Intelligent Assistants for Flexibility Management (Grant Agreement No 957670)

D6.4 Final Common iFLEX Framework

Date: 2023-12-21

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: D6.4 Final Common iFLEX Framework 1.0.docx
Document version: **1.0**
Document owner: **VTT**

Work package: WP6 System integration and service packaging
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	Anne Immonen (VTT)	2023-11-20	Structure and introduction updated.
0.2	Nikos Charitos (ICOM)	2023-12-06	Updated descriptions of Aggregator and Market interface, and End-user interface.
0.3	Jussi Kiljander (VTT), Arso Savanovic (SCOM)	2023-12-08	Updated description of Automated Flexibility Management, Digital Twin Repository and Weather Service Interface, Updated section on plan and tools for continuous integration
0.4	Anne Immonen (VTT)	2023-12-18	Version prepared for internal review
1.0	Anne Immonen (VTT)	2023-12-21	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Siiri Lapila (Caverion)	2023-12-19	Accepted with minor corrections and comments.
Roman Tomažič (ZPS)	2023-12-20	Accepted with minor corrections and comments.

Index:

List of abbreviations

Abbreviation	Term
A&M (Interface)	Aggregator and Market (Interface)
ADR	Automated Demand Response
AFM	Automated Flexibility Manager
AMI	Advanced Metering Infrastructure
ANN	Artificial Neural Network
API	Application Programming Interface
BEMS	Building Energy Management System
BRP	Balance Responsible Party
DB	Database
DH	District Heat
DR	Demand Response
DRMS	Demand Response Management System (from OpenADR terminology)
DT	Digital Twin
ENTSO-E	European Network of Transmission System Operators for Electricity
FOI	Flexibility Operator Interface
FMI	Finnish Meteorological Institute
HEMS	Home Energy Management System
iFA	iFLEX Assistant
MQTT	Message Queuing Telemetry Transport
OASIS	Organization for the Advancement of Structured Information Standards
oBIX	open Building Information Xchange
RAI	Resource Abstraction Interface
REST API	Representational State Transfer Application Programming Interface
SaaS	Software as a Service
UI	User Interface

1 Executive summary

This document describes the final version of the common iFLEX Framework. Task 6.2 *Integration of submodules into a common software framework* is responsible for the integration of the components developed in the technical Work Packages into the common iFLEX software framework. It is based on Task 6.1 *Continuous integration and deployment planning* that defines the continuous integration (and deployment plan) and mechanisms, and Task 2.3 *iFLEX Framework architecture design* that specifies the iFLEX Assistant system architecture. As D2.3 provides the general document for the said integration by defining the component interfaces at the logical level, T6.2 executes the co-development and integration at the practical level (i.e., software implementation and packaging). Thus, the common iFLEX software framework provides the libraries, scripts and tools for creating application-specific iFLEX Assistant instances.

The development of the common iFLEX framework is a continuous and iterative process executed in three distinct phases. This document describes the high-level summary of the third and final phase implementation of the seven functional components that construct the iFLEX framework, as follows: **Aggregator and Market Interface, End-user Interface, Resource Abstraction Interface, Automated Flexibility Management, Digital Twin Repository, Weather Service and Security and Privacy Interface**. Aggregator and market interface and End-user interface enable the interaction between the relevant actors and the iFLEX Assistant, whereas Resource abstraction interface enables the iFLEX Assistant to monitor and control relevant aspects in the consumer/prosumer premises. Automated flexibility management provides services for implementing demand-response control and optimization operations. Digital twin repository estimates the impact of a consumer and provides optimal and consumer-centric flexibility management. Weather service provides country-specific data about weather forecasts and, finally, Security and privacy interface ensures trust and security among all elements of the iFLEX framework.

For household-specific iFLEX Assistants (Slovenia and Greek pilots), the implementation concentrates on the Resource abstraction interface to enable data collection and control of flexible assets via smart meters and Home Energy Management Systems (HEMS), and on mock-up implementation of the End-user Interface to collect feedback on the design from pilot participants and allow co-creation of new functionalities. For iFLEX Assistant targeted to apartment buildings (Finnish pilot), the implementation focuses on data collection, flexibility management experiments and testing. Also, the goal is to experiment with the Automated flexibility management and associated Digital twin repository modules, and to deploy and collect feedback via an End-user interface designed for residents of an apartment building. The developed solutions are different in some parts of the different pilots.

2 Introduction

2.1 Purpose, context and scope

This is a demonstrator-type deliverable that documents the final version of the iFLEX Framework. The iFLEX Framework is a software framework for the development of intelligent assistants for flexibility and holistic energy management. It consists of the following modules: Aggregator and Market Interface, End-user Interface, Resource Abstraction Interface, Automated Flexibility Management, Digital twin repository, Weather service and Security and privacy interface. The iFLEX Assistant is an application-specific instance of an intelligent assistant developed on top of the common iFLEX Framework.

The iFLEX Framework development is an iterative process implemented in three phases:

- Phase one: A pre-pilot with the Minimum Viable Product of the iFLEX Framework and Assistants has been carried out with selected users (this phase has already been concluded).
- Phase two: A small-scale pilot including the iFLEX Framework with full functionality will be validated with small-scale pilot groups.
- Phase three: The improved iFLEX Framework is deployed and validated in large-scale pilots.

This document describes the final common iFLEX Framework in Phase three. Compared to the initial framework, the main updates of the phase two include the fully revision of the implementation and documentation of the modules to match the second phase status, whereas the phase three validates and refines the implementations with the help of real-world pilots.

2.2 Content and structure

This document is structured as follows. Section 3 describes how the iFLEX Framework is related to use cases and iFLEX architecture, and then the focus of the third implementation phase is introduced. Section 4 introduces the implementation of the main components of the iFLEX Framework. Concluding remarks are provided in Section 5.

2.3 Main changes compared to the previous version

- Section 4.8 has been added to describe the integration and deployment tools for iFLEX Assistants.
- As this deliverable describes the iFLEX Framework implementation at a high abstraction level (details are provided in component specific deliverables) there are no major updates to the individual component descriptions. Nevertheless, each component description has been updated with the latest information.

3 Overview

3.1 Relation to use cases

High-level requirements for the iFLEX Assistants, specified in D2.1 *Use cases and requirements,* are documented in the form of use cases. Based on these use cases, component-specific requirements have been engineered in the project. These requirements document the functional and non-functional needs that the iFLEX Assistants developed on top of the iFLEX Framework aim to satisfy. The requirements are collected and managed via the project's issue and project tracking tool, Jira. There are a total of 112 requirements that have been documented Jira. Figure 1 illustrates a snapshot of the requirements collected in Jira.

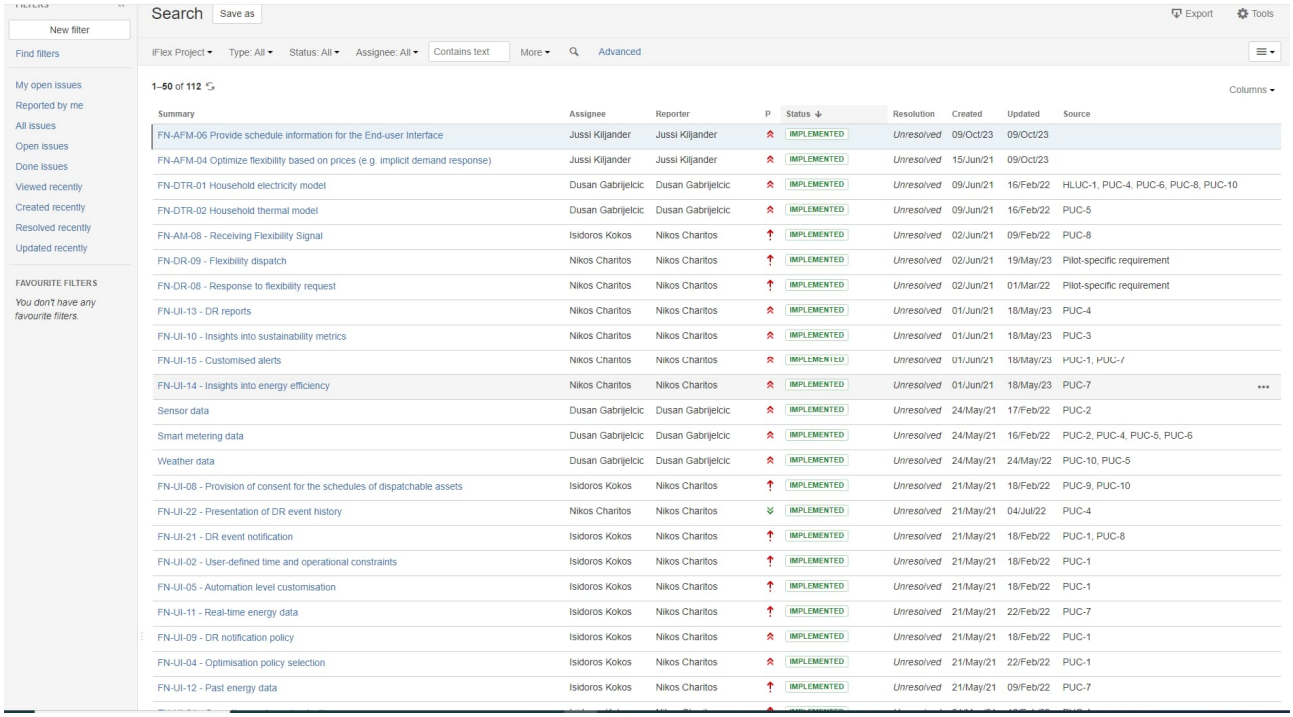

Figure 1: Snapshot of the requirements in Jira.

3.2 Relation to the functional architecture of the iFLEX Framework

This deliverable describes the third phase implementation of the iFLEX Framework and thus covers the whole architecture of the iFLEX Framework, as depicted in Figure 2. This deliverable is a summary of the third phase implementation, and it should be used together with the D2.5, which provides a more detailed description of the iFLEX Framework Architecture.

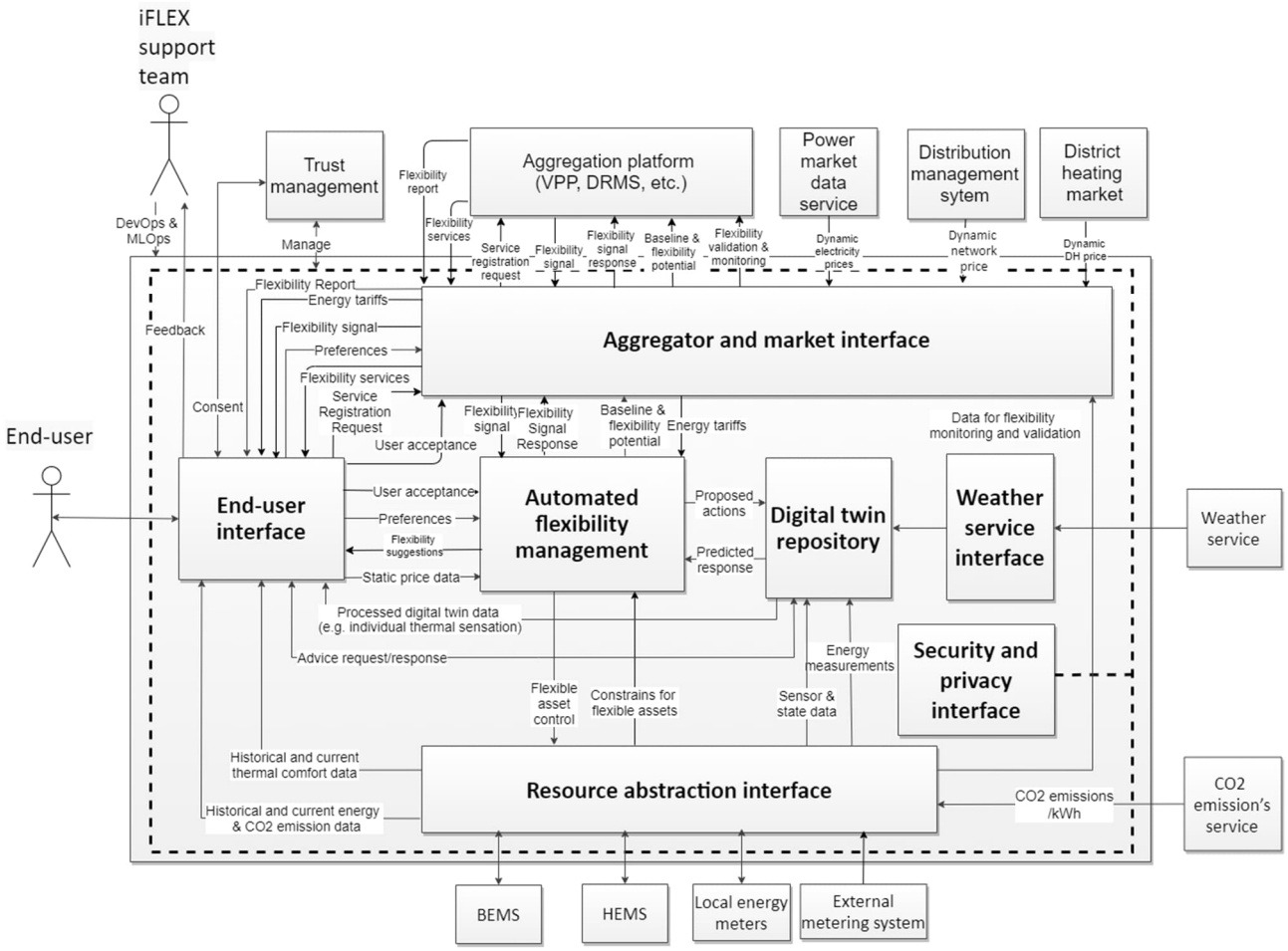

Figure 2: Functional view of the iFLEX Framework.

The iFLEX Framework consists of seven (7) functional components as illustrated in Figure 2. The implementation of these components is documented in this deliverable as follows:

- Section 4.1 introduces 3rd phase implementation of the Aggregator and market interface module.
- Section 4.2 describes the End-user interface module.
- Section 4.3 documents the Resource abstraction interface module.
- Section 4.4 introduces the Automated flexibility management module.
- Section 4.5 summarizes the Digital twin repository module.
- Section 4.6 describes the Weather service interface module.
- Section 4.7 summarizes the Security and privacy interface module.

3.3 Third phase focus

The focus in the third phase implementation of the iFLEX Framework is to provide tools and libraries for developing the iFLEX Assistants for the Finnish, Greek and Slovenian pilots during the large-scale piloting phase.

For household specific iFLEX Assistants, to be deployed in the Slovenian and Greek pilots, the required functionality focuses on four aspects: First, a Resource abstraction interface to enable data collection and control of flexible assets via relays, smart plugs, and Home Energy Management Systems (HEMS). Second,

the large-scale implementation of the End-User Interface, to collect feedback on the design from the participants and allow co-creation of new functionalities with them. Third, the AFM and Digital Twin repository modules to forecast the flexibility and execute automated DR actions. Fourth, the integration with the aggregation platforms provided by ICOM and SCOM.

The third phase implementation of the iFLEX Assistant targeted for apartment buildings (to be deployed in Finnish pilot) focuses on data collection and flexibility management experiments and testing. Additionally, the goal of the large-scale pilot in Finland is to experiment and demonstrate a fully functional iFLEX Assistant in the context of the apartment building flexibility management. The iFLEX Assistant includes all the functional components documented in the Final iFLEX Architecture (see D2.5 for details).

To this end, the third phase focuses on extending and integrating the components in order to enable the aforementioned functionality to be deployed and tested in the large-scale pilots. Moreover, the focus has been on implementing the Aggregator and market interface modules to support the full use case to be demonstrated in the large-scale pilots during Phase 3 of the project.

4 Implementation for phase three

The main functional components of the common iFLEX Framework and their interactions are depicted in Figure 3 Figure 3 and described in more detail in the following sub-sections. For each component, the functional description and the description of implementation are provided.

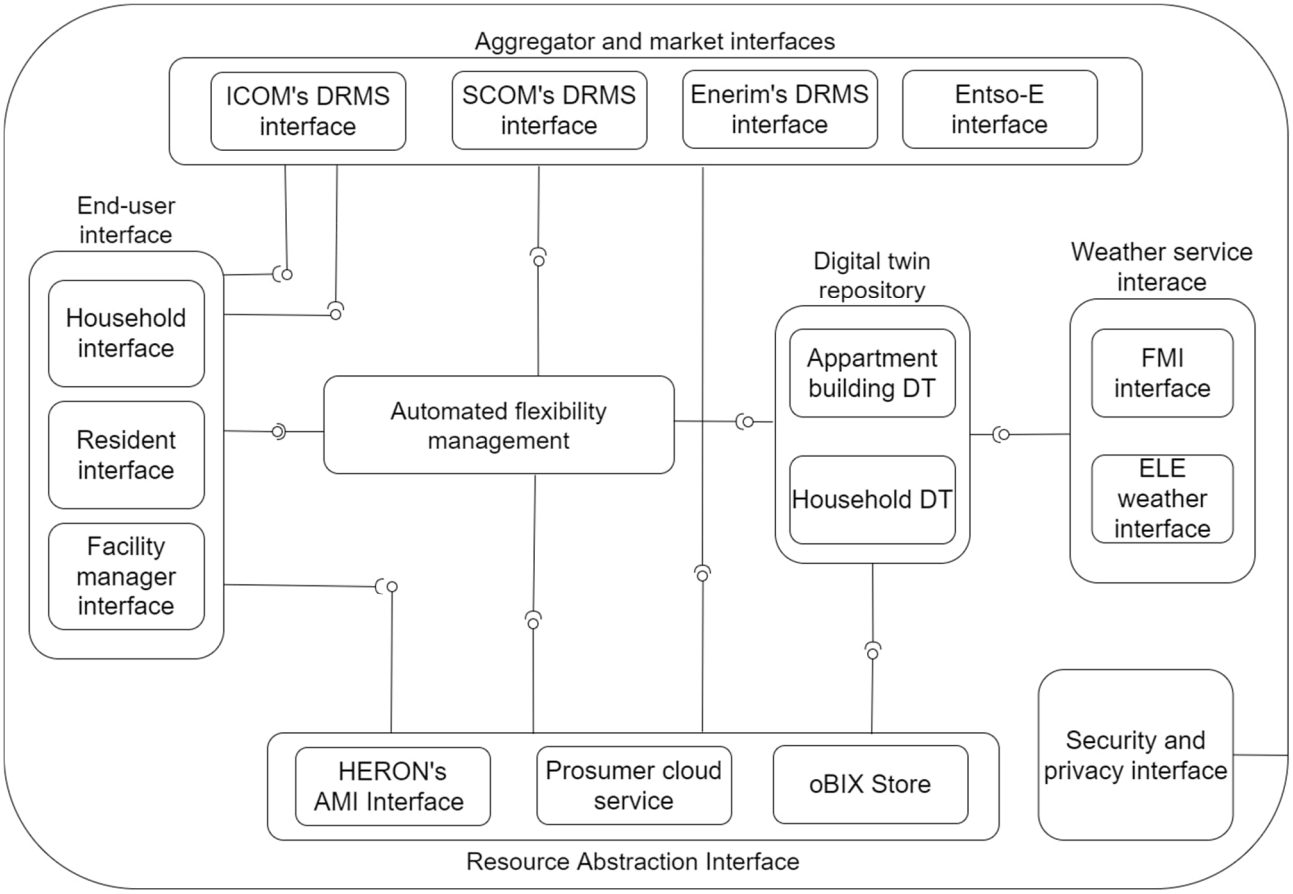

Figure 3: Overview of the iFLEX Framework.

4.1 Aggregator and market interface

The Aggregator and Market Interface (A&M) is the component of the iFA, which enables the participation of the iFA end users in flexibility services via interfacing the iFA with Aggregation Platforms. Furthermore, it collects energy tariff data from energy markets' APIs. The A&M Interface component should support the following functionalities by the end of Phase 3:

- Communicate the flexibility potential of the iFA end user to the Aggregation Platform;
- Receive flexibility signals from the Aggregation Platform and forward them to relevant iFLEX components;

Particularly in case of explicit DR events, the A&M component should also be able to:

- Send the response to the flexibility signal back to the Aggregation Platform;
- Receive flexibility validation data from the RAI and send them to the Aggregation Platform;
- Evaluate the iFA end user's participation and communicate the result of the assessment to the Aggregation Platform;
- Receive a DR report from the Aggregation Platform and expose its content to the iFA end user.

Furthermore, it should be equipped with the ability to receive electricity network and retail tariffs, as well as district heating tariffs, and expose them to relevant iFLEX components.

Two versions of the A&M Interface component are developed: one for the residential iFA and one for the building community. The residential A&M module will host various back-end services, which are built exploiting the Django Web Framework (Django Software Foundation, 2022). Furthermore, a DR client – also known as Virtual End Node (VEN) according to the OpenADR (OpenADR Alliance, 07-01-2013) terminology – will be hosted in the A&M Interface component. This sub-component is responsible for the communications with the DR server of the Aggregation Platform, and is built utilising the aiohttp (AIOHTTP, 2022) and OpenLEADR (OpenLEADR, 2022) packages of Python. Moreover, cron jobs are used in order to schedule periodically executed tasks, such as receiving the latest energy tariffs. All the above sub-components read from and write to a remote PostgreSQL (PostgreSQL, 2022) database. In Figure 4, the technology stack of the A&M Interface module for the residential iFA is demonstrated.

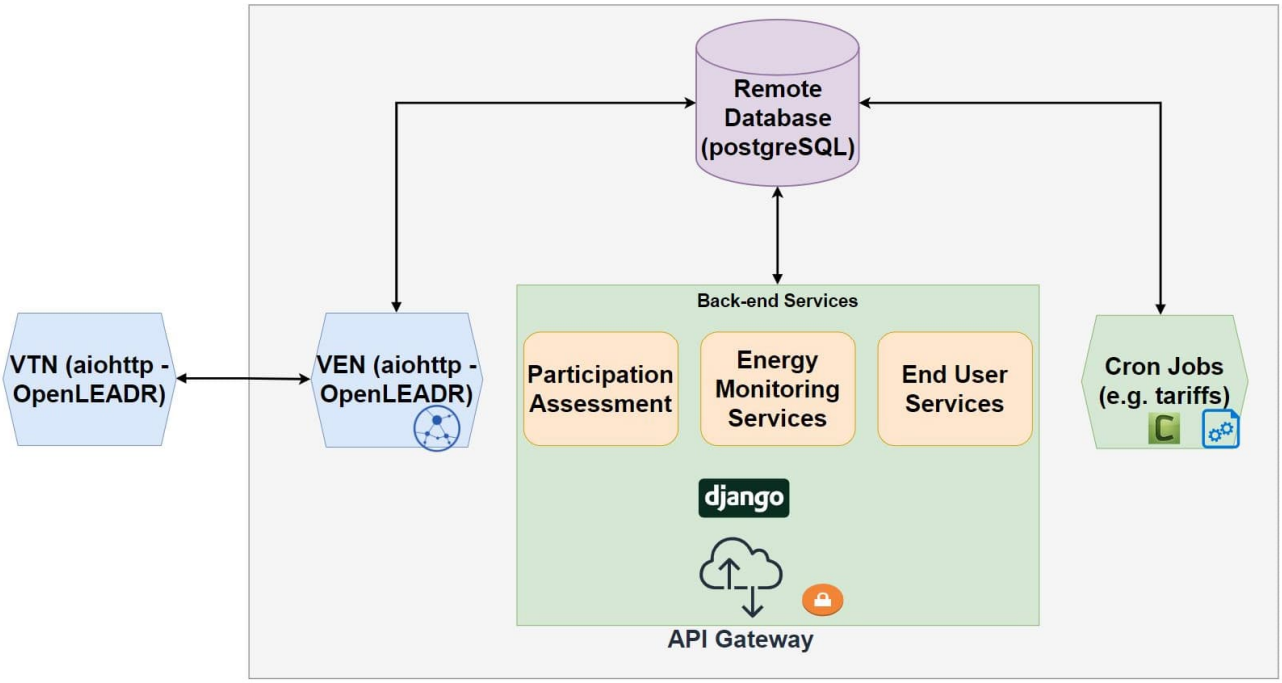

Figure 4: Technology stack of A&M's Interface module for the residential iFA

More information on the features of the Aggregator and Market Interface can be found in the dedicated deliverable, namely D4.6 *Final Market and Aggregation Interface Module* (iFLEX project, 2023). The same deliverable documents also the Aggregation Platforms – as deployed per pilot – which are briefly described in the following sub-section.

4.1.1 Aggregation Platforms

The iFA has to be interfaced with an Aggregation Platform in each pilot, so that its full set of functionalities can be deployed and tested. Their interactions are shown in Figure 5. Therefore, the Aggregation Platforms should support the following features:

- Receive the flexibility potential of the iFA end users from the A&M Interface;
- Send flexibility signals to the iFA and receive the respective responses;
- Receive flexibility validation data from the iFA;
- Send DR reports to the iFA;
- Receive flexibility requests from third parties (e.g. a DER Aggregator, etc.), dispatch DR events based on these requests, and report back to them with respect to the finally activated flexibility.

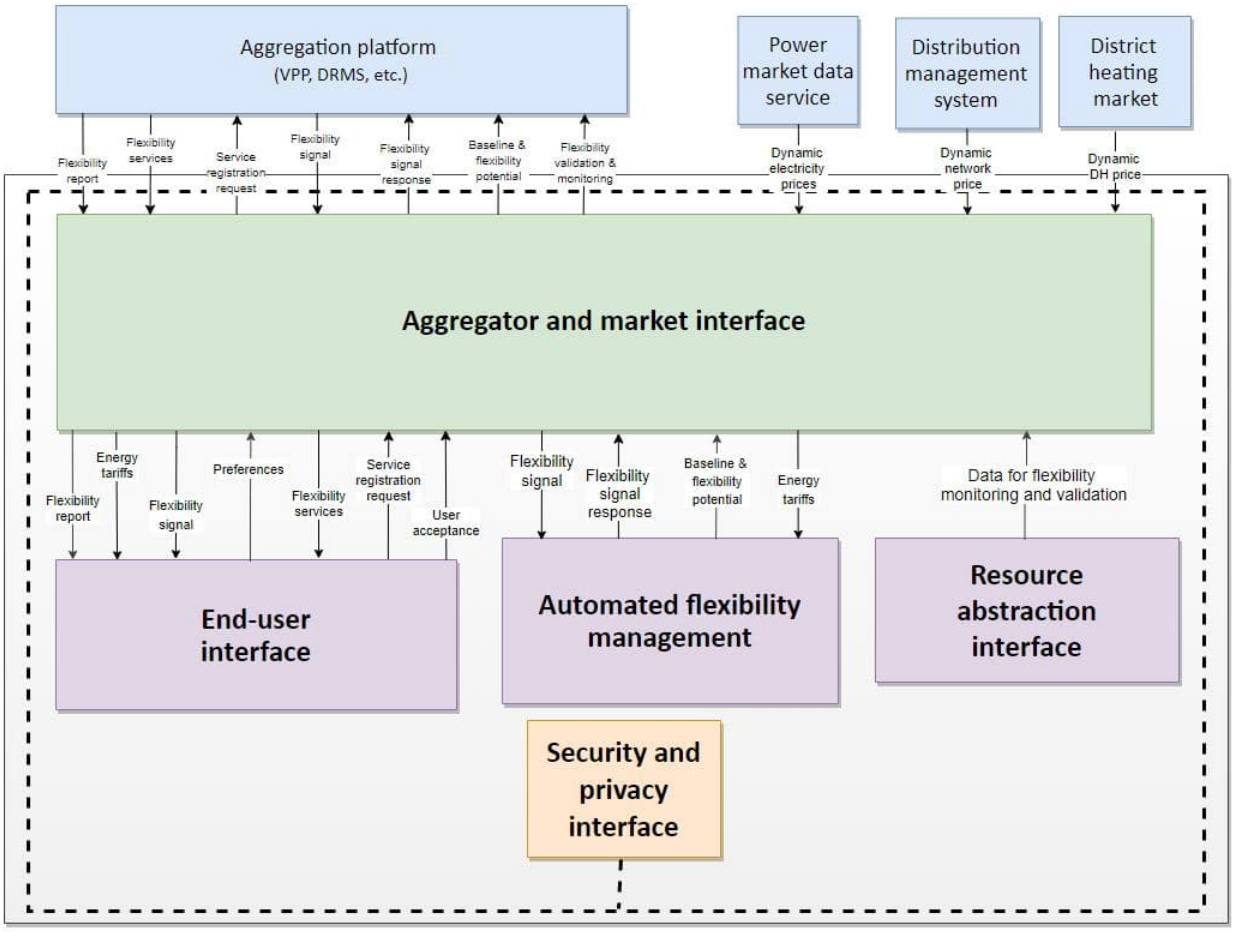

Figure 5: Context view of the A&M Interface Module

The three solutions utilized in the iFLEX pilots are: ICOM's Demand Response Management System (DRMS) in Greece, SCOM's DRMS in Slovenia, and Enerim's Aggregation Platform in Finland. More details on the features and employed technologies of each one of them are documented in the relevant deliverable D4.6 *Final Market and Aggregation Interface Module* (iFLEX project, 2023).

4.2 End-user interface

The User Interface (UI) component enables the interaction of the end users with their iFAs. Through the developed applications, the users can both set their preferences on the operation of iFA and observe its impact on the energy and flexibility management of their premises.

These applications should equip their users with the following capabilities by the end of the project:

- Monitor energy flows, namely both electricity consumption and – when applicable – electricity generation within premises (self-generation);
- Opt for the desired level of automation by activating or deactivating the automated mode, and by setting up scheduled and flexible operation modes for their flexible assets – providing time and operational constraints;
- Choose the objectives upon which the optimisation of the household's energy use will be based;
- Pause push notifications – either temporarily or periodically;
- Receive DR event notifications, and accept or reject the iFLEX-proposed schedules for their flexible assets;
- Receive DR reports and access DR participation history;

- Submit feedback on their comfort level;
- Observe energy efficiency and sustainability metrics, and track the users' personalised relevant goals via customised alerts;
- Receive tailored advice, which aims at improving the household's energy performance;
- View energy tariffs, as well as an estimation of future energy costs;
- View an estimation of the users' savings thanks to the operation of the iFA;
- View the operational schedules of assets online and offline;
- Submit feedback on users' experience while using the app;
- Choose the system interface language between English and the users' native language;
- View an app walkthrough, which explains to the users the functionality of each page of the app, whenever they want.

Depending on applicability, the availability of the functionalities described above varies per pilot.

Two versions of the UI are developed; one is tailored to residential end users (which is the case for the Greek and Slovenian pilots), while the other is focused on building communities, which is the case for the Finnish pilot of the project. This UI version is a web application written with JavaScript[1], HTML[2], SVG[3], and CSS[4]. The application contains a backend and a frontend. The backend is written with Node.js (Node.js documentation, 2022) and stores user data into a MongoDB (MongoDB: The Developer Data Platform, 2022) database. As regards the UI for the residential end users, it is further divided into two instantiations: a hybrid mobile application, which is deployed in the Slovenian pilot, and a web application, which is exposed through the pre-existing Heron's mobile application in the Greek pilot. The UI backend server of both applications for residential end users is based on the Django web framework and a PostgreSQL database is exploited to achieve data persistence. The frontend of the web application – deployed in Greece – utilises HTML, CSS, and various JavaScript libraries, such as React (React: A Javascript library for building user interfaces, 2022). On the other hand, the hybrid mobile application – deployed in Slovenia – is built using the React Native framework (React Native, 2022).

More information on the functionalities, employed technologies, and design of the UI components are provided in D3.6 *Final Natural User Interfaces* (iFLEX project, 2023). Indicatively, Figure 6 shows the "Dashboard" screen of the mobile app for residential end users in Slovenia, while Figure 7 presents the front page of the UI for the building community in Finland.

[1] About JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
[2] What is HTML?: https://www.w3schools.com/html/html_intro.asp
[3] What is SVG?: https://www.w3schools.com/graphics/svg_intro.asp
[4] What is CSS?: https://www.w3schools.com/css/css_intro.asp

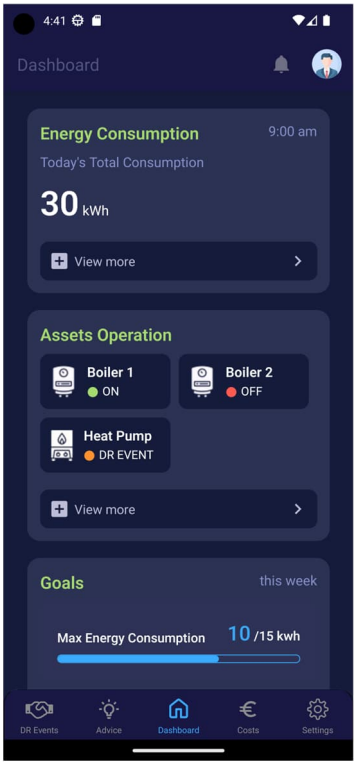

Figure 6: Mobile app for residential end users in Slovenia – "Dashboard" screen.

Figure 7: Building Community UI in Finland – The Front Page: Building Electricity and District Heating.

4.3 Resource abstraction interface

The Resource abstraction interface (RAI) component provides interfaces for home energy management systems (HEMS), building energy management systems (BEMS), smart meters, and other external metering/sensor infrastructure. That is, the role of the RAI is to provide iFLEX Assistant with means to monitor and control energy consumption, comfort, and other relevant aspects in the consumer/prosumer premises.

The current RAI implementation consists of two interface stacks. The first stack is implemented on top of an oBIX store in the Finnish pilot. The second stack is being developed on the top of HERON's southbound HEMS interface in the Greek pilot and ECE/Amibit southbound HEMS interface in the Slovenian pilot.

The oBIX Store module provides means to access data and control resources provided by BEMS via standard Open Building Information Xchange format[5]. It is based on existing baseline implementation hosted by VTT. The oBIX Store is implemented with Java. The iFLEX RAI interface for buildings has been implemented as a programmable client in Python programming language. The backend server is implemented in Javascript using Node.js on top of MongoDB. The backend server wraps the oBIX Store functionality and exports it through the Node.js in a form of REST API.

The Greek and Slovenian pilot share the same RAI implementation. In the Greek pilot the RAI is implemented on top of HERON's HEMS API. The HERON's API interface provides means both for real-time monitoring of consumer loads and for remote control of energy assets at the consumer premises. In particular, HERON's AMI provides the following functionality: 1) access to Smart Meter and IoT data (i.e. temperature and occupancy), 2) access to user schedules provided through manual operation, and 3) methods for controlling assets in the end-user premises. Currently, the RAI API wraps only access to the Smart Metering data, whereas work on other interfaces is in progress. Additionally, the RAI API provides access to Weather data through OpenWeatherMap service for three piloting locations in Greece.

In the Slovenian pilot the RAI is implemented on top of a southbound ECE/Amibit HEMS cloud interface. The southbound interface is wrapped with the RAI backend implementation implemented in Python, Tornado web framework[6] and a backend database MongoDB[7]. The RAI interface allows access to household devices information, close to real-time smart metering information, sensors data and controls of household devices when available. The RAI is augmented with weather data for the Celje region and tariff data for pilot users.

The RAI used in the Greek and Slovenian pilots provides common interface to access the measurement and sensor data, as well controls of the devices. A feature of the interface is an aggregation framework for the data, so that the data is aligned at multiple time intervals (5 min, 15 min, 60 min, day, week, month). The RAI is integrated tightly with the security and privacy interfaces as presented in Section 4.7.

The third phase design and implementation of the RAI component is described in more detail in D4.3 *Final Resource abstraction interface module*.

4.4 Automated flexibility management

The Automated flexibility management (AFM) component provides services for implementing DR control and optimization operations. This is achieved by implementing model predictive control (MPC) with the models provided by the Digital Twin Repository. More concretely, it provides the following functionality:

- **Baseline forecasting:** AFM component requests forecast from digital twin repository to form a baseline load forecast. Then it publishes the baseline forecast via MQTT.
- **Flexibility estimation:** AFM component predicts minimum and maximum loads utilizing models from the digital twin repository together with internal optimization module for different timeslots in order to evaluate the flexibility of the prosumer.
- **DR control and optimization:** AFM component utilizes the digital twin models to predict the demand response for different control options and iterates through them in order to find optimal control strategy. Chosen controls are sent to the Resource Abstraction Interface via MQTT.

An UML class diagram of the third phase AFM-implementation is depicted in Figure 8. There are no major changes in the AFM module compared to the second phase prototype. The main changes include new Resource Manager implementations for the HVAC system in the Slovenian pilot and the water boilers in the Greek pilot.

[5] http://docs.oasis-open.org/obix/obix/v1.1/csprd01/obix-v1.1-csprd01.html
[6] See Tornado web pages for details: https://www.tornadoweb.org/en/stable/
[7] See MongoDB wep pages for details: https://www.mongodb.com

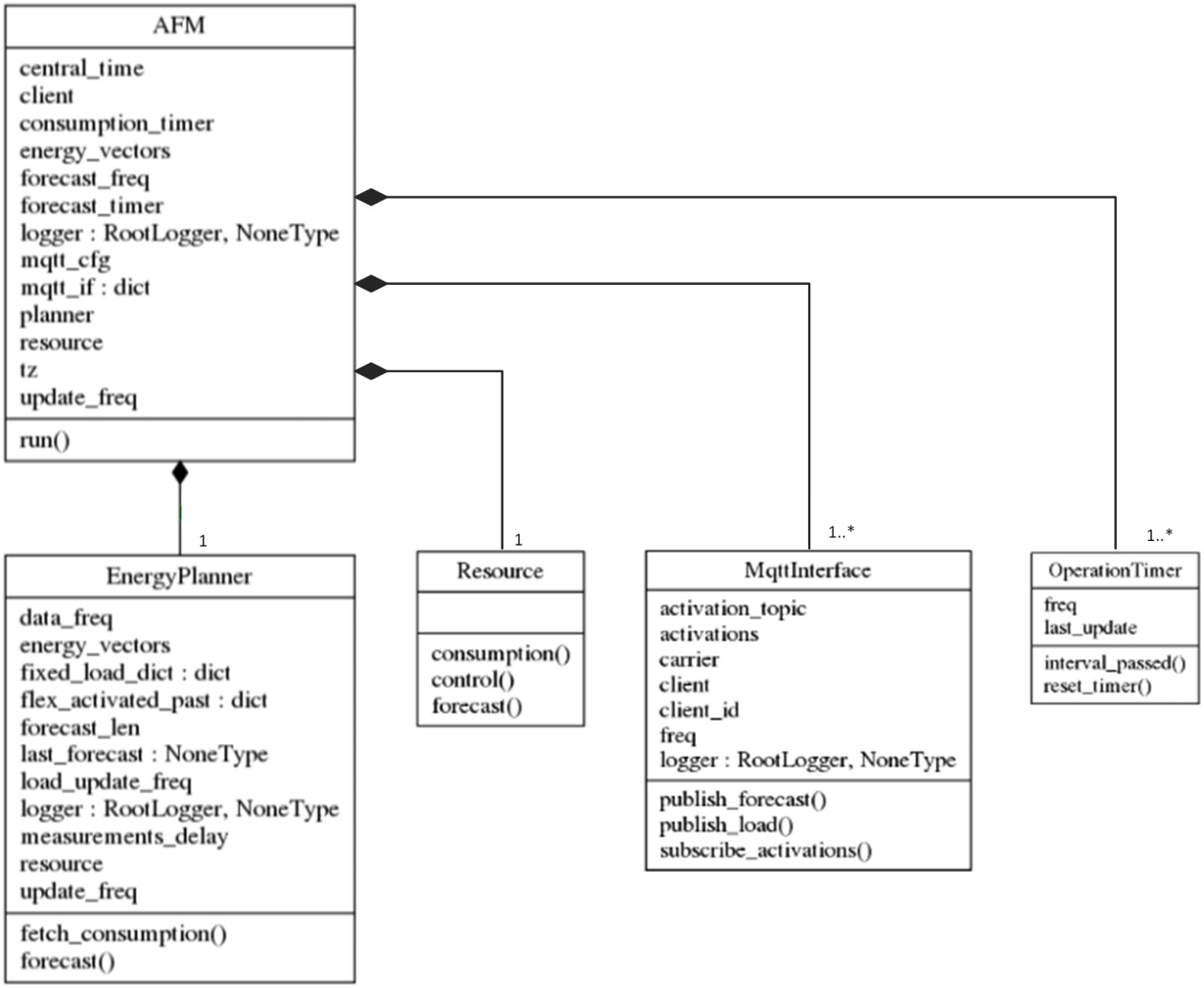

Figure 8: UML class diagram for the Automated Flexibility Manager.

The architecture consists of two main entities represented by the EnergyPlanner and Resource classes.

The EnergyPlanner entity is similar to the Customer Energy Manager in the EN 50491-12 standard family. There is a single EnergyPlanner component for each AFM. 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, and 4) informing each Controller about the flexible asset specific load profiles the asset should follow. The EnergyPlanner utilizes MPC based approach for optimizing the energy and flexibility management within consumer premises.

There is a single Resource 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 EnergyPlanner. 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 final version of the AFM includes Resource Managers for four types of resources: 1) hybrid heating system consisting of an exhaust air heat pump (deployed in the apartment building in the Finnish pilot), 2) water boiler(s) deployed for consumers in the Greek pilot, 3) air-to-water heating system (deployed in households in the Slovenian pilot), and 4) combined refrigeration and heating system (deployed in a supermarket in the Finnish pilot).

In addition to the EnergyPlanner and Resource classes, the other main classes of the implementation include: 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 MqttInterface class implements the MQTT interface of the

AFM module. The OperationTimer class represents timers that are used to synchronize the operation of the EnergyPlanner.

AFM module is written in Python programming language. Data is primarily stored in DataFrame format form Pandas (McKinney, 2010) library. The MPC optimization, as well as some other data manipulation, is implemented using NumPy (Harris et al., 2020). Communication to other interfaces is done via MQTT protocol, a Python client is implemented by Eclipse Paho project[8].

The final version of the AFM component is described in more detail in *D3.9 Final Automated flexibility management module*.

4.5 Digital twin repository

The main purpose of the Digital twin repository is to:

- Estimate the impact of a consumer (i.e. metering point) for flexibility management.
- Provide optimal and consumer-centric flexibility management with model-based planning and control.

To this end, the Digital twin repository module consists of models for predicting and simulating consumer loads, flexibility and potential response to flexibility signals. These models are utilized by the Automated Flexibility Management module to estimate the baseline, flexibility and response of the consumers. The models are implemented with combination of machine learning and physics-based modelling. The current version of the repository includes digital twins for an apartment building (the Finnish pilot) and residential houses (Slovenian and Greek pilots).

The Digital Twin (DT) for apartment buildings consists of four main models as illustrated in Figure 9. It contains two machine-learning models for predicting the electricity and district heating baseline energy consumption, a physics-inspired heating system model with several parameters learned from data, and a simple physics-based model for predicting the indoor temperature during the DR events. The revised version of the DT utilizes artificial neural networks for forecasting the baseloads for electricity and district heating. Thermal flexibility models include a room temperature model based on Newton's law of cooling and simple energy consumption models for space heating provided by heat pumps and district heating. The idea in the heating system model is to modify the baseline prediction provided by ML models according to the law of energy conversion. The full heating system model as well as the other models are presented in more detail in D3.3 *Final Hybrid Modelling Module*.

[8] https://www.eclipse.org/paho/index.php?page=clients/python/index.php

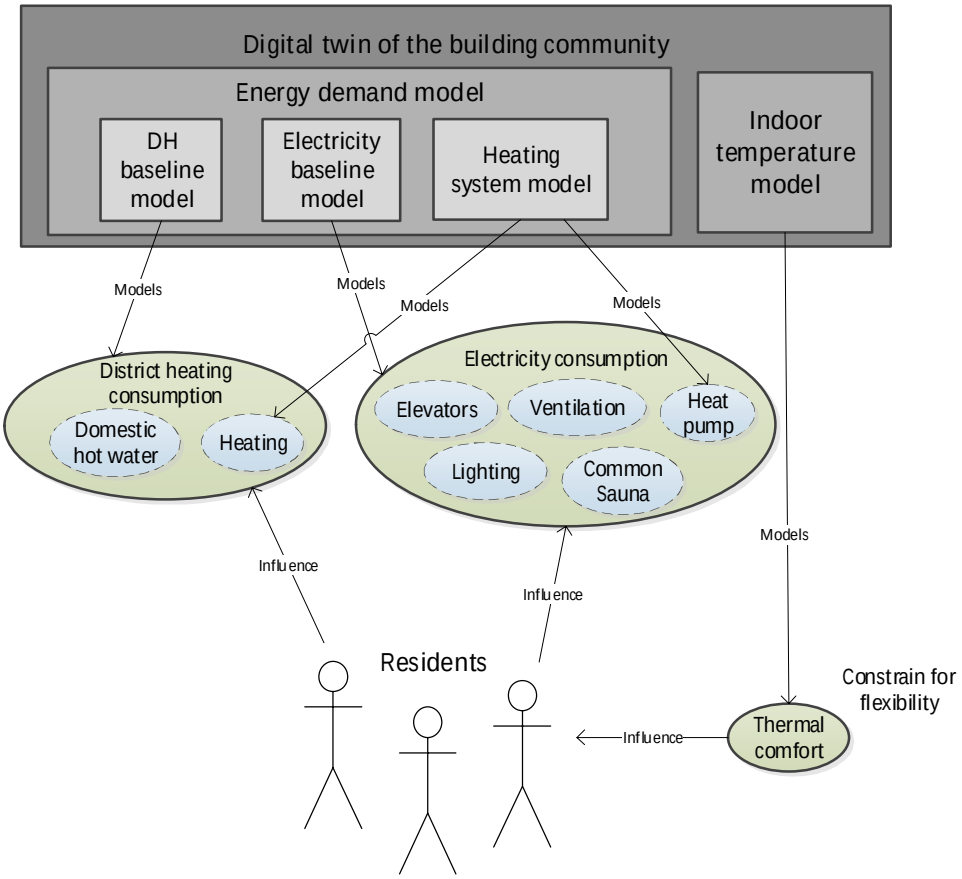

Figure 9: Conceptual representation of the interdependencies among the Digital Twin (measured parameters (green), non-measured parameters (blue)) and the residents of the building community.

The digital twin of a household, as illustrated in Figure 10, is based on four models: the thermal house model, the electrical model, the flexibility model, and the mobility model. The first basis of the model is explained and presented. The data used for modelling is displayed. This model consists partly of a neural network and partly of a gradient enhancement solution.

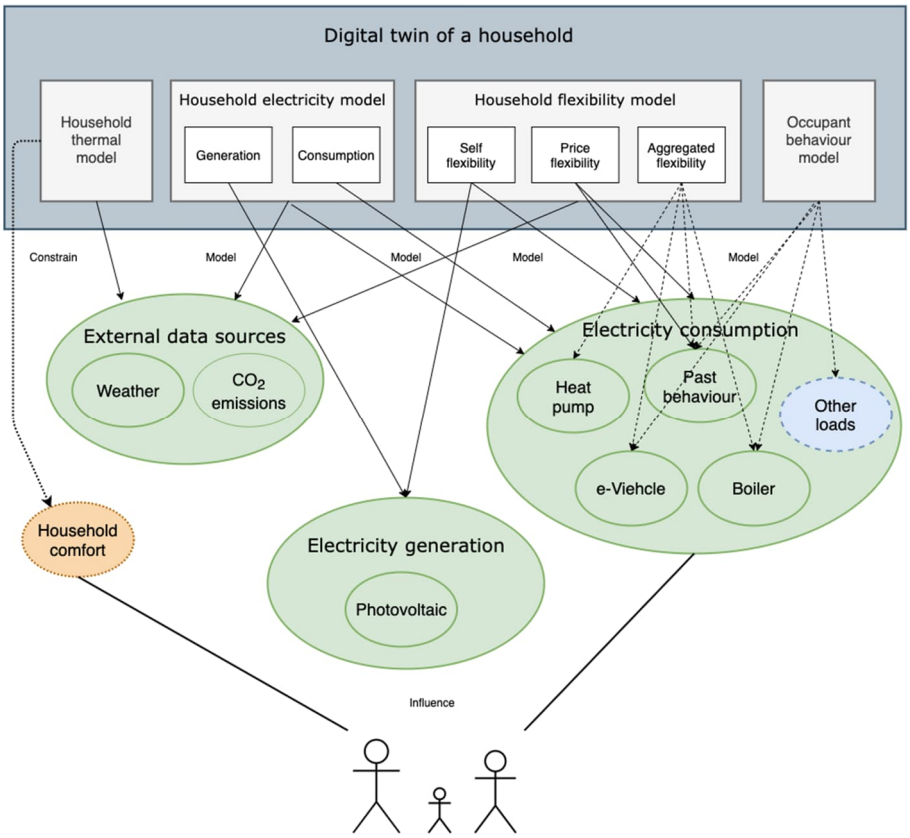

Figure 10: Household modelling (models in grey, measured parameters in green, constraints in orange, non-measured parameters in blue). Full arrows represent phase one focus.

The digital twin repository module is implemented with Python programming language. All ANN models in the repository are implemented with Tensorflow 2.0 (Abadi et al., 2016) on top of its Keras API (Chollet, 2018). Physics-based models are mainly implemented in NumPy (Harris et al., 2020). The predictive data analysis tool scikit-learn (Pedregosa et al., 2011) is used to determine some parameters of physical models and data aggregation. Pandas (McKinney, 2010) is used for analysis and manipulation of time series.

The revised version of the Digital twin repository module is presented in more detail in D3.3 *Final Hybrid-modelling module*.

4.6 Weather service interface

As the name implies, the role of the Weather service interface module is to provide data about weather forecasts to the iFLEX Framework. This module is utilized inside the iFLEX Assistants to provide the digital twin repository component with weather forecast data, which were required for predicting the response of weather-related consumption units such as the building's heating and cooling system.

Providers of the weather forecast services are typically country specific. Because of this, the weather service interface module consists of individual modules for different countries. The final version of the module contains interfaces for accessing weather forecasts provided by the Finnish Meteorological Institute (FMI) and the OpenWeatherMap in Slovenia. There is no interface for the Greek pilot because the flexible assets utilized in the Greek pilot (i.e., water boiler and other smart appliances) are not weather dependant.

The FMI provides an open API to access Finland's weather forecast 48 hours in advance. The data is shared according to the standards of the Open Geospatial Alliance[9]. The module providing access to weather forecast data in Finland is integrated to the RAI component via a common database. Figure 11 illustrates the setup for integrating FMI data into the iFLEX Framework. The interface for fetching data into the RAI database is

[9] https://www.ogc.org/standards

implemented in Java programming language. A client for accessing the weather forecast data via the oBIX RESTful API is implemented in Python to enable easy integration with the Digital twin repository module.

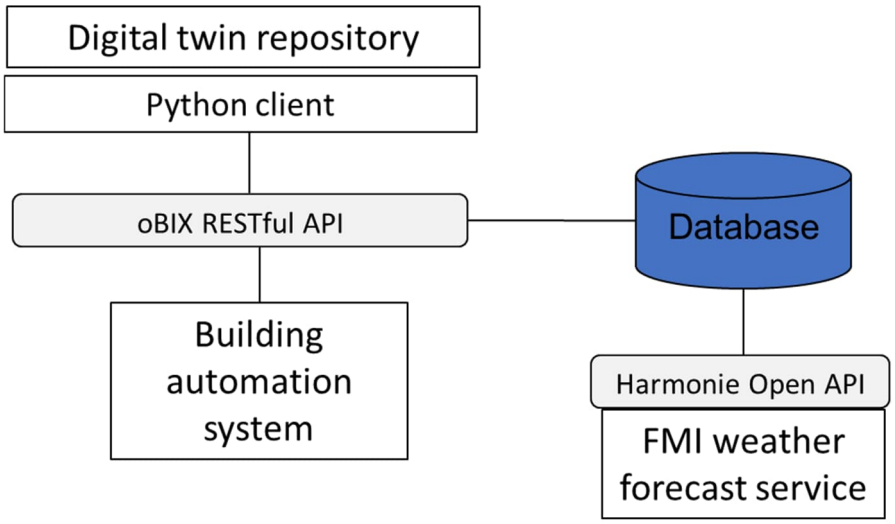

Figure 11: Weather forecast interface module integrated into the Resource abstraction interface component.

In Slovenia, the weather data is integrated as a combination of weather data ingestion service, feeding the Prosumer Cloud Service, and the REST API providing access to forecasted and realized weather data. The ingestion service is implemented in Python, current implementation is further described in deliverable D4.3 *Final Resource abstraction interface module*. The weather data provides information on temperature, radiation and precipitation on 1 h basis. The forecast is available for 3 days in advance for 1 h granularity. The weather data originates from the free OpenWeatherMap service[10].

4.7 Security and privacy interface

The Security and privacy interface provides a number of services and mechanisms needed for implementing the security and privacy requirements as defined in the deliverable D2.1. The following services are currently implemented:

- Enrolment module: the service provides a web portal for end-user registration and personal account management. During the registration, the certificates for the end user can be dynamically generated and provided. The service allows for utilizing of various invitation methods for registration, like magic numbers[11] provided in the electricity bill, etc. The service provides means to register an additional device under the same account with same privileges in a single trust management setup. The feature could be used for providing access control certificates for mobile devices to be able to access RAI REST API. The module takes care to pseudonymise incoming smart metering data in Slovenian pilot and provides user management interfaces for pilot host management of the pilot users data details such as HEMS ids, tariffs, etc.
- Trust management module: the module provides a Certification Agency for issuing digital certificates for services, service managers and end users. According to the application security policies, the certification can be issued according to the role that the entity (either a service or user) has in the system. Multiple trust management setups could be prepared, for example one per pilot. The module is based on OpenSSL[12] and implemented with a build automation software Make[13] and parts in a shellscript. An additional component is a component that allows issuing dynamic certificates based on authenticated and authorized access from other system components, like a privacy-compliant, end-user engagement service. The module design has been extended with Self Sovereign Identity (SSI) management implementation. Both X.509 and SSI implementation provide similar functionality for basic services. The SSI implementation utilizes intrinsic properties of the approach and provides

[10] See the service webpage for details: https://openweathermap.org/
[11] https://en.wikipedia.org/wiki/Magic_number_(programming)
[12] See OpenSSL web page for details: https://www.openssl.org/
[13] See Gnu Make Wikipage for details: https://en.wikipedia.org/wiki/Make_(software)

the functionality in more user-oriented approach. Future requirements like ability to integrate additional outside entities and fain-grain delegation can be supported by the SSI out of the box. The SSI implementation is provided in cloud, in multitenant approach.

- Security and privacy interface: the interface is a collection of a number of interconnected security services, specified below:
    - Communication security: All the communication among the security components, system components and resources should be secured. In the baseline implementation, the communication was in general protected by Transport Layer Security -TLS protocol.[14] The TLS protocol provides data origin integrity and authenticity as well as confidentiality service of information exchanged in the communication.[15] The baseline approach is used in communication with the Prosumer Cloud Service and presented security components. The challenge of secure communication with the Resource abstraction interface and beyond, towards the devices and actuators and the End-user interface is a work in progress and needs to be further integrated in the final system setup.
    - Authorization interface: The interface provides an ability to manage devices ids in household access control security policies. The interface is used to add or remove End-user Interface device id to the household security policy. The policy is used while providing access control decisions.
    - Access control service: The Access Control Service (ACS) uses the digital certificates issued by the trust management module and internal security policies to provide access control decisions regarding access of system entities to the Prosumer Cloud Service REST API or access to messaging service messages, for example MQTT. The service is performing well to support large number of requests to the REST API or messaging communication channels. The policies can be fain grain and can control access to REST API HTTP methods (GET, PUT, PULL, etc.) or read and/or write access to MQTT topics. The policies access to resources can be either role- or identity-based. Through specific procedures, the policies could be updated to manage dynamically issued digital certificates.
    - Security interceptors: The interceptors are services or programmable features that enforce authorisation decisions. An MQTT interceptor is a service running as a plugin on a Mosquitto MQTT broker. The service intercepts each topic's read or write, authenticates the connection, invokes the authorization engine with information who would like to access the resource (topic) and in what manner (read, write) and enforces authorization decision when provided. The programmable solutions are embedded into the Python Tornado implementation environment and enable controlling access to each RAI API REST web server route. In the same manner the other services like Digital Twin Repository, Automated Flexibility Management, etc. will be mannered.

4.8 Tools for continuous integration and deployment support

In the early stages of the project, a development and integration environment has been implemented based on widely established tools to facilitate iFLEX component development and integration. The environment included the following main DevOps tools: JIRA tool for requirements engineering and issues/bug management, Gitea as the server for code management, and Drone in combination with Gitea for continuous integration and deployment.

The iFLEX technical work packages were organised in three major phases, original timing of the phases was adapted due to the six-month project extension and, thus, the phases were running between M1-M14, M15-M30, and M31-M42. Due to the adopted agile methodology, component development, system integration, and components/system testing are intertwined, coupled, and performed in a continuous manner.

[14] See TLS Wikipages for details: https://en.wikipedia.org/wiki/Transport_Layer_Security
[15] All security terminology used in the document is aligned with the RFC4949 - Internet Security Glossary, Version 2, see the glossary online: https://datatracker.ietf.org/doc/html/rfc4949

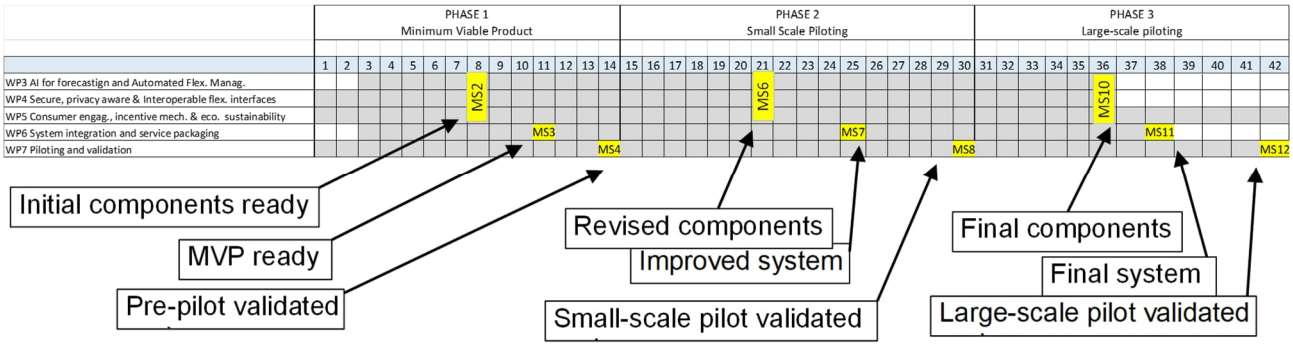

Figure 12: Updated iFLEX integration timeplan

Nevertheless, in different parts of a single project phase, the main focus was on a different aspect of technical R&D, as depicted in Figure 12. Thus, in the first part of each phase the emphasis was on the development of various iFLEX components in line with specifications and user feedback. The focus then shifted to the delivery of an integrated iFLEX Framework and customized iFLEX Assistant prototypes and deployment thereof at the three pilot sites. The last part of each phase was mainly concerned with execution of the pilot in the field and validation of the piloted system. The experiences and results of the first two pilot phases were used as feedback for further developments and improvement of iFLEX components in subsequent phases.

5 Conclusion

This is a demonstrator-type deliverable that documents the third phase implementation of the iFLEX Framework. The iFLEX Framework presented in this deliverable consists of the components developed in the technical Work Packages of the project. The components of the framework and their main responsibilities include:

- **Aggregator and market interface**: Enables the interaction between the iFLEX Assistant and the market.
- **End-user interface**: Enables the consumers/prosumers to interact with the iFLEX Assistant.
- **Resource abstraction interface**: Provides iFLEX Assistant with means to monitor and control energy consumption, comfort, and other relevant aspects in the consumer/prosumer premises.
- **Automated flexibility management**: Provides services for implementing DR control and optimization operations.
- **Digital twin repository**: Estimates the impact of a consumer for flexibility management and provides optimal and consumer-centric flexibility management.
- **Weather service**: Provides country-specific data about weather forecasts.
- **Security and privacy interface**: Provides trust and security among iFLEX subsystem and services, users and data.

This document provides a high-level description of the whole iFLEX Framework. More details for each functional component of the framework are provided in component specific deliverables referenced throughout this deliverable.

6 List of figures and tables

6.1 Figures

7 References

(AIOHTTP, 2022) AIOHTTP. (2022, September 21). Welcome to AIOHTTP. Retrieved June 21st, 2022, from https://docs.aiohttp.org/en/stable/

(Chollet, 2018) Chollet, F. (2018). Deep Learning with Python and Keras. MITP-Verlags GmbH & Co. KG.

(Django Software Foundation, 2022)

Django Software Foundation. (2022, September 21). The Django project. Retrieved June 21st, 2022, from https://www.djangoproject.com/

(EC, 2007) European Commission (2007). A lead market initiative for Europe. Brussels. COM(2007) 860 final.

(Harris *et al.*, 2020) Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*. https://doi.org/10.1038/s41586-020-2649-2

(McKinney, 2010) McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference.

(MongoDB: The Developer Data Platform, 2022)

MongoDB: The Developer Data Platform. (2022, September 21). Retrieved July 19th, 2022, from https://www.mongodb.com/

(Node.js documentation, 2022)

Node.js documentation. (2022, September 21). Retrieved July 19th, 2022, from https://nodejs.org/en/docs/

(OpenADR Alliance, 07-01-2013)

OpenADR Alliance. (07-01-2013). OpenADR 2.0. Profile Specification - B Profile v1.0, (Final Specification).

(OpenLEADR, 2022) OpenLEADR. (2022, September 21). Retrieved May 17th, 2021, from https://openleadr.org/

(Pedregosa *et al.*, 2011) Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research.

(PostgreSQL, 2022) PostgreSQL. (2022, September 21). PostgreSQL: The World's Most Advanced Open Source Relational Database. Retrieved June 21st, 2022, from https://www.postgresql.org/

(React: A Javascript library for building user interfaces, 2022)

React: A Javascript library for building user interfaces. (2022, September 21). Retrieved July 19th, 2022, from https://reactjs.org/

(React Native, 2022) React Native. (2022, September 21). Retrieved July 19th, 2022, from https://reactnative.dev/