer.

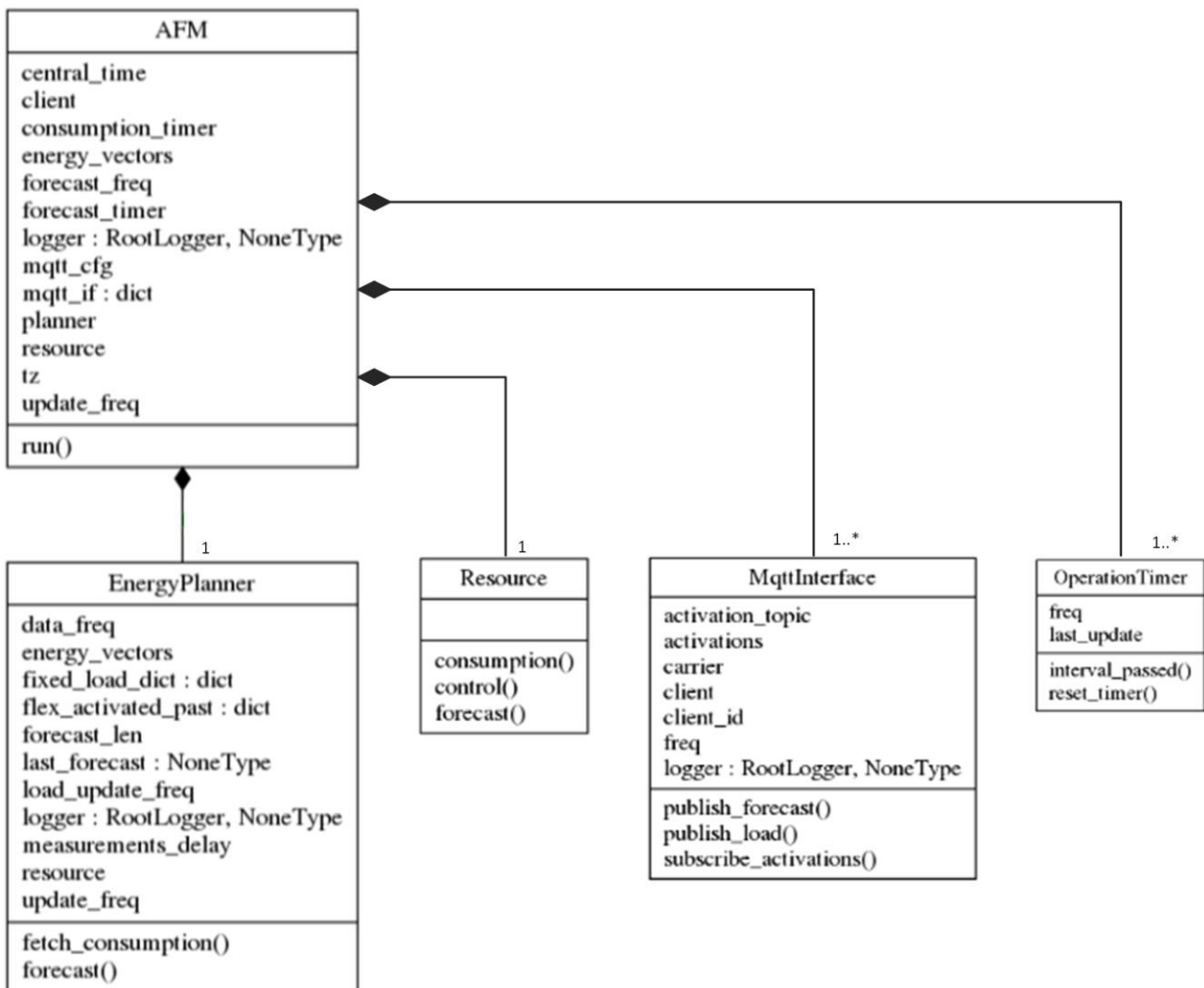


Figure 7: UML class diagram for the Automated Flexibility Manager.

The AFM class represents the whole Automated Flexibility Manager entity. To setup the AFM for a new consumer an instance of this class is created. The documentation of the class is represented below.

```
class AFM(flex_resource, cfg=None)
```

Automated Flexibility Manager.

**Parameters**

**flex\_resource** : resource.resource  
Instance of the resource class.

**cfg** : dict  
Config for the AFM instance.

**Methods**

**def run(self, only\_log\_exceptions=True)**

Main loop for the Energy Planner

**Parameters**

**only\_log\_exceptions** : bool  
If true exceptions are only logged (not raised).

The EnerPlanner class represent the Energy Planner component of the AFM architecture. The documentation of the class is represented below:

**class EnergyPlanner(flex\_resource, cfg=None)**

Energy Planner component of the Automated Flexibility Manager.

**Parameters**

**flex\_resource** : resource.resource  
Instance of the resource class.

**cfg** : dict  
Config for the AFM instance.

**Methods**

**def fetch\_consumption(self, now)**

Fetches consumption data.

**Parameters**

**now** : Datetime  
Current time.

**Returns**

dict - Dictionary of Dataframes (a DF for each energy vector). DF contains the consumption data.

**def forecast(self, now, activations)**

Forecast the baseline and flexibilities.

**Parameters**

**now** : Datetime  
Current time.

**activations** : dict  
New DR activations. Dictionary of DataFrames for each energy vector.



## Returns

dict - Dictionary of Dataframes (a DF for each energy vector). DF contains the forecast.

The Resource class represents the consumer. In the current implementation the whole consumer (i.e., people, building, flexible assets, local production) is represented as a single resource. The Resource class implements the controllers for each flexible asset. It also provides the EnergyPlanner with baseline and flexibility forecasts by acting as an interface to the Digital Twin Repository. The implementation of this class is consumer specific and only an abstract class (defining the interface) is implemented as part of the generic AFM library. The documentation of the class is represented below:

### class Resource

This is a blueprint for the flexible assets and baseline loads.

In the current implementation the whole consumer premises needs to be implemented as a single resource. (i.e., not possible to divide the consumer into several resources).

## Methods

### def consumption(self, start, end)

Returns consumption between start and end in the market resolution.

#### Parameters

**start**: datetime.datetime  
Start of the period.

**end**: datetime.datetime  
End of the period.

#### Returns

DataFrame - DF with consumption columns for each energy vector, e.g. [ele\_load, dh\_down]

### def control(self, now, fixed\_load)

Sends control commands to the flexible resource to follow the load plan.

#### Parameters

**now**: datetime.datetime  
Current datetime.

**fixed\_load**: dict  
Dictionary of fixed loads (series) for every energy vector. Fixed loads represent loads planned by the AFM which need to be followed by the resource.

### def forecast(self, now, start, end, fixed\_load)

Forecasts the baseline and flexibilities.

Only down flexibilities are provided in the current version.

#### Parameters

**now**: datetime.datetime  
Current datetime.

**start** : datetime.datetime  
Start of the forecast period.

**end** : datetime.datetime  
End of the forecast period.

**fixed\_load** : dict  
Dictionary of fixed loads (series) for every energy vector. Fixed loads represent loads planned by the AFM which need to be followed by the resource.

### Returns

DataFrame - Forecast represented as a dataframe with following columns  
[\_baseline, \_\_down]

The MqttInterface class implements the MQTT interface of the AFM module. The documentation of the class is represented below:

```
class MqttInterface(client_id, client, carrier, freq)
```

### Parameters

**client\_id** : str  
The MQTT client identifier

**client** :  
The MQTT client instance.

**carrier** : {'ele', 'dh'}  
The energy vector/carrier.

**freq** : int  
The sampling rate of the data shared via the interface.

### Methods

```
def publish_forecast(self, now, forecast_df)
```

Publish baseline and flexibility forecast to MQTT.

### Parameters

**now** : Datetime  
Current time.

**forecast\_df** : DataFrame  
DataFrame containing the baseline and flexibility forecasts.

```
def publish_load(self, now, load_df)
```

Publishes the measured load info to the MQTT broker.

### Parameters

**now** : Datetime  
Current time.

**load\_df** : DataFrame

DataFrame containing the energy measurement.

```
def subscribe_activations(self)
```

Subscribes explicit demand response activation messages.

The OperationTimer class represents timers that are used to synchronize the operation of the Energy Planner. The documentation of the class is represented below:

```
class OperationTimer(now, freq)
```

#### Parameters

**now**: datetime.datetime

Current time.

**freq**: int

Update frequency (sampling rate) of the timer.

#### Methods

```
def interval_passed(self, now)
```

Check whether the interval has passed.

#### Parameters

**now**: datetime.datetime

Current time.

#### Returns

bool

Boolean indicating whether the interval has passed.

```
def reset_timer(self, now)
```

Resets the time to start counting from now.

#### Parameters

**now**: datetime.datetime

Current time.

```
class Time(tz='UTC', update_freq=1, simulation=None)
```

This class is used to allow either simulated or real-time to be used.

#### Parameters

**tz**: str

Timezone.

**update\_freq** : int

Update frequency.

**simulation** : dict or None

Simulation specific data. None if timer is used for real-time operation.

## Methods

**def now(self)**

Get the current time.

### Returns

Datetime - Current datetime.

## 5.1.2 Use cases

### 5.1.2.1 Offer flexibility

Flexibility refers to the potential to deviate from a baseline load at different time periods. The baseline load in turn is the power load that would ensue without explicit or implicit demand response. Flexibility can be used to either increase or decrease consumers' baseline load. The potential to decrease the baseline is called down flexibility and the potential to increase up flexibility. The amount of down and up flexibility are presented more formally in equations ( 5 ) and ( 6 ), respectively.

$$D_i = C_i^{base} - \min (C_i^{ctrl}) \quad (5)$$

$$U_i = \max (C_i^{ctrl}) - C_i^{base} \quad (6)$$

Where  $D_i$  is the down flexibility and  $U_i$  the up flexibility at time step  $i$ ,  $C^{base}$  is the baseline consumption and  $C^{ctrl}$  is the consumption obtained through DR control inputs. For simplicity reasons, we only consider the maximum flexibility value for each direction (up or down) in each time step. Even still, calculating the maximum amount of flexibility for each time step is not trivial. If we want to evaluate all possibilities, we would need to calculate  $\sum_i^r n^i$  options, where  $n$  is the number of control options,  $r$  is the length of the forecast and  $i$  is the current time step. In practice this exponential growth implies that for any forecasts longer than a couple of steps it is impossible to use brute force (calculate all possible trajectories), so instead a heuristic or algorithm is used to reduce search space. One option is to perform control only in the step that is being evaluated while leaving the other as is. This approach reduces the number of iterations from  $\sum_i^r n^i$  to  $r * n$ , which is much more feasible to calculate in short time.

Figure 8 depicts the flexibility evaluation process that is initiated by a timer (the OperationTimer is omitted from the figure for the sake of clarity). The EnergyPlanner first requests the baseline load and flexibility from each Resource (only one resource per site is supported in the current implementation). The Resource calls the relevant models from the Digital Twin Repository to predict the baseline, minimum and maximum loads. This information is then returned to the AFM class that calls the MqttInterface to send the flexibility and baseline forecast to the Aggregation & Market Interface module.

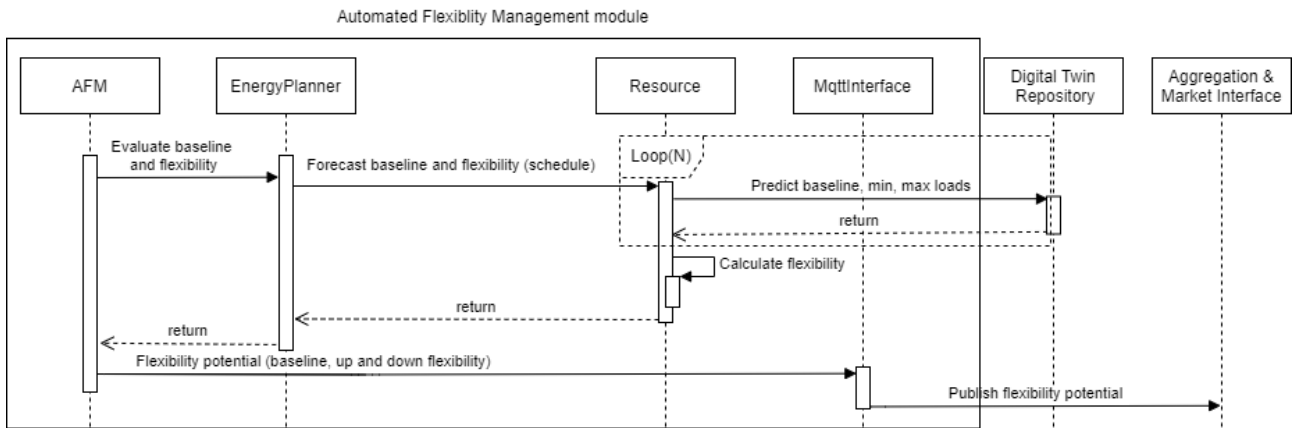


Figure 8. Sequence diagram of the flexibility offering process.

**5.1.2.2 Optimize schedule consider prices and/or incentives**

The schedule optimization use case covers both implicit and explicit demand response. In both cases the schedule optimization can be divided into two distinct parts: planning and control. The EnergyPlanner class is responsible for the planning while the control is implemented by the Resource class (i.e., Controller is embedded into the Resource class). The planning part is slightly different in implicit and explicit DR. The control part, focusing on following the load plan specified by the EnergyPlanner, is executed in the same way in implicit and explicit DR. Figure 9 illustrates the implicit DR optimization process.

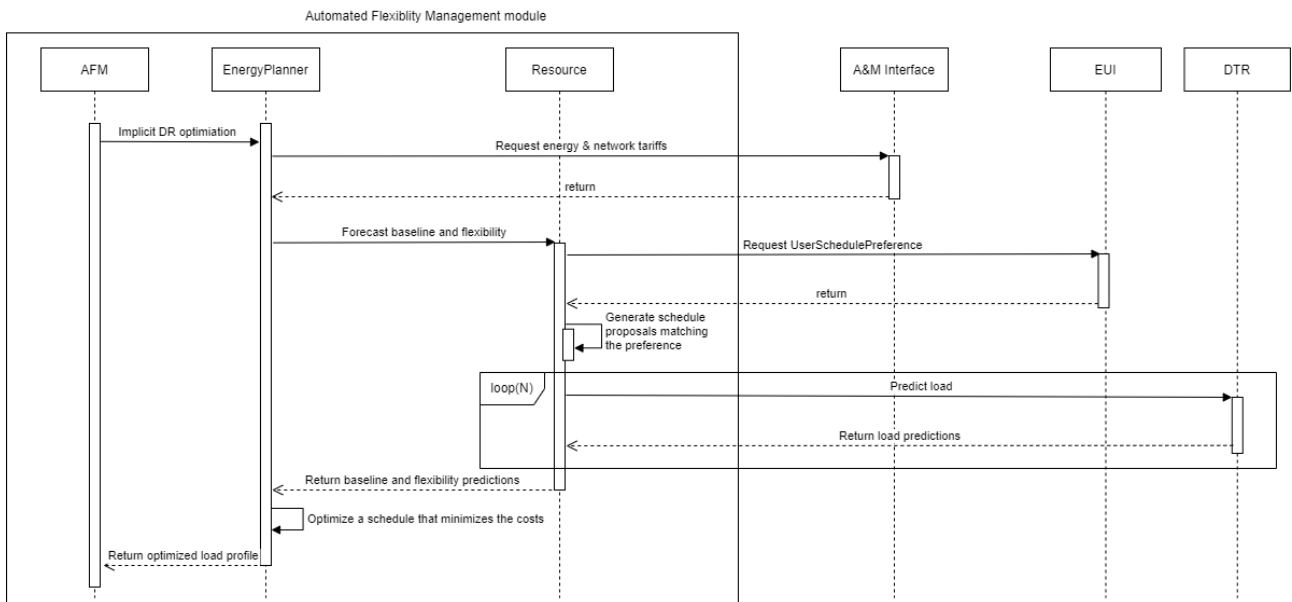


Figure 9: Sequence diagram depicting the implicit DR optimization.

The optimization is initiated by a timer in the AFM class (OperationTimer class is omitted from the figure for the sake of clarity) that synchronizes the implicit DR optimization at fixed intervals (defined in the AFM configuration file). The EnergyPlanner class is the main responsible for planning the optimal schedules for each Resource. It first fetches the energy and network tariffs from the A&M Interface and then requests the baseline and flexibility forecast from the Resource. How the baseline and flexibility forecast are formed depends on the type of resource. In this example, the Resource first fetches user preferences from the user interface component. Then it generates schedule proposals matching the user preferences and request load profiles from the Digital Twin Repository (i.e., DTR) module. The EnergyPlanner then utilizes the baseline and flexibility forecast to find optimal load profile for each Resource (remember that only one Resource is supported in the current implementation). The optimal load profile is the one that minimizes the cost function. It should be noted that the term cost refers to the total value of the cost function (consumer specific) and can in theory include non-monetary parameters such as CO2 emissions.

Figure 10 represents the explicit DR optimization process. This example assumes that manual acceptance is not required from the end-user in this case. From the AFM point of view the scenario is initiated by the

Aggregator and Market Interface. It utilizes the MQTT interface of the AFM module to notify about the explicit DR activation. Next, the AFM requests the EnergyPlanner to optimize a new load profile that matches the new flexibility activations. In this example this is done by requesting the Resource to forecast a load plan matching the activations. Finally, the AFM notifies the A&M Interface about the flexibility activation.

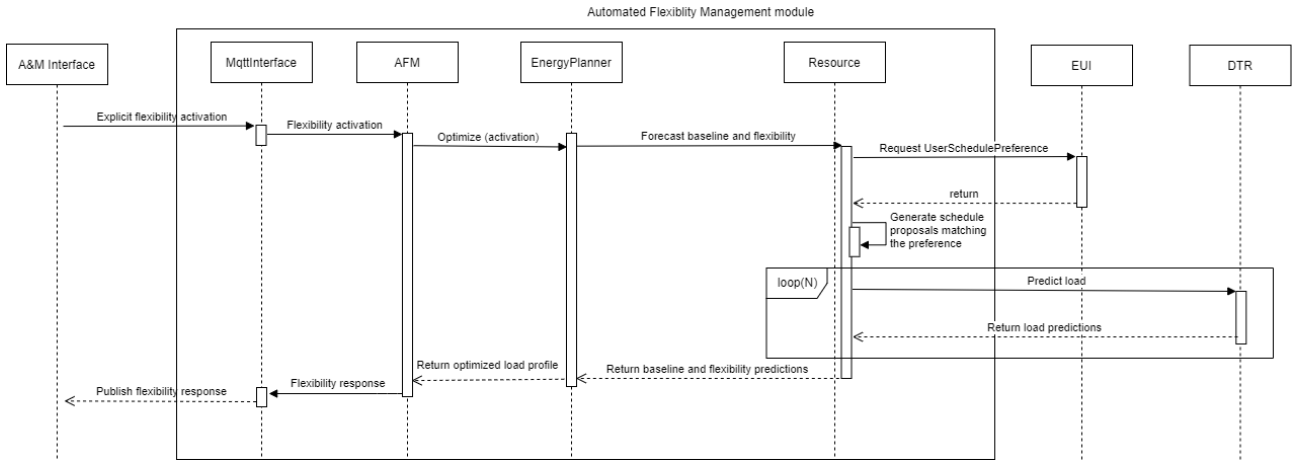


Figure 10: Sequence diagram illustrating explicit DR signal and schedule optimization.

The control part of the optimization process is represented in Figure 11. In the same way as the other operations, the control is initiated by a timer in the AFM class. The AFM class then request the optimized load plan from the EnergyPlanner and request the Resource to follow the load plan. The Resource class then calculates the set point (control command) that provides the closest response to the optimized load plan. The Resource could utilize the models in DTR for search for optimal set point but in this example is considered to be simple enough so that this is not needed.

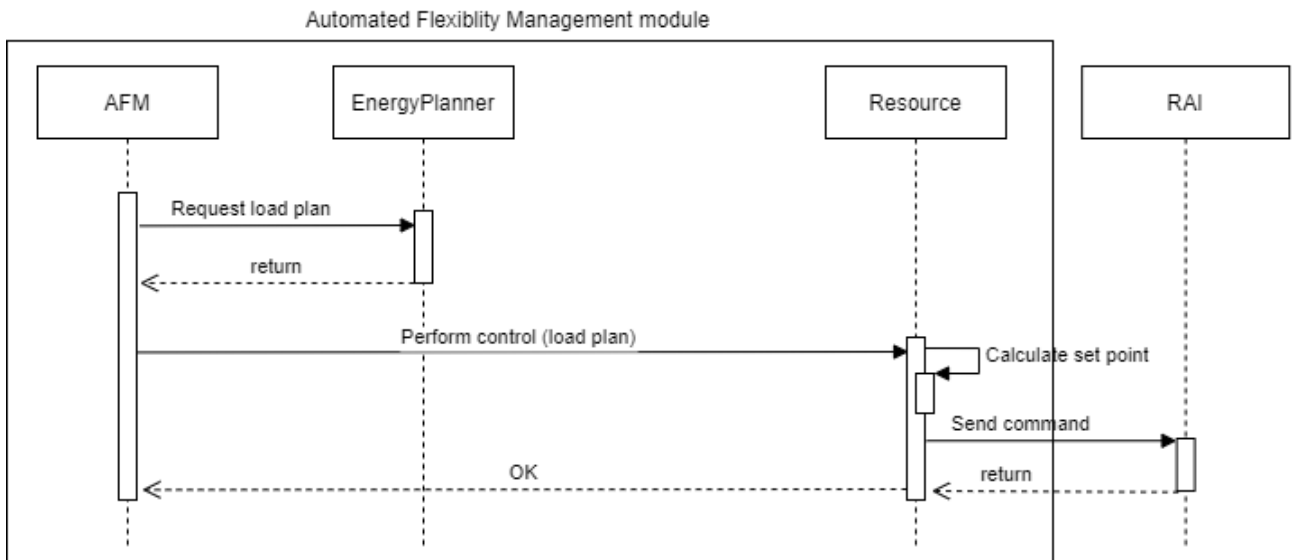


Figure 11: Sequence diagram illustrating the control process.

### 5.1.3 Interface provided by the Automated Flexibility Management module

Automated Flexibility Manager provides an MQTT interface for accessing forecasts and activating flexibilities. This interface can be utilized e.g., by an aggregator. JSON is utilized for serializing the payloads. All timestamps in the payloads are presented in ISO format. All loads are presented in kW and represent the average load during the sampling period. Table 1 introduces the MQTT topics that form the AFM interface.

Table 1. MQTT topics of AFM interface.

Interface	Topic	Message
-----------	-------	---------

<b>Flexibility potential</b>	afm/<ifa_id>/<energy_vector>/flexibility	A
<b>Explicit flexibility activation</b>	afm/<ifa_id >/<energy_vector>/activation	B
<b>Load measurement</b>	afm/<ifa_id >/<energy_vector>/load	C
<b>Flexible asset schedule</b>	afm/<ifa_id >/<asset_id>/schedule	D

As can be seen from Table 1, there are following parameters in the MQTT topics: *ifa\_id* and *energy\_vector*. The *ifa\_id* parameter is a unique identifier for the iFLEX Assistant. The *energy\_vector* specifies the energy vector for associated flexibility forecast, activation and measurements. Currently supported energy vectors include electricity (represented with keyword *ele*) and district heating (represented with keyword *dh*).

```
afm/hoas/ele/flexibility
afm/hoas/ele/load
afm/hoas/ele/activation
afm/hoas/dh/flexibility
afm/hoas/dh/load
afm/hoas/dh/activation
afm/hoas/heat_pump/schedule
```

Figure 12. Examples of the MQTT topic instances.

Figure 13 presents an example of the Message A format. This serialization format was selected as it can be easily extended with new parameters. It is also directly supported by Pandas data frame JSON serialization methods so it is easy to implement in practise. The Message A consist of header and data fields. The header field consists of *sentAt*, and *freq* fields documented in Table 2. The data field consist of a list of JSON object literals with following attributes: *timestamp*, *baseline*, *down*, *up*, *capacity*, and *allocated\_flexibility*. The semantics of these attributes is presented in

Table 3. The down and/or up attributes can be omitted if flexibility to that direction is not provided by the given AFM.

Table 2. Attributes of the Payload A header.

Parameter	Description
<b>sentAt</b>	The timestamp of when AFM sent the message.
<b>freq</b>	Sampling rate of the data in minutes.

Table 3. Attributes of a JSON object literal contained in Payload A. List of these structures is stored in the data attribute of the message.

Parameter	Description
<b>timestamp</b>	Start time of the period that the forecast targets (ISO format). End time can be calculated based on <i>freq</i> parameter present in the message header.
<b>baseline</b>	Forecasted load of the metering point (e.g., building) if the current control plan is followed. Flexibilities that are activated will modify the baseline. The values are represented in kW are refer to the average power during the time period.
<b>down</b>	Down flexibility provided for the given time period (kW). Down flexibility means reduction of the load with respect to baseline in the same forecast message.

<b>up</b>	Up flexibility provided for the given time period (kW). Up flexibility means increase of the load with respect to baseline in the same forecast message.
<b>down_capacity</b>	The capacity (kWh) for the down flexibility in the given period. The down capacity represents the total energy down flexibility starting from the given period. The down capacity can be used to estimate the number of consecutive down flexibility periods that can be activated.
<b>up_capacity</b>	The capacity (kWh) for the up flexibility in the given period. The down capacity represents the total energy down flexibility starting from the given period. The down capacity can be used to estimate the number of consecutive down flexibility periods that can be activated.
<b>allocated_flexibility</b>	Used to represent how much flexibility has been allocated. Negative value indicates down flexibility (decrease compared to the old baseline) and positive up flexibility. The purpose of this parameter is mainly to notify the aggregator on how much flexibility the AFM estimates to be able to obtain. It should be noted that this attribute does not provide any new information that cannot be obtained by comparing different forecasts (its purpose is mainly to provide all the relevant information without the need to track previous forecasts).The flexibility allocation is taken into account in the baseline (i.e., the baseline is modified with respect to the allocation) and this attribute basically reflects the change compared to the previous baseline forecast before the modification.

Eight-hour forecast is illustrated in the example. Only down flexibility is presented in the example. In addition to the baseline and down flexibility forecast, each timestamp contains an *allocated\_flexibility* field, which documents the flexibilities that will be activated by the AFM. The flexibility can be activated via the Flexibility activation interface. Message B documents the activation payload.



```

{
  "sentAt": "2022-02-18T16:00:00+00:00",
  "freq": 60,
  "data": [
    {
      "timestamp": "2022-02-18T16:00:00.000Z",
      "down": 0,
      "down_capacity": 0,
      "up": 0,
      "up_capacity": 0,
      "baseline": 71.767955924,
      "allocated_flexibility": 0
    },
    {
      "timestamp": "2022-02-18T17:00:00.000Z",
      "down": 0,
      "down_capacity": 0,
      "up": 0,
      "up_capacity": 0,
      "baseline": 71.0991792252,
      "allocated_flexibility": 0
    },
    {
      "timestamp": "2022-02-18T18:00:00.000Z",
      "down": 0,
      "down_capacity": 0,
      "up": 0,
      "up_capacity": 0,
      "baseline": 74.0061196065,
      "allocated_flexibility": 0
    },
    {
      "timestamp": "2022-02-18T19:00:00.000Z",
      "down": 0.7579430686,
      "down_capacity": 6.0387044812,
      "up": 0,
      "up_capacity": 0,
      "baseline": 75.3614803299,
      "allocated_flexibility": 0
    }
  ]
}

```

Figure 13. Payload A: Baseline and flexibility forecast payload.

Figure 14 illustrates the Message B, which is used in Flexibility activation messages. It contains following attributes: *sentAt*, *startTime*, *endTime*, and *delta*. The semantics of these attributes are elaborated in Table 4

Table 4. Attributes of the Payload B.

Parameter	Description
<b>sentAt</b>	The timestamp (ISO format) when AFM sent the message.
<b>startTime</b>	The start time (ISO format) of the flexibility event.
<b>endTime</b>	The end time (ISO format) of the flexibility event.
<b>accepted</b>	Optional attribute. List of appliances that user has authorized to be controller outside of their preferences. This is needed to inform the AFM that is should process an event happening outside of the user preferences.
<b>delta</b>	Specifies the change to the power when compared to baseline in the selected period. I.e., negative values indicate decrease to the building's load profile.

```
{
  "sentAt": "2022-05-01T14:32:41.864590+00:00",
  "startTime": "2022-05-01T11:00:00+00:00",
  "endTime": "2022-05-01T12:00:00+00:00",
  "accepted": [],
  "delta": -105.3111965137}

```

Figure 14. Payload B: Used in Flexibility activation messages.

Figure 15 presents an example payload for load measurement messages. It contains the same header field (see Table 2) as Message A. The data attribute consists of a list of JSON object literals with timestamp and load attributes.

```
{
  "sentAt": "2022-05-01T01:00:58+00:00",
  "freq": 60,
  "data": [{"timestamp": "2018-02-17T23:00:00.000Z", "load": 110.0}]}

```

Figure 15. Payload C: Used in Load measurement messages.

Message D consists of a header field (with *sentAt* attribute) and a payload field called commands. The commands attribute consists of a list of JSON objects whose parameters are described in Table 5,

Table 5. Attributes of the Message D.

Parameter	Description
<b>sentAt</b>	The timestamp of when AFM sent the message.
<b>commands</b>	A list of command objects. Each command object contains a start, end, and setpoint attributes.
<b>start</b>	The start time of the command.
<b>end</b>	The end time of the command.
<b>setpoint</b>	The set point of the command.

An example for the Message D is depicted in Figure 16.

```
{
  "sentAt": "2022-02-18T12:00:58+00:00",
  "commands": [
    {
      "start": "2022-02-18T12:00:00+00:00",
      "end": "2022-02-18T12:30:00+00:00",
      "setpoint": 1.0
    },
    {
      "start": "2022-02-18T12:30:00+00:00",
      "end": "2022-02-19T12:00:00+00:00",
      "setpoint": 0.0
    }
  ]
}

```

Figure 16. Example of the Message D, used in flexible asset schedule messages.

## 5.2 Instantiations

### 5.2.1 Instantiation in the Finnish pilot

#### 5.2.1.1 Apartment building

The apartment building consists of 93 residential apartments, each monitored for thermal comfort and air quality. All apartments share infrastructure for heating and domestic water. In addition, other notable (consumption related) infrastructure includes an elevator and a common sauna. The building is also equipped with a building automation system, including a building energy management system (BEMS).

The flexibility is provided by a centralized heating system that is responsible for space heating and heating of the domestic hot water (DHW). In practice, controlling heating and ventilation is done via actuating several components in the heating system, including valves, water pumps and heat exchangers simultaneously. To simplify control options, these controls are bundled into different control “modes” which are designed by human experts and stored in the Resource Abstraction Interface and the BEMS. The space heating can be controlled in three main modes. First, the space heating can be directly constrained by dropping the heating water temperature during a demand response event. How much this restricts the space heating in percentages is an identifiable parameter of the building’s digital twin. Please refer to *D3.3 Final hybrid-modelling module* for further details on the digital twin and modelling. Second, the heat pump can be turned off. This will put all the heat production (i.e., space heating and heating of domestic hot water) responsibility for the district heating. Third, the ventilation can be constrained during the DR event. When the ventilation is constrained, its power is reduced to 30% from the maximum. This can be done only when the heat pump is also turned off. Detailed documentation of these control points is provided in *D4.3 - Final Resource Abstraction Interface*.

A simplified overview of the energy flows in the building is presented in Figure 17. In short, incoming heating energy is coming from the district heating network and from electricity that is used to heat apartment air. Within the system is an exhaust heat pump that captures energy from exhaust air back into apartment heating. Additionally, there is a water boiler in the system.

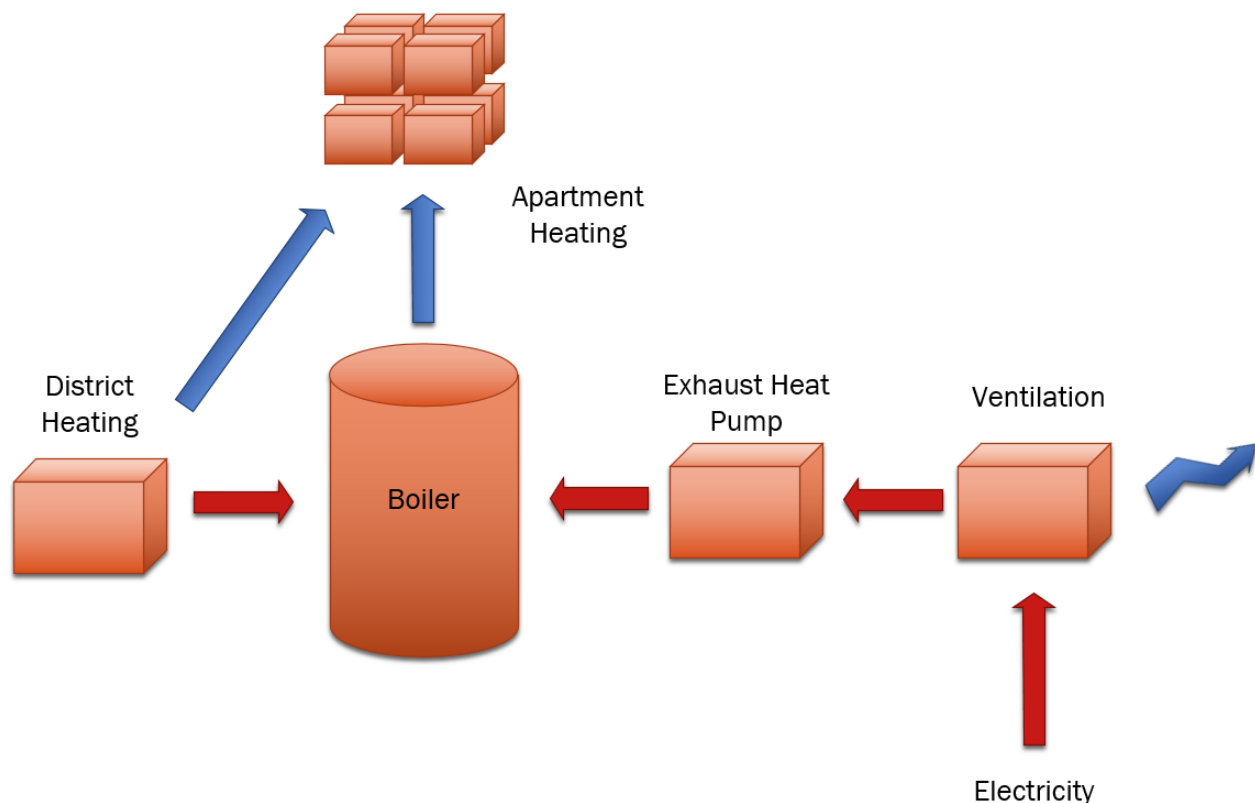


Figure 17. Simplified view of heating energy flows in the pilot building. Red arrows represent energy inflows while blue arrows indicate energy outflows.

A good selection of measurements is also available from the building. As with the control points, the full list of measurements is found in deliverable *D4.3*, but the most important are listed below:

- Building level electricity consumption (1-minute sampling rate)
- District heating consumption (1-minute sampling rate)
- Average indoor temperature (5-minute sampling rate)
- Weather forecast data from Finnish Meteorological Institute (60-minute sampling rate)

Figure 18 visualizes the AFM instance that was deployed into the Finnish pilot in the phase 2 and updated in phase 3.

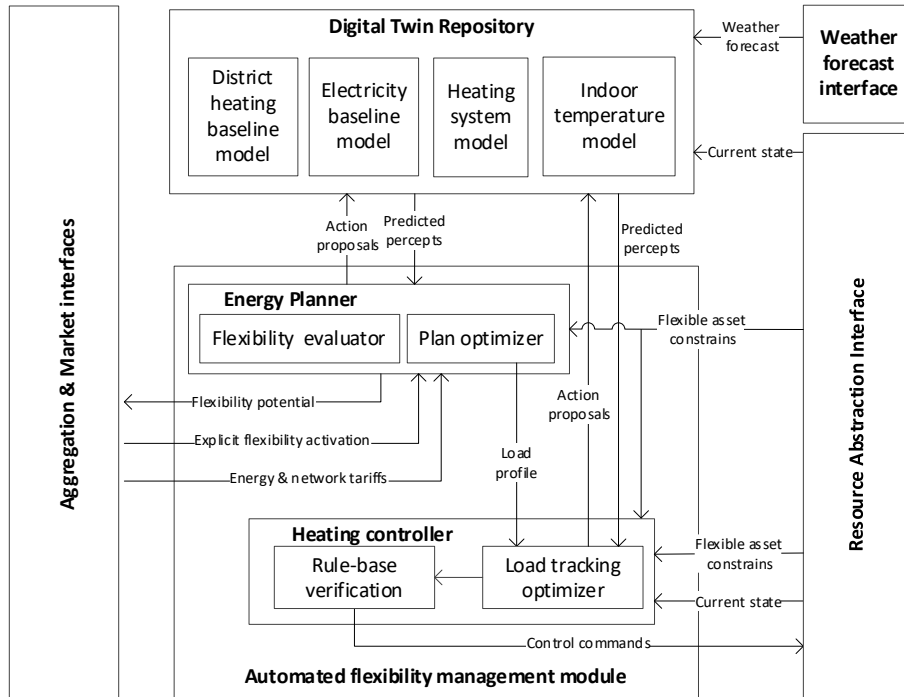


Figure 18. Functional view of the AFM instantiation in the Finnish pilot.

The AFM module instance is configured to optimize the flexibility obtained from the heating system. It consists of single controller, called Heating controller, and the common Energy Planner component. The Digital Twin Repository consist of four types of models, including District heating baseline model, Electricity baseline model, Heating system model and Indoor temperature model. The Weather forecast interface provides interface for Finnish Meteorological Institute (FMI) weather forecast service. The Resource Abstraction Interface is implemented as an oBIX Database with mappings to the BEMS and interfaces to receive weather and CO2 emissions data.

5.2.1.2 Supermarket

The supermarket contains many subsystems that have potential to be used as flexible assets. Currently only the heating system can be controlled.

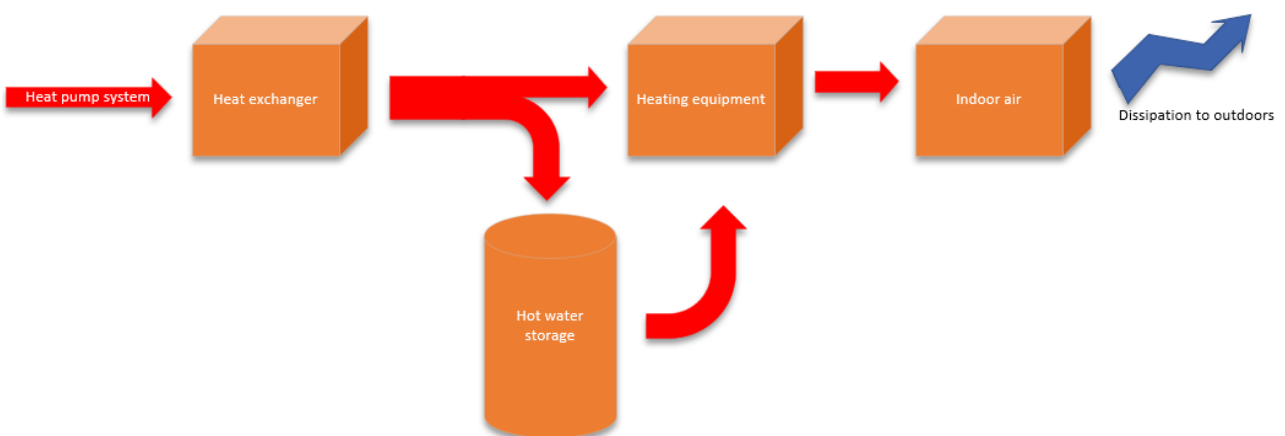


Figure 19. Simplified energy balance of the heating system.

The heating system can be controlled by adjusting incoming water temperature to the heating equipment. Water coming from the heat exchanger and hot water tank are mixed to form the heating water flow. We control the temperature of this flow to enable flexibility. In practice, we use two settings to control the water temperature:

- Nominal heating – The heating system aims to provide the amount of heat that the equipment are asking
- Restricted heating – We restrict all heat going to the heating equipment (The hot water storage will still be available to store extra heat from the heat pump)

Measurement data is collected from the pilot to a data aggregation platform. Data resolution for the measurement points is 1 minute. Key measurements include:

- Heat of the heat exchanger to the heating system
- Heat to the heating equipment
- Temperature of the hot water boiler (3 sensors)
- Temperature of indoor air exhaust flow
- Temperature of outdoor air
- 

The AFM instance used has similar architecture as shown in figure Figure 18. The only differences lie in the digital twin repository. In this repository there are models for electricity consumption (for producing heat) and indoor air temperature. These models are described in more detail in *D3.3*.

### 5.2.2 Instantiation in the Greek pilot

In the Greek pilot the iFLEX Assistants (and associated AFM components) are tailored for single family houses in three locations: Athens, Thessaloniki and Volos. The flexibility in these households is provided by white goods that can be controlled remotely. From AFM perspective, the primary focus is on water boilers that can be controlled via a relay. In Greece, it is common for the people to heat the water in their boiler just before they take the shower. This is done by manually controlling a relay, and the storage capacity of the boiler is not typically utilized. The approach for demand side flexibility is based on rescheduling the assets without compromising user preferences. Users define their personal preference for asset schedule and the AFM component utilizes this to plan optimal schedules. Figure 20 illustrates this process and depicts the user preferences and schedule allocated by the AFM component. It also highlights the up and down flexibilities calculated based on the user preferences.

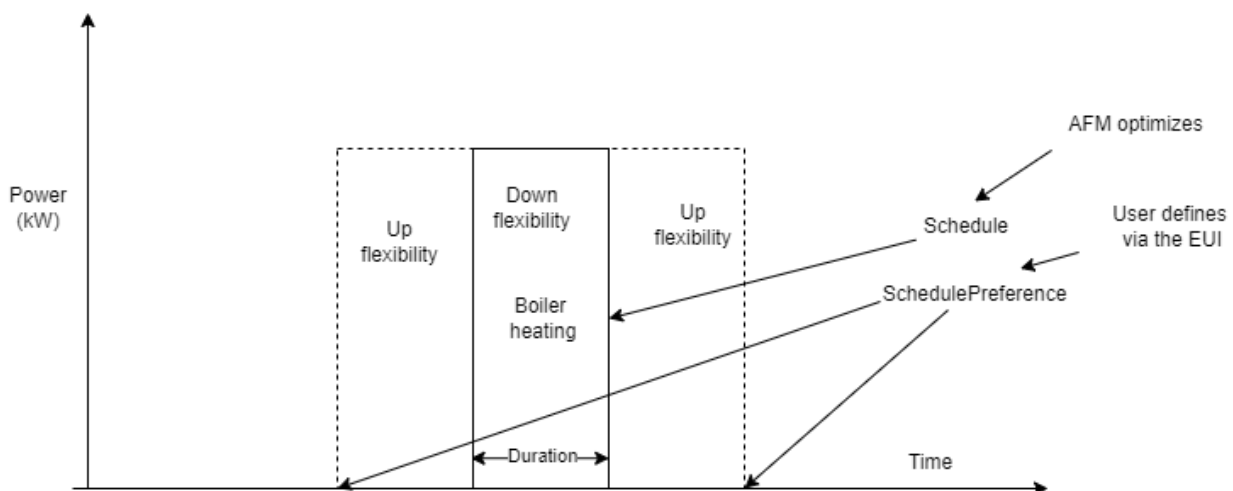


Figure 20: Illustration of the flexible assets scheduling based on user preferences.

Figure 21 illustrates a functional view of the automated flexibility manager component instantiated for the Finnish pilot. The AFM component is tailored based on the DTR component, which consists of three main models. These models are utilized by the AFM for planning and optimizing flexibility management. These three key models include Electricity baseline model, Boiler model, and Water demand model. The Electricity baseline model is trained to forecast the non-flexible electricity demand at the consumer premises. The Boiler Model represents the heating power and capacity of a boiler deployed in the consumer premises. The boiler electricity consumption is measured separately which models it possible to model and forecast as an independent resource. The Boiler model is designed to work together with a Water demand model. The Water demand model is designed to forecast the water consumption (in terms of energy) based on temporal data, consumer preferences and learned behaviour of the consumer.

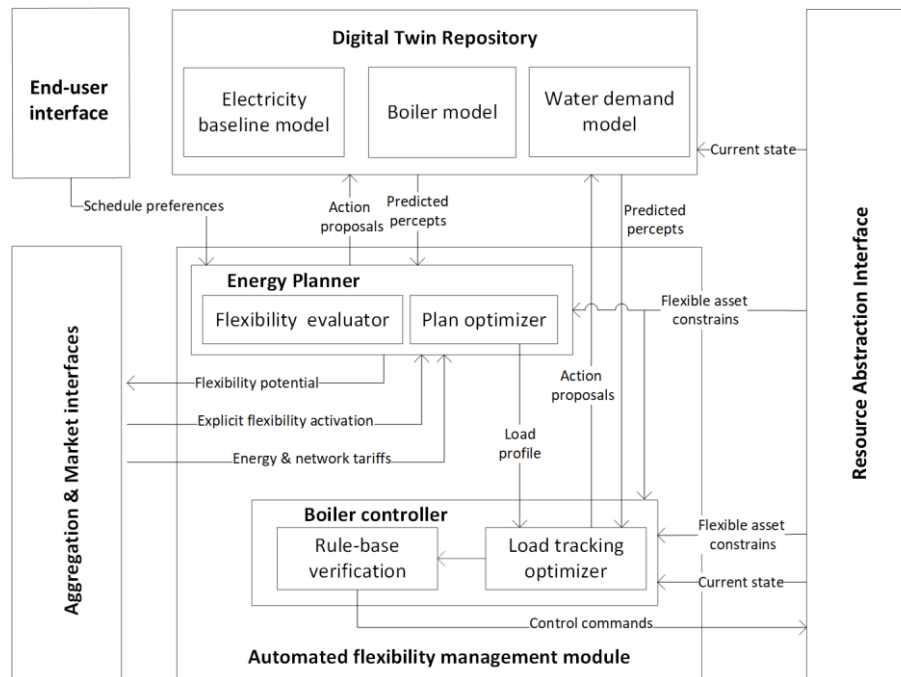


Figure 21: Functional view of the AFM instantiation in the Greek pilot.

The EnergyPlanner component is similar to the AFM instances in the other pilots. It utilises the Electricity baseline model to estimate the inflexible part of the baseline load. The flexible part of the baseline is provided by the Boiler model. The flexibility is calculated based on the user preferences, expected heating period, and expected heating power as illustrated in Figure 20. Only on/off control is utilized in the flexibility evaluation. It should be noted that in the Greek pilot there is no local optimisation or implicit DR incentives available. Therefore, the baseline load plan and associated schedule is implemented with a policy where the water heater is always heated as quickly as possible. If e.g., a spot price contract would be available the schedule would be optimized with respect to dynamic prices so that the costs are minimized.

As described in section 4.1, there is a single controller for each flexible asset. In this example there is only a controller for the water boiler. The water boiler controller is responsible for realising the load plan scheduled by the Energy Planner. In this case the control is very simple to implement as the Boiler controller only needs to decide whether 'On' or 'Off' command is sent to the RAI.

### 5.2.3 Instantiation in the Slovenian pilot

In the Slovenian pilot the iFLEX Assistant is tailored for individual households. The households in the pilot are in Celje region. The D7.3 Final Pilot Specification gives more details on household regions' properties. The primary source of flexibility in most households is a heating system. Some of the heating systems are installed together with a storage boiler. The boilers are of various capacities, some hold 900 or even up to 3000 litres of water. In addition to the boiler, the building envelope (thermal mass) is utilised as a source of flexibility besides household inhabitants' tolerance to change of indoor temperature. Some of the households in the pilot have available RES generation in a form of a PV power plant. More information on household installations is available in the D4.3 - Final Resource Abstraction Interface.

The AFM deployment for a typical site in the Slovenian pilot is presented in Figure 22. There are following three models in the DTR: Electricity baseline model, Heating system model and Indoor temperature model. These models have similar role as in the case of the Finnish pilot. Also, the heating controller is similar to the one utilised in the apartment building of the Finnish pilot. In the Slovenian pilot the control is executed by sending temperature set points to the thermostats in different rooms. The Energy Planner component is configured to support two types of use cases: explicit DR and self-optimization.

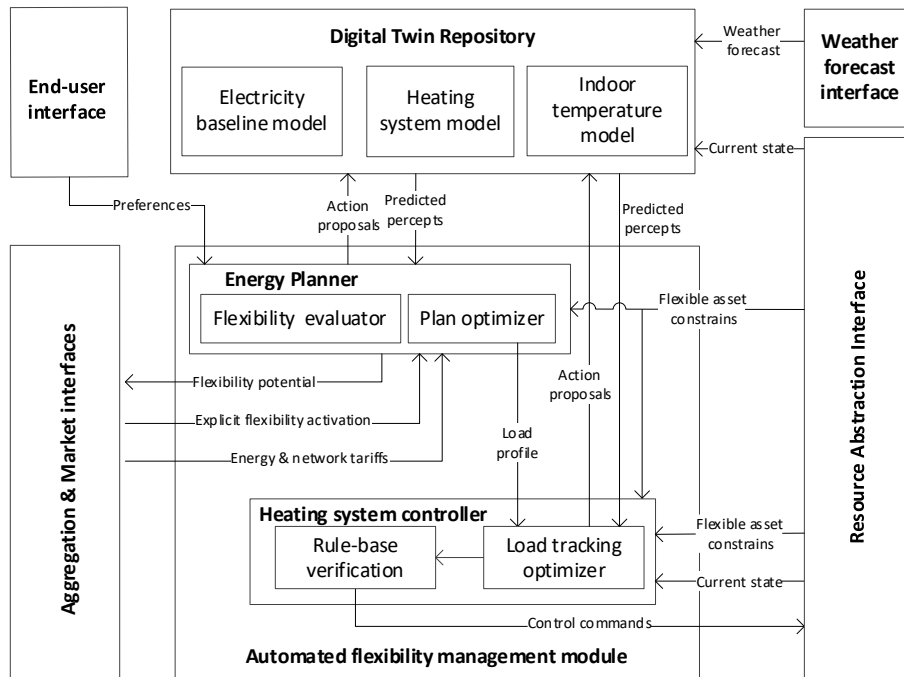


Figure 22: Functional view of the AFM instantiation in the Slovenian pilot.

In the explicit DR use case the Energy Planner utilizes the heating system models to estimate the baseline and flexibility and provide the Aggregator and Market Interface with these forecasts. The forecast are utilized by the Aggregation platform for more optimal bidding in the Slovenian Day-ahead and intra-day markets. I.e., the goal is to acquire the energy as cost-efficiently as possible by moving the consumption (flexibility) into the cheaper hours. Only the technical feasibility of this use case is evaluated in the pilot.

Self-optimisation will combine the household sources of flexibility either with the PV generation, if available at the household, or in combination with a tariff system to optimise the consumption according to the price of the energy. The self-optimisation in the case of storage boiler/building structure flexibility and the PV generation will look into utilising as much as possible of generated electricity for heating of the house. The optimisation will utilise a capacity of the storage to store the heat at the times of high generation. The heat will be realised when needed, usually in the afternoon and in the evening, when outside temperatures drop. A brief example of such scenario is presented in Figure 23. In the figure two days of consumption are presented. On the left hand side a non-optimized day in January is shown. The self-consumption (denoted in sky-blue) is very low. The production, in green, is not properly used. The consumption in red happens in the morning and in the afternoon when the PV generation is no longer active. In the figure on the right-hand side an optimised day of consumption in March is presented. The afternoon consumption is at large shifted to the time of generation. The generation is much better utilised and afternoon consumption is more uniform than before.

The self-optimisation case as described utilizes the following data in at least one hour granularity from a Digital twin repository, as is denoted in Figure 22:

- Electricity consumption prediction: based on electricity baseline model a prediction of the household electricity consumption is needed for next 24 hours,
- Heating demand prediction: based on heating system model a heating demand of the household needs to be provided for next 24 hours,
- Generation prediction: based on generation model a prediction of PV generation for next 24 hours are needed,
- Indoor temperature prediction: based on indoor temperature model an indoor temperature prediction for next 24 hours is needed,
- Weather prediction: weather prediction for next 24 hours is needed. The predictions are obtained through weather forecast interface. Typical weather parameters needed are temperature and radiation. Weather prediction is needed for other predictions and not for AFM operation directly.



Figure 23: Self-optimisation in Slovenian pilot example, household flexibility and PV generation

In a case of self-optimisation regarding the tariff changes the available flexibility is used to optimise the consumption regarding the price of energy. The tariffs can be either static network tariffs or dynamic electricity prices as are described in the deliverable the D4.6 - Final Market and Aggregation Interface Module. In any case the flexibility, either a storage boiler or household building structure thermal capacity, as described in D3.3., is used to consume the energy and store heat at the times when the energy prices are low and avoid to consume energy at the time when the prices are high. In the case of network tariffs, the consumption is scheduled over the night (between 22:00 and 6:00) and the heat is used over the day or even afternoons and evenings, when the tariffs are high.

In both cases of self-optimisations a type of flexibility available plays a crucial role in automated flexibility management implementation. In case of storage boiler the flexibility is more predictable and easier to exploit. The boiler can store the heat for a longer period of a time. The flexibility depends on capacity of the boiler. In the case when flexibility is obtained from household building thermal capacity the amount of flexibility can be smaller, the digital twin models could need some identification time before the model is accurate enough and the flexibility cannot be stored for a longer period of time. The automated flexibility management optimisations need to take into account that such flexibility cannot be retained for a longer period of time like in case of the storage boiler.



## 6 Conclusions

This deliverable documented the third phase prototype of the Automated Flexibility Management. The AFM is a core functional component of the iFLEX Assistant. It optimizes energy management within the consumer premises to maximize consumer benefits while offering the available flexibility to power grid and energy system balancing.

The AFM module is based in innovative approach that combines AI technologies with optimal control methods. The key idea in the approach is to apply model-based planning and control where ML is combined with physics-based modelling to learn accurate and robust models of the consumers, their premises, including flexible assets. The AI-based control algorithms such as the MCTS is then applied for finding optimal control policies.

The AFM control architecture consist of two types of components: single Energy Planner and one or more Controllers. The Energy Planner is responsible for maximizing consumer benefits by orchestrating the operation of individual Controllers. It plans the load profile for each Controller without a need to have detailed knowledge about the flexible assets. Each Controller is then responsible for following the load profile assigned by the Energy Planner.

The AFM prototype described in this deliverable is implemented with Python programming language. The implementation consists of four core classes, including the AFM, EnergyPlanner, Resource and MqttInterface. The AFM is the central class the aggregates all other classes to realize the AFM instance. The EnergyPlanner realizes the Energy Planner component of the AFM architecture. The Resource class implements the Controller component and also acts as an interface to the Digital Twin Repository utilized for model predictive control. Only single resource per consumer is supported in the current prototype of the AFM module. The AFM module provides MQTT based interface which is implemented by the MqttInterface class. MQTT based API was chosen as it provides lightweight communication protocol based on publish and subscribe communication pattern. The AFM module is tailored for three different types of pilot sites, located in Finland, Greek and Slovenia.

## List of figures and tables

### 6.1 Figures

Figure 1: AFM requirements updated in the project's Jira tool. ....	7
Figure 2: Functional view of the iFLEX assistant with Automated Flexibility Management module highlighted. ....	8
Figure 3: Functional view of the Automated Flexibility Management module. ....	10
Figure 4: Q-learning algorithm. ....	11
Figure 5: Model predictive control (MPC) method for model-based optimal control. ....	12
Figure 6: Monte Carlo Tree Search. Reprinted from Image from An Introduction: Reinforcement Learning by Sutton & Barto (Sutton & Barto, 2018) ....	13
Figure 7: UML class diagram for the Automated Flexibility Manager. ....	15
Figure 8: Sequence diagram of the flexibility offering process. ....	21
Figure 9: Sequence diagram depicting the implicit DR optimization. ....	21
Figure 10: Sequence diagram illustrating explicit DR signal and schedule optimization. ....	22
Figure 11: Sequence diagram illustrating the control process. ....	22
Figure 12: Examples of the MQTT topic instances. ....	23
Figure 13: Payload A: Baseline and flexibility forecast payload. ....	25
Figure 14: Payload B: Used in Flexibility activation messages. ....	26
Figure 15: Payload C: Used in Load measurement messages. ....	26
Figure 16: Example of the Message D, used in flexible asset schedule messages. ....	26
Figure 17: Simplified view of heating energy flows in the pilot building. Red arrows represent energy inflows while blue arrows indicate energy outflows. ....	27
Figure 18: Functional view of the AFM instantiation in the Finnish pilot. ....	28
Figure 19: Simplified energy balance of the heating system. ....	28
Figure 20: Illustration of the flexible assets scheduling based on user preferences. ....	29
Figure 21: Functional view of the AFM instantiation in the Greek pilot. ....	30
Figure 22: Functional view of the AFM instantiation in the Slovenian pilot. ....	31
Figure 23: Self-optimisation in Slovenian pilot example, household flexibility and PV generation. ....	32

### 6.2 Tables

Table 1. MQTT topics of AFM interface. ....	22
Table 2. Attributes of the Payload A header. ....	23
Table 3. Attributes of a JSON object literal contained in Payload A. List of these structures is stored in the data attribute of the message. ....	23
Table 4. Attributes of the Payload B. ....	25
Table 5. Attributes of the Message D. ....	26

## 7 References

- Bianchini, G., Casini, M., Vicino, A., & Zarrilli, D. (2016). Demand-response in building heating systems: A Model Predictive Control approach. *Applied Energy*, 168, 159-170. <https://doi.org/10.1016/j.apenergy.2016.01.088>
- Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. In *IEEE Transactions on Computational Intelligence and AI in Games*.
- Chen, Y., Norford, L. K., Samuelson, H. W., & Malkawi, A. (2018). Optimal control of HVAC and window systems for natural ventilation through reinforcement learning. *Energy and Buildings*. <https://doi.org/10.1016/j.enbuild.2018.03.051>
- Kiljander, J., Sarala, R., Rehu, J., Pakkala, D., Pääkkönen, P., Takalo-Mattila, J., & Känsälä, K. (2021). Intelligent consumer flexibility management with neural network based planning and control. *IEEE Access*, PP. <https://doi.org/10.1109/ACCESS.2021.3060871>
- Kocsis, L., & Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. In (pp. 282-293). Springer Berlin Heidelberg. [https://doi.org/10.1007/11871842\\_29](https://doi.org/10.1007/11871842_29)
- Kusiak, A., Xu, G., & Zhang, Z. (2014). Minimization of energy consumption in HVAC systems with data-driven models and an interior-point method. *Energy Conversion and Management*. <https://doi.org/10.1016/j.enconman.2014.05.053>
- Ma, Z., & Wang, S. (2011). Supervisory and optimal control of central chiller plants using simplified adaptive models and genetic algorithm. *Applied Energy*, 88(1), 198-211. <https://doi.org/10.1016/j.apenergy.2010.07.036>
- Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. 30th International Conference on Machine Learning, ICML 2013,
- Patyn, C., Ruelens, F., & Deconinck, G. (2018). Comparing neural architectures for demand response through model-free reinforcement learning for heat pump control. 2018 IEEE International Energy Conference, ENERGYCON 2018,
- Ruelens, F., Claessens, B. J., Vandael, S., De Schutter, B., Babuska, R., & Belmans, R. (2017). Residential Demand Response of Thermostatically Controlled Loads Using Batch Reinforcement Learning. *IEEE Transactions on Smart Grid*, 8(5), 2149-2159. <https://doi.org/10.1109/tsq.2016.2517211>
- Ruelens, F., Iacovella, S., Claessens, B. J., & Belmans, R. (2015). Learning agent for a heat-pump thermostat with a set-back strategy using model-free reinforcement learning. *Energies*. <https://doi.org/10.3390/en8088300>
- Silver, D. (2016). AlphaGo. *Nature*. <https://doi.org/10.1038/nature16961>
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*. <https://doi.org/10.1126/science.aar6404>
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second ed.). The MIT Press. <http://incompleteideas.net/book/the-book-2nd.html>
- Tang, F., & Xu, G. (2011). Multi-objective optimization of HVAC system with an evolutionary computation algorithm. *Fuel and Energy Abstracts*, 36. <https://doi.org/10.1016/j.energy.2011.01.030>
- West, S. R., Ward, J. K., & Wall, J. (2014). Trial results from a model predictive control and optimisation system for commercial building HVAC. *Energy and Buildings*, 72, 271-279. <https://doi.org/10.1016/j.enbuild.2013.12.037>

## 8 Appendix: Jira requirements

### 8.1.1 [IF-72] [FN-AFM-04 Optimize flexibility based on prices \(e.g. implicit demand response\)](#) Created: 15/Jun/21 Updated: 09/Oct/23

<b>Status:</b>	IMPLEMENTED
<b>Project:</b>	<a href="#">iFlex Project</a>
<b>Component/s:</b>	None
<b>Affects Version/s:</b>	None
<b>Fix Version/s:</b>	None

<b>Type:</b>	Functional	<b>Priority:</b>	Major
<b>Reporter:</b>	<a href="#">Jussi Kiljander</a>	<b>Assignee:</b>	<a href="#">Jussi Kiljander</a>
<b>Resolution:</b>	Unresolved	<b>Votes:</b>	0
<b>Labels:</b>	AFM		

<b>Rationale:</b>	iFLEX Assistant needs to be able to optimize flexible assets schedule and/or setpoints in order to reduce end-user costs. This can include energy efficiency improvements and spot-price optimization
<b>Pilot Finland:</b>	Phase two
<b>Pilot Greece:</b>	Not applicable
<b>Pilot Slovenia:</b>	Phase three

#### Description

The automated flexibility management module should provide mechanisms to optimize flexibility with respect to dynamic tariffs, possible across different energy vectors.

**8.1.2 [IF-114] FN-AFM-06 Provide schedule information for the End-user Interface** Created: 09/Oct/23 Updated: 09/Oct/23

<b>Status:</b>	IMPLEMENTED
<b>Project:</b>	<a href="#">iFlex Project</a>
<b>Component/s:</b>	None
<b>Affects Version/s:</b>	None
<b>Fix Version/s:</b>	None

<b>Type:</b>	Functional	<b>Priority:</b>	Major
<b>Reporter:</b>	<a href="#">Jussi Kiljander</a>	<b>Assignee:</b>	<a href="#">Jussi Kiljander</a>
<b>Resolution:</b>	Unresolved	<b>Votes:</b>	0
<b>Labels:</b>	AFM		

<b>Rationale:</b>	With resources such as the boilers in Greek pilot it is important to provide the user with a schedule.
<b>Pilot Finland:</b>	Not applicable
<b>Pilot Greece:</b>	Not applicable
<b>Pilot Slovenia:</b>	Phase two

#### Description

AFM needs to publish a schedule for each resource (set points resources at different times) via its MQTT interface.

### 8.1.3 [IF-107] [FN-AFM-05 Optimize flexibility locally \(self-consumption, consumer load reduction\)](#) Created: 30/Jul/22 Updated: 14/Apr/23 Resolved: 14/Apr/23

<b>Status:</b>	Validated
<b>Project:</b>	<a href="#">iFlex Project</a>
<b>Component/s:</b>	None
<b>Affects Version/s:</b>	None
<b>Fix Version/s:</b>	None

<b>Type:</b>	Functional	<b>Priority:</b>	Major
<b>Reporter:</b>	<a href="#">Jussi Kiliander</a>	<b>Assignee:</b>	<a href="#">Jussi Kiliander</a>
<b>Resolution:</b>	Validated	<b>Votes:</b>	0
<b>Labels:</b>	AFM		

<b>Rationale:</b>	The requirement comes mainly from the needs of the PUC 10: Increase self-balancing through forecasting and automation.
<b>Pilot Finland:</b>	Phase two
<b>Pilot Greece:</b>	Phase two
<b>Pilot Slovenia:</b>	Phase two

#### Description

The AFM needs to support energy management policies for local control. This includes energy efficiency improvements (e.g. for cost and CO2 emission reduction) as well as self-consumption maximization (Slovenian pilot).

**8.1.4 [IF-70] [FN-AFM-02 Flexibility potential](#)** Created: 15/Jun/21 Updated: 14/Feb/22 Resolved: 14/Feb/22

<b>Status:</b>	Validated
<b>Project:</b>	<a href="#">iFlex Project</a>
<b>Component/s:</b>	None
<b>Affects Version/s:</b>	None
<b>Fix Version/s:</b>	None

<b>Type:</b>	Functional	<b>Priority:</b>	Major
<b>Reporter:</b>	<a href="#">Jussi Kiliander</a>	<b>Assignee:</b>	<a href="#">Jussi Kiliander</a>
<b>Resolution:</b>	Validated	<b>Votes:</b>	0
<b>Labels:</b>	AFM		

<b>Rationale:</b>	Flexibility potential data is needed to provide more deterministic explicit demand response.
<b>Pilot Finland:</b>	Phase one
<b>Pilot Greece:</b>	Phase two
<b>Pilot Slovenia:</b>	Phase two

**Description**

The automated flexibility management module should provide flexibility potential information to the A&M Interface module. The flexibility potential should be calculated by comparing the minimum and maximum loads to the baseline load profile at different time periods.

The length of the flexibility window and the frequency should be configurable.

### 8.1.5 [IF-71] [FN-AFM-03 Activate offered flexibility](#) Created: 15/Jun/21 Updated: 14/Feb/22 Resolved: 14/Feb/22

<b>Status:</b>	Validated
<b>Project:</b>	<a href="#">iFlex Project</a>
<b>Component/s:</b>	None
<b>Affects Version/s:</b>	None
<b>Fix Version/s:</b>	None

<b>Type:</b>	Functional	<b>Priority:</b>	Major
<b>Reporter:</b>	<a href="#">Jussi Kiljander</a>	<b>Assignee:</b>	<a href="#">Jussi Kiljander</a>
<b>Resolution:</b>	Validated	<b>Votes:</b>	0
<b>Labels:</b>	AFM		

<b>Rationale:</b>	Flexibility activation is required in implicit and explicit demand response.
<b>Pilot Finland:</b>	Phase one
<b>Pilot Greece:</b>	Phase two
<b>Pilot Slovenia:</b>	Phase two

#### Description

The automated flexibility management module should activate the offered flexibilities when requested via the A&F interface (assumes flexibility activation is authorized by the end-user).

To activate the flexibility the AFM module needs to find an optimal control schedule by optimizing with the model provided by the digital twin repository. Once optimal schedule is found the flexibility is activated by modifying the local energy management system parameters.

### 8.1.6 [IF-69] [FN-AFM-01 Provide baseline forecasts](#) Created: 15/Jun/21 Updated: 14/Feb/22 Resolved: 14/Feb/22

<b>Status:</b>	Validated
<b>Project:</b>	<a href="#">iFlex Project</a>
<b>Component/s:</b>	None
<b>Affects Version/s:</b>	None
<b>Fix Version/s:</b>	None

<b>Type:</b>	Functional	<b>Priority:</b>	Major
<b>Reporter:</b>	<a href="#">Jussi Kiljander</a>	<b>Assignee:</b>	<a href="#">Jussi Kiljander</a>
<b>Resolution:</b>	Validated	<b>Votes:</b>	0
<b>Labels:</b>	AFM		

<b>Rationale:</b>	Baseline load profile is required to estimate and validate the flexibility at individual prosumer level. Prosumer/consumer specific baseline load profiles can be also used to calculate aggregated load profiles at different levels of the power system.
<b>Pilot Finland:</b>	Phase one
<b>Pilot Greece:</b>	Phase two
<b>Pilot Slovenia:</b>	Phase two

#### Description



The automated flexibility management module should provide information about the baseline load profile (including consumption and local production) of the prosumer/consumer.