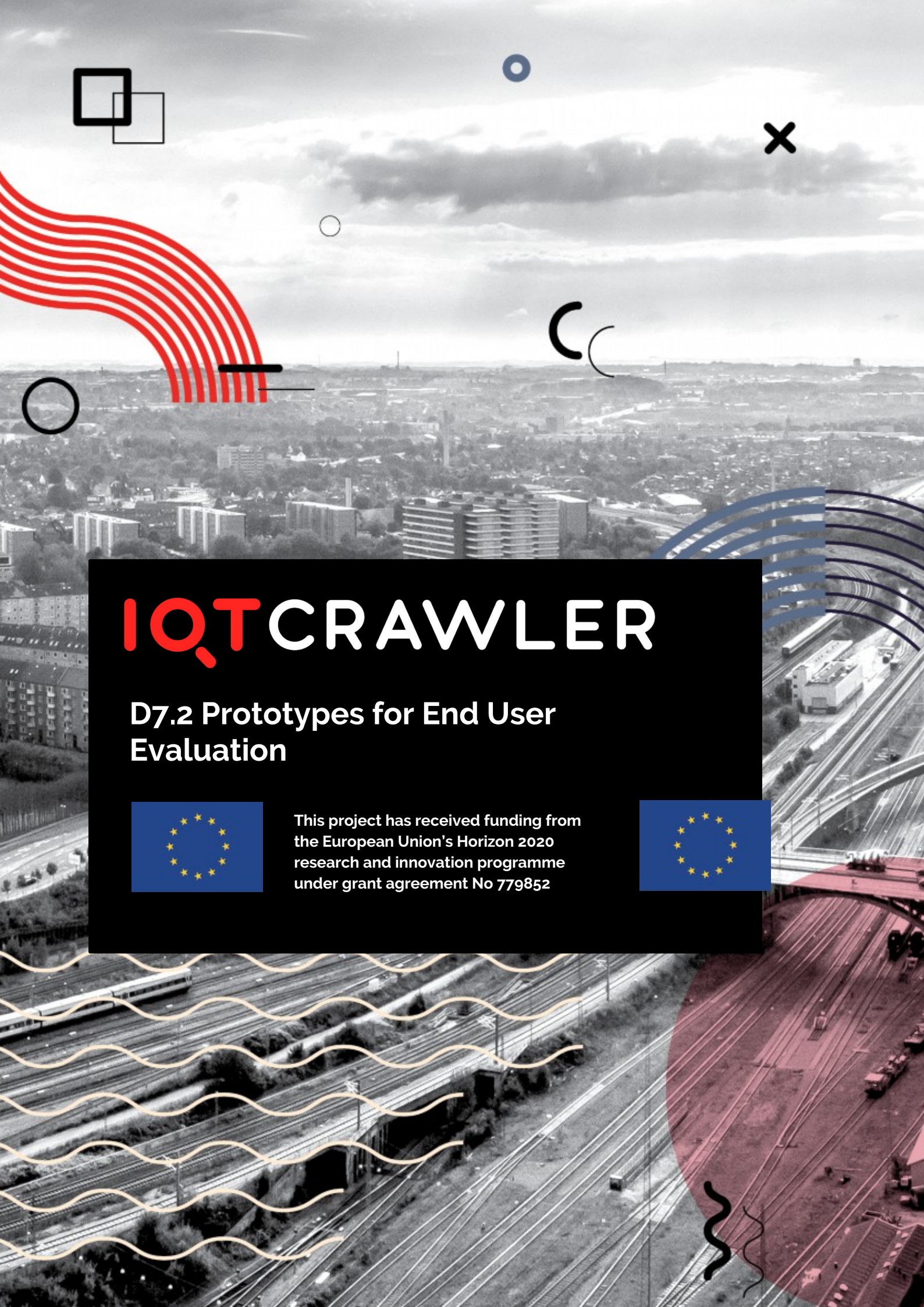

IQTCRAWLER

D7.2 Prototypes for End User Evaluation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779852

IOTCRAWLER

Title:	Document Version:
D7.2 Prototypes for End User Evaluation	1.0

Project Number:	Project Acronym:	Project Title:
779852	IoTCrawler	IoTCrawler

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type* - Security**:
31/01/2021	31/01/2021	D – PU

* Type: P – Prototype, R – Report, D – Demonstrator, O – Other

** Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)

Responsible and Editor/Author:	Organisation:	Contributing WP:
Pavel Smirnov	AGT International	WP7

Authors (organisations):

Sebastian Christophersen (AAR), Marianne Krogbæk (AAR), Michail Beliatis (AU), Parwinder Singh (AU), Juan A. Martinez (OdinS), Pedro González Gil (UMU), Antonio Skarmeta (UMU), Rohit Bohara (DW), Tarek Elsaleh (UoS), Andreas Fernbach (SIE), Stefan Bischof (SIE), Josiane Parreira (SIE), Pavel Smirnov (AGT), Martin Strohbach (AGT), Eushay Bin Ilyas (UASO), Thorben Iggena, (UASO), Marten Fischer (UASO), Hien Truong (NEC)

Abstract:

This deliverable concludes task T7.2 "Development of prototypes MVP", where 6 functional prototypes have been developed to the state when they can be exposed to end-users for evaluation of the user experience and the potential business models they are unlocking. Each of the MVPs have their own domain-specific requirements and KPIs to the IoTCrawler platform. As part of this deliverable, we also present the evaluation of these KPIs. The prototypes cover use-cases in 5 different domains: Smart City, Smart Home, Smart Energy, Industry 4.0 and Social IoT. The prototypes have been implemented on top of the IoTCrawler platform and are demonstratable online or on-premise hardware.

Keywords:

Minimum Value Product, Integration, Testbed Use-case scenarios, Deployment, Best Practices

Revision History

The following table describes the main changes done in the document since created.

Revision	Date	Description	Author (Organization)
V0.1	29/10/2020	Initial Proposal for TOC	Pavel Smirnov (AGT)
V0.2	03/11/2020	Changed heading of section 2	Martin Strohbach (AGT)
v0.3	16/12/2020	Contribution to Smart Parking MVP	Juan Antonio Martinez (OdinS), Pedro Gonzalez (UMU)
V0.4	14/01/2021	Contribution to Smart Parking MVP and Best Practices	Juan Antonio Martínez (OdinS), Pedro Gonzalez (UMU)
V0.5	18/01/2021	All missing contributions gathered and aggregated	Parwinder Singh (AU), Rohit Bohara (DW), Andreas Fernbach (SIE), Josiane Parreira (SIE), Eushay Bin Ilyas (UASO), Thorben Iggena, (UASO), Marten Fischer (UASO), Tarek Elsaleh (UoS)
V0.6	19/01/2021	Review of Smart Home MVP and update of 3.3	Martin Strohbach
V0.7	20/01/2021	Review	Sebastian H. Christophersen, Hien Truong, Martin Strohbach
V0.8	28/01/2021	Review comments addressed. Ready to be finalized	ALL
V1.0	29/01/2021	Final version	Pavel Smirnov (AGT)

Abbreviations

AFRR	Automated Frequency Restoration Reserve
BEMS	Building Energy Management System
BESS	Battery Energy Storage System
CDF	Cumulative Distribution Function
CI-CD	Continuous Integration and Continuous Deployment
EVCS	Electric Vehicle Charging Stations
GKE	Google Kubernetes Engine
HP	Heat Pump
KPI	Key Performance Indicator
MDR	Metadata Repository
MVP	Minimum Viable Product
QoI	Quality of Information
RPZ	Regulated Parking Zones
SPA	Single Page Application
TRL	Technology Readiness Level
WHAT	What's Happening At Home
WP	Work Package

Executive Summary

This deliverable concludes task T7.2 “Development of prototypes MVP”. The objective of this task is to develop functional prototypes or minimum viable products (MVPs) for at least 5 scenarios defined in the outcome of task T7.1. These prototypes are required to be functional enough so that they can be exposed to end-users for evaluation of the user experience and the potential business models they are unlocking. This deliverable presents 6 MVPs developed for use-cases in 5 domains as defined in D7.1: Smart City, Smart Home, Smart Energy, Industry 4.0 and Social IoT. MVPs are implemented as standalone applications on top of the IoTCrawler platform and deployable in cloud or on a premise hardware. Each of the MVPs have a clearly defined use-case specific KPIs as well as KPI evaluation results. All MVPs are ready for demonstration and user experience evaluation.

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779852, but this document only reflects the consortium's view. The European Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

Table of Figures

1 Introduction

The main target of the IoTCrawler project is to develop a search engine for the Internet of Things provides access to vast range of IoT sources. The objective of work package 7 is to demonstrate the benefits of the IoTCrawler platform in the context of by applying it to real-world use cases. The goal of task T7.2 is to develop a set of technical prototypes (called Minimum Viable Products, MVPs). The idea of the MVP is to create of a basic version of a technology that can be usable and understandable by target users. The result is aimed for evaluating user feedback and testing the potential business opportunities. High-level use-case scenario descriptions as well as concepts have been generated during user-centric design process, which took place during task 7.1. At least 5 MVPs defined in D7.1 had to be implemented to the stage ready for user experience testing. The 6 selected use-case scenarios belong to 5 domains, such: Smart City, Smart Home, Smart Energy, Industry 4.0, Social IoT. The deliverable demonstrates the technical aspects of the MVPs implementation on top of the IoTCrawler platform. Key Performance Indicators (KPIs) are defined for each MVP according to a classification "ISO/IEC 25010 - System and software quality models" presented in Figure 1. KPI measurements and their explanations are provided. The results of the task T7.2 presented in this deliverable are a prerequisite for finishing task 7.3, in which evaluation of user experience and business model testing is being performed.

The document is organized as follows: the following 6 sections describe each of the MVPs in detail: use-case specifics, architecture, deployment, technical KPIs and their evaluation. In section 8, we describe how a Kubernetes Cluster was used for deployment. The deployment procedure is similar to the one used for the IoTCrawler core components. In section 9 we describe best practices showing how to use the core IoTCrawler components in the MVPs. These themselves components have been developed in WP6.

Figure 1: System and software quality models used for KPIs classification.

2 Smart Parking MVP

2.1 Description

SmartParking tries to minimize the roaming of vehicles around the city, in search for a parking spot. It tries to do so by providing an optimal way for drivers to find a potential parking spot, close to their desired destination. It even goes beyond real time parking availability measures, allowing for planning ahead of time, by utilizing estimates of parking allocation at the desired arrival time. When a driver wants to plan their route, he uses SmartParking (showcased in Figures 2,3) in order to find the closest parking spot for the desired destination and time of arrival, matching other constraints as well (such as price or different parking characteristics).

Figure 2: User getting directions to a near-destination parking from SmartParking.

Parking Providers are key stakeholders for SmartParking, as they will be providing parking for many of those drivers. The capability of IoTCrawler of integrating external data sources and semantically enriching that information, unlocks the capability of using indexing and ranking approaches for a performant and powerful search among very different data origins. Security is another aspect of interest in IoTCrawler for SmartParking, as both Parking Providers and potential clients of that

information (SmartParking in this case) might benefit on secure access to information depending of the specific use-case.

SmartParking offers a flexible solution, that can grow to other cities, and can accommodate other models of parking as well, such as the shared or collaborative economy approaches that have lately seen a flourishing, like AirBNB or Uber.

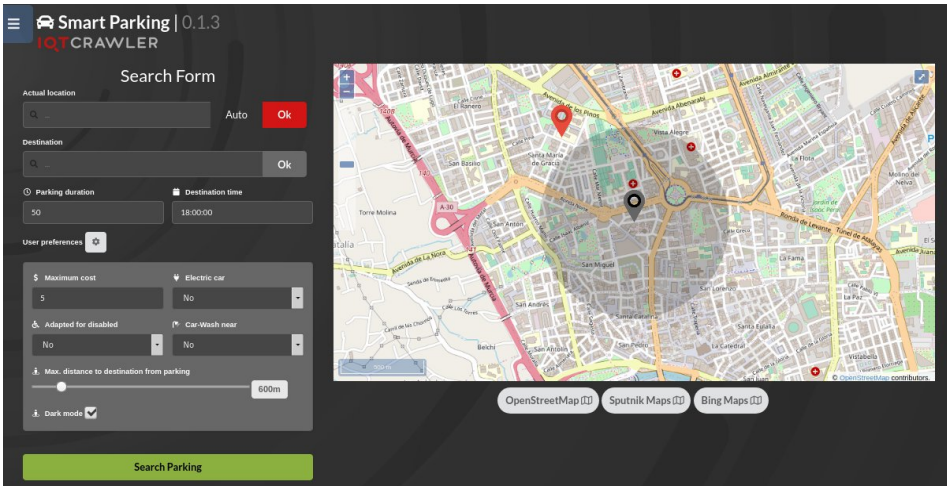

Figure 3: SmartParking web interface.

Additionally, this Smart Parking MVP takes advantage of the IoTCrawler framework so that parking site and regulated parking zone facilitators could specify access policies so that only authorised users could obtain the availability information for these parking site facilities. This way, a facilitator could specify that only premium users could access specific information parking sites, or that the information could be available only if the user is inside the specific city.

2.2 Implementation Details

SmartParking feeds at its base, from Parking-site Providers and Regulated Parking Zones (RPZs) data, as can be seen in Figure 4. That information, that is later processed in order to create future estimates, is integrated into IoTCrawler architecture, by means of IoT Connectors. Those are the pieces in charge of performing the crawling; that is, the discovery of all the relevant information offered by those providers, and its later registration in IoTCrawler's Meta Data Repository (MDR).

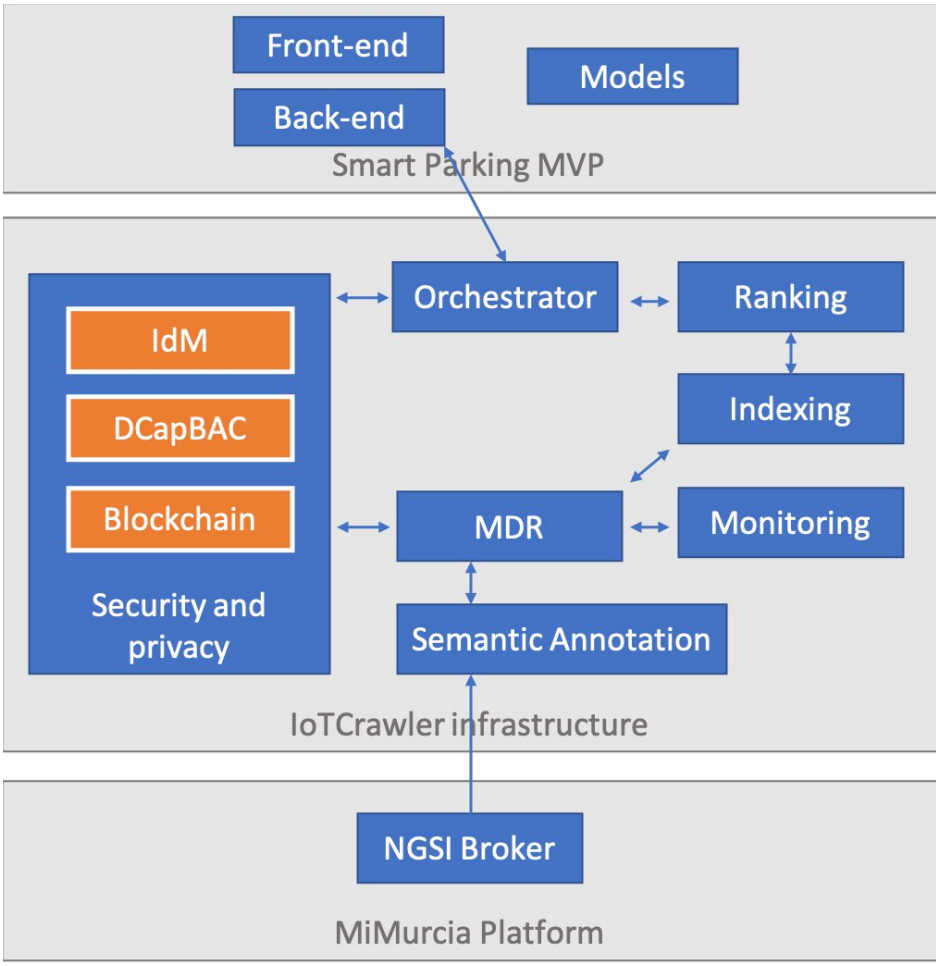

Figure 4: Component structure of the SmartParking MVP.

Before being introduced in the MDR, information must be semantically annotated so that it further expands IoTCrawler's capabilities beyond that of being a data warehouse (of sorts) into the search realm. IoTCrawler leverages semantic information in its data in order to index information and provide ranked results for searches.

Another component that further enriches the semantically annotated information, is the Monitoring, which continuously monitors information, searching for possible deviations that might imply faults in the related sensors. If some sensor is deemed to be faulty, it would be correspondingly annotated in the meta-information associated to it, in the MDR, to be used at a later time by the Indexing and Ranking components.

The contact point for SmartParking with the IoTCrawler framework, is the Orchestrator. This is the component where SmartParking sends user queries and

retrieves the potential results according to the set of constraints selected. Those results will contain matches from the very diverse parking providers registered in the system, whose IoTCrawler will search among, independently of their original data format and structure.

Another important aspect of IoTCrawler, relevant for the SmartParking MVP, is that of security. IoTCrawler provides a set of components that enable secure data access both for data providers and consumers. By defining policies, only some consumers are allowed to certain subset of information, according to different properties or attributes, allowing for a fine-tuned tailoring of information for different users. This could be the case for specific parking providers that only allow access to some of that information to premium users, or even more ambitious cases, such as providing parking information in-campus, only to faculty members, staff and students; or special parking for public health professionals; or public administration parking facilities only allowed to public servants.

2.2.1 Deployment

All the components of the MVP are deployed in a Google Kubernetes Engine (GKE) cluster, that is controlled via Gitlab's CI-CD mechanism, leveraging Helm Charts and Kubernetes for automated deployment, scaling and management of containerized applications. More details can be found in section 8.

2.2.2 Source Codes

The Smart Parking MVP is currently available from our internal Gitlab repository, as well as the public GitHub repository for the project. It comprises the IoTCrawler architecture instantiated for the MVP as well as the back-end and front-end developed for the MVP.

https://github.com/IoTCrawler
https://gitlab.iotcrawler.net/demonstrator-murcia/parking-iotcrawler
https://gitlab.iotcrawler.net/demonstrator-murcia
https://gitlab.iotcrawler.net/demonstrator-murcia/web-app-frontend
https://gitlab.iotcrawler.net/demonstrator-murcia/web-app-backend

2.3 Technical KPIs and evaluation

In Table 1, a list of the key performance indicators (KPIs) that SmartParking requires from IoTCrawler platform, can be found. KPIs are use-case specific and affect certain IoTCrawler components.

Table 1: KPIs for the Smart Parking MVP

#	**KPI and Category**	**Components involved**	**Rationale**	**Results**
1	The MVP should be able to collect data from at least two other sources of information to demonstrate the interoperability of IoTCrawler. (Compatibility)	MDR, Semantic Annotation.	The MVP needs to connect to both parking site availability information, as well as regulated parking zone information.	Two different sets of information have been integrated. MiMurcia parking site availability and Regulated Parking Zone (RPZ) parkingmeter estimation.
2	The platform should provide authentication and authorisation mechanisms to protect data disclosure. (Security)	MDR, IdM, DCapBAC.	The MVP must ensure that different users access to different datasets depending on their permissions.	Different access permissions were tested, so that one user had access to all parking information (RPZs and MiMurcia Parkings), while other had access only to MiMurcia.
4	Latency on obtaining results for parking queries below 2 seconds. (Performance)	MDR, Indexing, Orchestrator, DCapBAC	Queries for parking need to be served to the back-end, in as little time as possible.	Less than a second average measured latency.
5	Latency of data/metadata (parking availability information, and monitoring metadata) update below 5 seconds. (Performance).	MDR, Indexing	Parking information updates need to be available to the users as soon as possible.	Less than a second.

All the latencies represented in this section, have been obtained, based on 100 runs of the different queries under inspection, against the SmartParking MVP components hosted in the cluster instance described in Section 8: Deployment.

Latencies for KPI 4 (obtaining results for parking queries) have averaged 619ms, but out of those, an average of 577ms are directly related to token validation due to security components. This is caused by some implementation details related to the usage of elements external to the PEP proxy implementation. This interval can (and will) be dramatically reduced by making specific implementations for the token validation. Our estimations reduce the validation delay by at least one order of magnitude, bringing the final estimated total delay in the vicinity of 100ms.

3 Smart Home MVP

Within the Smart Home use case we were interested in understanding the challenges that homeowners are facing when deploying and using smart home devices. As a result of a previous project[1] and collaborations with smart home gateway vendors, we have identified that :

- smart home owners want to understand their energy consumption, reduce their energy costs and carbon footprint,
- and that power users having more than 200 devices need a better overview and knowledge about their devices.

But at the same time discussions with smart home owners and developers showed that IoTCrawler has the potential to be an effective platform for faster and secure development of a range of Smart Home applications.

3.1 Description

Due to the different aspects that we wanted to:

- assess the effectiveness of conveying energy insights
- test other use-cases for Smart Home applications

To do so we will describe two implementations of the Smart Home MVP. The first implementation is an extension of the GrowSmarter dashboard with IoTCrawler's semantic enrichment component. We have implemented and tested this implementation in a longitudinal study with end users in an early stage of the project. The second implementation is a simple smart home application that recognizes user activities. It is fully implemented on top of the IoTCrawler platform and serves as a basis for evaluating further smart home applications.

3.1.1 Energy insights dashboard

The energy insight dashboard was built with the objective to provide smart home users insights about their energy consumptions and thereby reducing their energy costs and carbon footprint. This was achieved by collecting energy measurements

[1] https://grow-smarter.eu/home

from smart plugs and other smart energy meters. The web-based application includes various aggregated and real-time views of the energy data as well as information about the usage frequencies of appliances attached to the smart plugs. Figure 5 shows the home screen of a user.

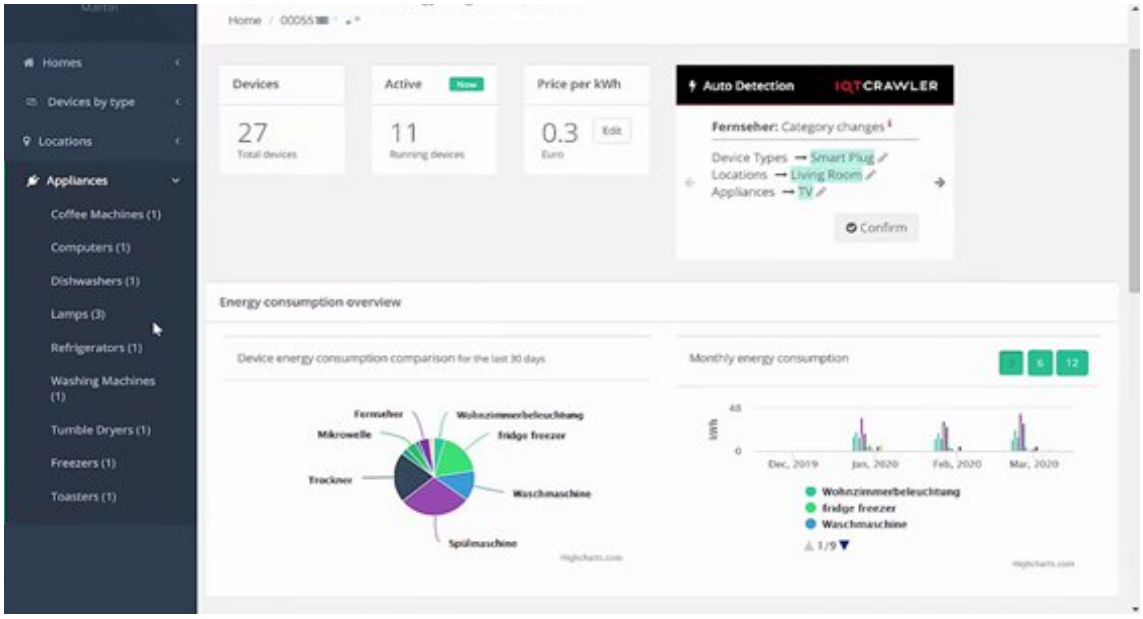

Figure 5: A user's home screen of the GrowSmarter Energy Insights Dashboard showing information about their devices, energy consumption and a notification about detected changes in his devices (in German).

As part of IoTCrawler we extended the dashboard to a public testbed running 24 hours a week for almost a year. More than 60 homes and 3,400 devices were connected during that period. Power users have more than two hundred devices connected to a smart home. Thus, we realized that managing these devices, including to know their locations, and for smart plugs what kind of appliances are connected to it, created a huge challenge for smart home owners. More importantly, the heterogeneity of devices with respect to their communication technologies, APIs and gateways they are connected to makes it hard to develop Smart Home applications that run seamlessly with different vendors. As a response to tackling this challenge we integrated an early version of an IoTCrawler feature for semantic annotations in which we use machine learning to detect device types, their locations and connected appliances in real-time [2].

3.1.2 What's Happening At Home

The application prototype called "What's Happening At Home" (WHAT) demonstrates how Smart Home applications can benefit from IoTCrawler's semantic abstraction for finding and accessing respective data streams. Using IoTCrawler it becomes much easier to develop smart home applications.

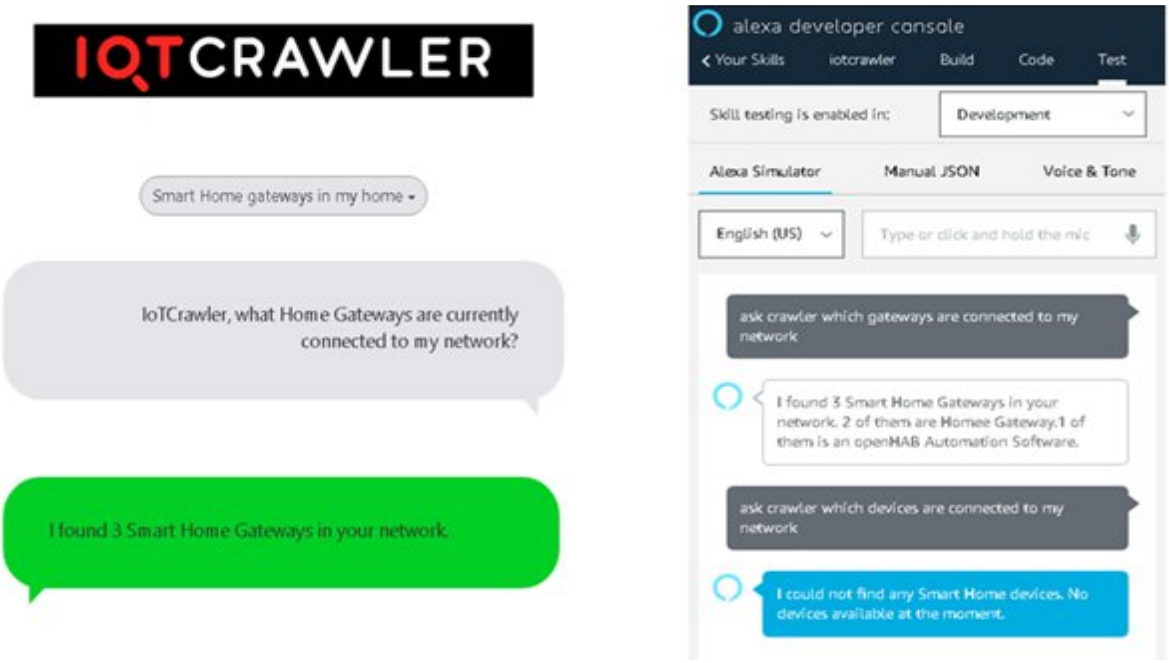

Figure 6. a) Screenshot of Dashboard App; b) Example of voice queries in Amazon Alexa

The WHAT prototype is fully integrated with the IoTCrawler infrastructure. The application detects user activities based on the energy consumption of appliances attached to smart plugs. The application interacts with IoTCrawler's Orchestrator and Search Enabler components. Activities are modelled in terms of Home Activity ontology, which is partly encoded in one of the GraphQL schemas used by the Search Enabler[3]. The schema allows to filter households by types or locations of detected activities (considering privacy policies). The application demonstrates separation between functionalities and benefits from the granularity of the IoTCrawler data model by dealing with sensors and their streams.

3.2 Implementation Details

3.2.1 Energy insights dashboard

The energy insights dashboard is implemented as a standalone cloud-based application and was deployed in a 24/7 operational environment. The application was targeted to a single type of smart home gateway (Homee[2]). The application's architecture as well as dependencies on the IoTCrawler stack are demonstrated in Figure 7.

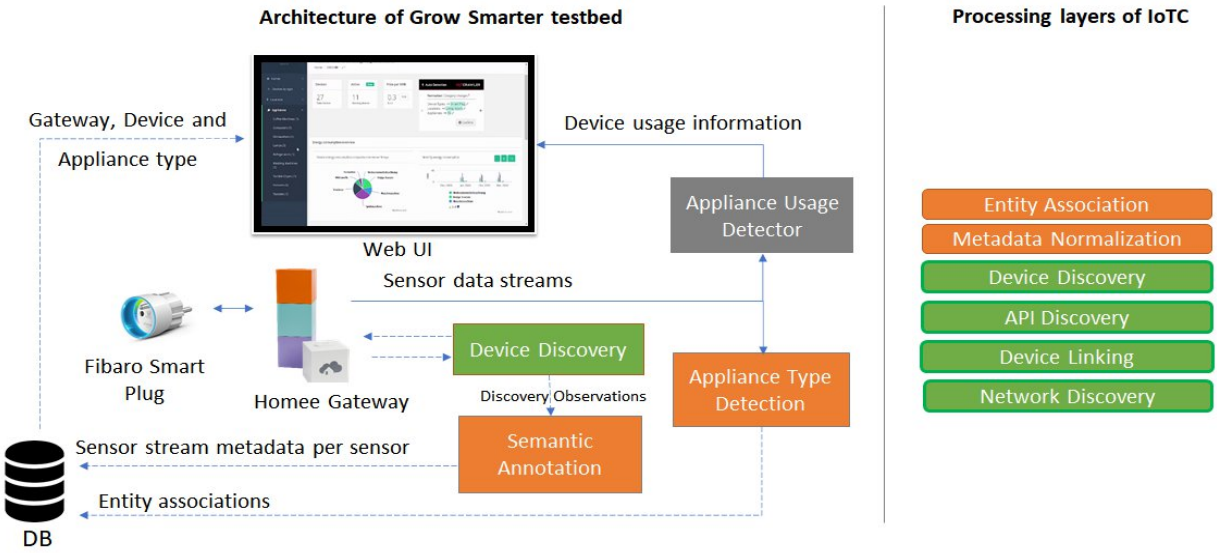

Figure 7: Architecture of Grow Smarter testbed demonstrating which levels of IoTCrawler stack have been reused.

3.2.2 What's Happening At Home

Sequence diagram of the WHAT app is demonstrated at Figure 8. Metadata and data of a user's smart home system is crawled by a local network crawler (running by user in his local network) and sent by source adapter (part of a crawler) to an NGSI-LD broker that is located either in the user's network or on the resources of the vendor of a smart home service. All the data access endpoints used by the applications are hidden behind security components (Auth Enabler & PEP proxy), so to perform any kind of data access, the applications have to pass the authentication procedure and get the necessary access tokens. Access tokens restrict the data access capabilities depending on a user role (e.g., smart home owner, analytical

[2] https://hom.ee/

service). The security components introduce a security layer over the data storage layer, which is highly relevant for smart home users.

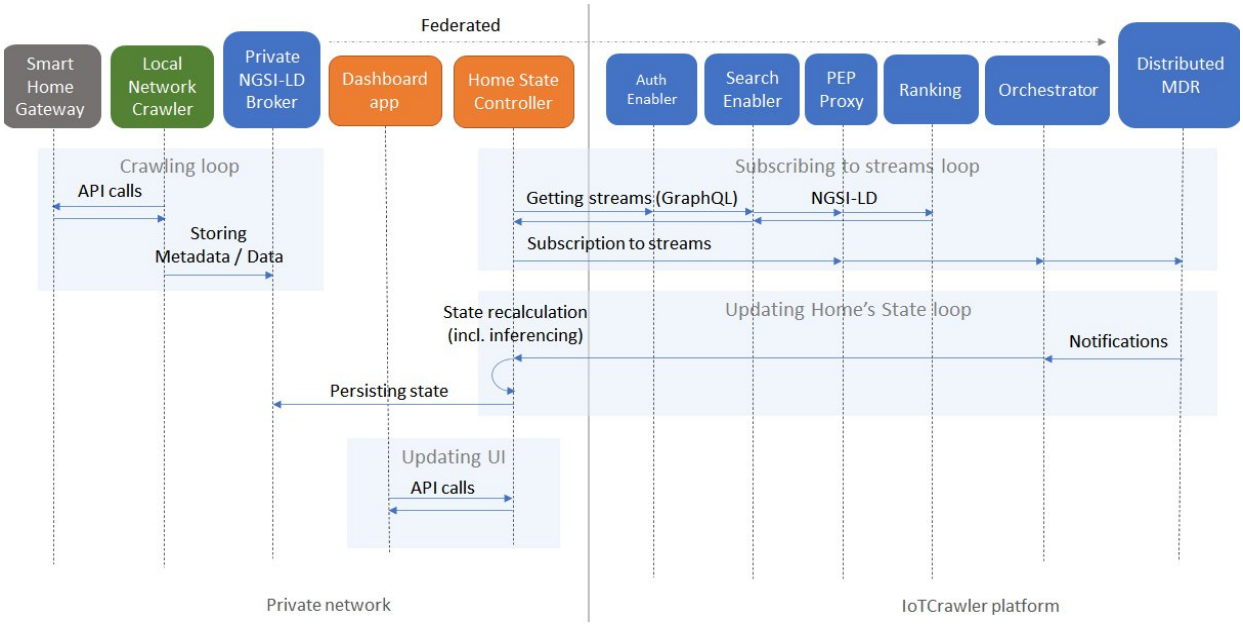

Figure 8. Interactions diagram of What's Happening At Home application

The application consists of two main components (marked orange):

Home State Controller – a component for maintaining the actual state of Smart Home by subscribing to relevant energy-measuring streams. Searching relevant streams is performed via continuous GraphQL query execution via the Search Enabler component. Arrival of new stream observations trigger their analysis, which affects the home state. The home state contains of list of devices, which are in active use and a list of activities happening in certain locations within a home. The state is intended to be persisted in the privately hosted NGSI-LD broker together with all other data of the user.

The Home State Controller plays the role of the analytics component, which constantly monitors observation streams and infers about activities based on appliance type and its location. Current implementation is based on a simple rule-based reasoning (a number of "if-then" rules). In more complex scenarios, the inference algorithms can also consider the statistical usage data, anonymously gathered from all households (using the federated mechanism of the distributed metadata repository). In this case, the analytical part of the State Controller might be

separated and offered as a standalone online service and State Controller remains responsible for providing input data and persisting analytics results into a private NGSI-LD broker.

Dashboard App – is a web-based application aimed to reflect the actual home state and send notifications to user. The Dashboard app does not perform any processing and because of that it can also be implemented as a mobile application (e.g. for wall-mounted tablet) or integrated into a voice-assistant such as Amazon Alexa (see Figure 6).

The application depends on data gathered and stored by supplementary resources:

1) Smart Home Crawler – crawls the local network of user and finds the SmartHome gateways to access the data. Found gateways are serviced by source adapters to send data (together with metadata) to an NGSI-LD broker located in the same or different network.

2) Private NGSI-LD broker – IoTCrawler platform does not consider storing target user's data. Users, who want to make their data operable within IoTCrawler, must store it in their private NGSI-LD brokers, which is considered to be federated into distributed MDR. Federation means, that data is not physically there, but is accessible from the distributed MDR via enforcing all the required security procedures. As a result, user can grant an access to a third-party analytics service developer and get some benefits (e.g. free service usage).

3.2.3 Deployment

The infrastructure components (Home State Controller, Local Network Crawler including source adapters, private NGSI-LD broker) are expected to be running in a user's network to provide user with a full control over the data. The dashboard application is also expected to be the part of a user's private network but might be running on another IoT device or on a mobile platform (not implemented).

However, if the user wants to outsource these parts to a third-party service company, all these components will be running on their infrastructure and the user

would be using web-based or mobile app. This demonstrates another potential user-case, that shows how service companies can benefit from building their services on top of the IoTCrawler platform. The IoTCrawler components involved in this MVP (security component, orchestrator, search-enabler, ranking, MDR) are deployed online via the Kubernetes cluster as described in section 8.

3.2.4 Source code

The source code of the What's Happening At Home app is publicly available under the following url:

https://gitlab.iotcrawler.net/demonstrator-agt/WhatsHappeningAtHome

3.3 Technical KPIs and evaluation

Below there is a list of key performance indicators (KPIs), which SmartHome applications require the IoTCrawler platform to meet. KPIs are use-case specific and affected by a certain IoTCrawler components.

Table 12: Technical KPIs and results for the Smart Home MVP

#	**KPI and Category**	**Components involved**	**Rationale**	**Results**
1	Application must work in private networks (e.g., private home or company networks) without requiring users to make any configuration changes, e.g. opening ports (Functional Suitability)	Orchestrator	Smart home users expect application to work anywhere anytime without requiring any network configuration.	Reached: by using orchestrator and it's AMQP pub-sub mechanism which does not require application to listen on a publicly reachable endpoint
2	Technology readiness level (TRL) not less than 7 (Maturity)	Semantic Annotation	Tests involving real users should be conducted on the operational environment offered them as service	Reached: A fully complete functional prototype was deployed in the cloud and used 24/7 by home owners
3	Latency of finding relevant streams not	Authorisation Enabler,	A smart home application needs to find	Measured latencies: average: 0.17s, min: 0.12s,

	higher than 2 seconds (Performance Efficiency)	Search Enabler, Ranking, MDR	streams relevant Smart Plug and Energy metering streams in a reasonable time	max: 4.37s
4	Latency of getting data updates (IoTCrawler notifications) not higher than 5 seconds (Performance Efficiency)	Orchestrator, MDR	Delays in SmartHome should be real-time so that the user is experiencing the effect of its actions (power consumption of a device, turning on lights based on activities)	Measured Infrastructure latencies: avg: 19s, min: 3s, max: 30s
5	Integration of 5 Smart Home Data sources (Compatibility, Interoperability)	Semantic Annotation	There are many different vendors of smart home devices and gateways and homeowners expect a working solution despite technological heterogeneity	5 Smart home sources integrated under a common semantic abstraction: OpenHAB, Homee, Vera, Netatmo, HomeAssist

The measurements shown in Table 12 for KPI 4 show that the latencies are effectively too high. However, almost the whole latency is caused by the latency of the Scorpio broker. It must be noted that the IoTCrawler can easily user other NGSI-LD standard compliant brokers and that we expect major performance improvements to the Scorpio infrastructure. Therefore, the results shown in the table do not invalidate the IoTCrawler infrastructure. To demonstrate that, we have decomposed the whole end to end latency experienced by a user in three major latencies:

1) Source Latency. This is the latency of the connected IoTCrawler data source. In the smart home MVP such a data source is typically a gateway and its connected devices. The source latency describes the time between a change measurable by a sensor (e.g. temperature change) and its availability via the gateway API. This latency includes latencies of the sensor itself (e.g. sleep times), latencies introduced by the data transmission protocols (e.g. Zwave, ZigBee) and the gateway software. The gateway itself plays an important role in this latency. Typically we have

observed latencies not higher than 1 second.
2) Latency of source adapters. This is the time between when the data is available from the gateway API and the first update request to the MDR has been made. The source adapter latency typically includes data transformation and normalization and semantic enrichment procedures (if applied). The latency depends on the complexity of methods applied for transforming the data. In our case, simple format transformations JSON to NGSI-LD entities) are in the range of few hundred milliseconds.
3) Latency of IoTCrawler infrastructure: this is the time between inserting (or updating) an NGSI-LD entity in the MDR and the time when the application gets notified about this change. This latency includes the latency of Scorpio's broker notification mechanism and the latency of an optional AQMP mechanism (part of orchestrator, which allows to receive notifications from broker and puts them into RabbitMQ queue, to deliver them to applications running in private network). The AMQP latency is not higher than 500ms. This infrastructure latency is the actual latency measured in Table 12.

So that means that to hit the KPI of 5s the broker should have a latency of maximum 3s.

The experiment values were calculated as an average of 100 iterations performed on the following hardware setup:
1) Smart Home gateway (openHAB) running on a RaspberryPI in a private 1GB/s network.
2) A source adapter for the gateways running on a laptop (Intel Core i7-5600U 2.6 GHz, 16 GB Ram) in the same private network.
3) MDR and Orchestrator running online (as described in section 3.2.3).
4) WHAT application is running on the same laptop.

Relatively high (for the Smart Home domain) latencies (average is 19 seconds) caused by the current implementation of publish-subscribe mechanism of the Scorpio broker, which is still under active development.

For the SmartHome scenario, high latency values are not necessarily critical. But they strongly affect user experience and could provide a roadblock to technology adoption. But as described above we expect that the availability of high performance NGSI-LD compliant brokers will deliver that user experience.

4 Discovery of small flexibilities in low voltage grids

4.1 Description

The energy transformation aims at strengthening renewable forms of electrical energy generation like wind and solar power, but also supporting new categories of consumers like heat pumps and e-mobility. However, these developments raise requirements on electrical power grids. The highly fluctuating fed-in power of photovoltaic plants and wind turbines, the high-power ratings of e-car charging stations as well as the bad predictability of these new network participants cause high loads, especially on the low voltage grid. The challenge is to nevertheless provide grid stability and to prevent overloads at any time.

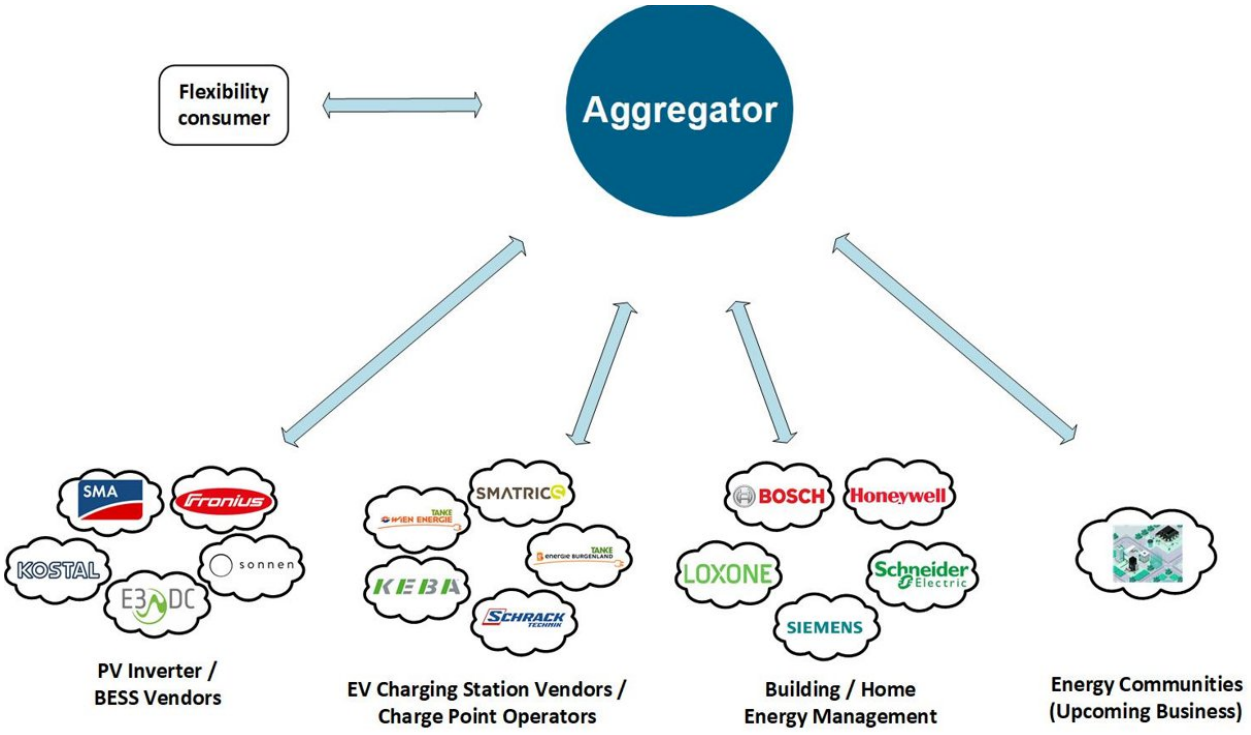

Figure 9: Aggregation of small flexible assets

A method to address this matter is to coordinate the load behaviour of these assets and to utilise the flexibility they potentially provide. This principle is well established for high-power assets like power and industry plants which need to pass a so-called prequalification to be allowed to participate in the control energy market. However, this cost-intensive procedure is not viable for small plants and single households. A way to circumvent this issue is to use a prequalified virtual power plant which

aggregates flexibilities from a high number of small assets. Applied within a balance group, this could be of big advantage to save balancing energy since fast reaction on deviations from power prognosis is possible. For discovering these flexibilities and for making them accessible to a balance group manager, a tailored infrastructure is required.

In the context of this project, Siemens addresses the question whether a discovery system for small flexible assets can be realised using IoTCrawler concepts. Therefore, semantic representations of these flexible prosumers are defined based on the IoTStream ontology. Attached meta information enables search and discovery across different deployments. Like in classical Web search engines, ranking mechanisms are in place to assure efficient access and sorting over a potentially large number of assets. As illustrated in Figure 8 assets from different domains are interfaced via their vendor-specific cloud backends by an aggregator. This includes home battery systems, PV inverters, E-car charging stations and building energy management systems. The advantage of using this pre-existing infrastructure is to separate the management and integration effort individual for each device and vendor from an aggregator solution. This way, an aggregator can communicate with a whole pool of assets by one central interface.

4.2 Implementation Details

An illustration of the overall architecture and how these components are used is shown in Figure 9. At the bottom, the data sources used for the demonstration of the use case can be seen. On the right, a real-life smart energy testbed located in the south of Austria incorporating several residential buildings equipped with photovoltaic (PV) plants, battery energy storage systems (BESSs), heat pumps (HPs) and electric vehicle charging stations (EVCSs). Building energy management systems (BEMSs) perform local energy optimization and coordinates the energy household between these appliances. Relevant datapoints (i.e., an information representation of the values of a physical quantity) from each appliance are exposed by an MQTT broker deployed on an Amazon AWS instance and hereby made accessible from the Web. A Data Acquisition component interfaces this MQTT broker via an SSH tunnel and in a first step calculates the upwards and downwards

flexibility datapoint values for each asset (where applicable) following a model which has also been developed in the course of this project. In a second step, these flexibility datapoints as well as the current load datapoint values are made accessible as NGSI-LD representations in the MDR.

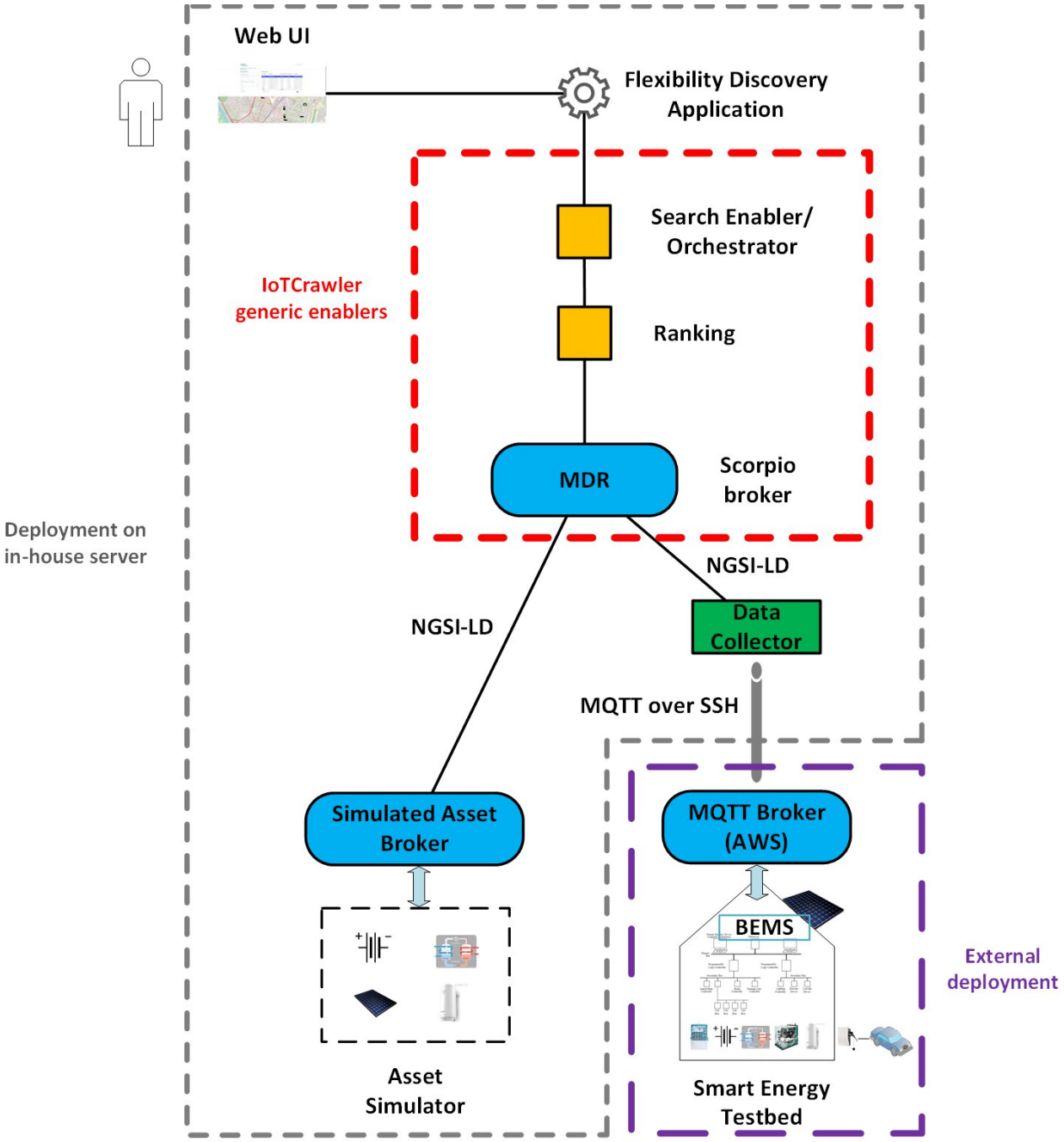

Figure 10: Architecture of the flexibility discovery MVP

In order to complement this rather limited number of assets available in this smart energy testbed, an asset simulator has been implemented (shown at the bottom left

of Figure 9). It allows a real-time simulation of the appliance classes described above. An arbitrary number of these assets can be instantiated and their datapoints exposed by an NGSI-LD representation (Simulated Asset Broker). This allows to upscale the number of assets integrated in the MVP towards numbers of real-life scenarios.

The SIEMENS MVP uses the following generic enablers from the IoTCrawler framework (indicated by the red dashed line in Figure 9):

- MDR
- Ranking
- Orchestrator
- Search Enabler
- Orchestrator

The MDR, in this case a Scorpio broker, holds a process image of the underlying assets in form of an NGSI-LD-based instance of the IoTCrawler ontology. This encompasses meta data on these assets like the asset type, a description, the current status of operation and the geolocation. Live datapoint values exist for the current load flow, upwards and downwards flexibility. The upwards and downwards flexibility datapoints (i.e. the respective IoTStreams) are annotated with reputation metrics which reflect the performance each asset during past flexibility activations. The MDR continuously receives updates on current load flow and flexibility datapoint value changes from the two data sources by means of NGSI-LD update requests. Hereby, the process image in the MDR is kept up to date at any time.

For accessing the MDR northbound by the NGSI query interface, several other generic enablers from the IoTCrawler framework are used. The Ranking component selectively evaluates the flexibility reputation metrics and orders the query results from the broker according to the selected asset property. This way, a query for a distinct type of flexibilities (either upwards or downwards) always returns with a list of assets ranked by the performance of the flexibility mode of current interest. This is required since assets in general have different flexibility provision performances for upwards and downwards direction.

The Flexibility Discovery Application has multiple tasks to process: once it runs the Web frontend shown in Figure 10 which is based on the Angular framework[3]. The other task is to handle the transformation of search criteria entered by users into GraphQL queries which are passed to the Search Enabler component and to process the query results received from the Search Enabler to a suitable format for the Web frontend.

User interaction with the Web frontend happens described in the following: The search criteria, i.e., the flexibility type, the asset type of interest and the geographic search pattern (circle or square) are entered in the Search Attributes form. The desired area is selected on the map by clicking and dragging. After hitting the search pattern, the results are shown in the table right to the Search Attributes form. Each line represents an asset with information on its type, a description, flexibility and current power values as well as the timestamp of the recent measurement. For each asset, a reputation value between 0-1 is shown in the rightmost column of the results table in Figure 10. This value indicates the performance of each asset as a flexibility provider. The bottom row of the table displays the sum values for upwards and downwards flexibility for all discovered assets. On the map, the geolocation of the search results is displayed by overlay icons. When hoovering the mouse pointer over an icon, the parameters of the respective asset are shown in a tooltip window. At the bottom of this user interface, a trend view shows the flexibility and power values of the selected asset from the past 24 hours interval.

[3] https://angular.io/

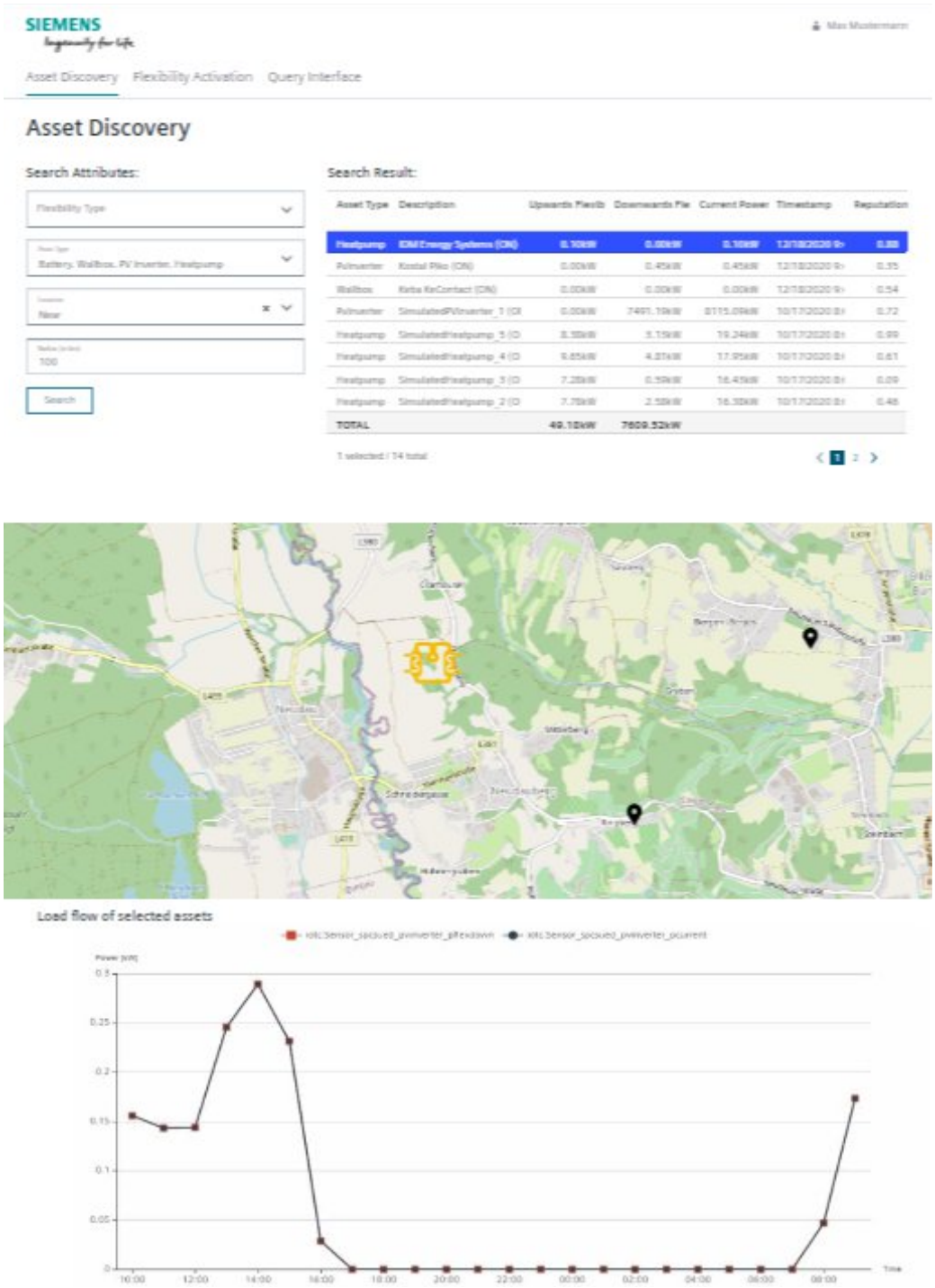

Figure 11: Web-based user frontend

4.2.1 Deployment

As indicated by the gray and purple dashed lines in Figure 10 all MVP components, except the Smart Energy Testbed and its associated MQTT broker, are deployed on an Ubuntu virtual machine running on an in-house server at SIEMENS City in Vienna. The testbed itself is at a remote location in southern Austria and the MQTT broker is

deployed at an Amazon Web Service instance. For the components deployed at the in-house server (Asset Simulator, Data collector, MDR, Indexing, Ranking, Search Enabler, Orchestrator, Flexibility Discovery Application) containerization by Docker is used.

4.3 Technical KPIs and evaluation

For the evaluation of this MVP, applicable quality attributes as defined in ISO/IEC 25010 have been selected. For the definition of KPIs desired values have been set for each selected quality attribute based on literature research and expert interviews. These KPIs and the evaluations carried out are described in the following sections. Table 3 gives a summary of these KPIs.

Table 3: Requirements and feature realization

#	KPI and Category	Components involved	Rationale	Results
1	KPI_{FC} (Functional Suitability)	MDR, Ranking, Orchestrator, Search Enabler	The MVP needs to be functionally complete with respect to the defined requirements	KPI_{FC} = 1 All defined functional requirements are met.
2	KPI_{RT} <Setup,Experiment> (Performance Efficiency)	MDR, Ranking, Orchestrator, Search Enabler	The MVP needs to meet certain performance goals to be able operate in the intended environment	The results is a matrix of KPI_{RT}'s for different scales and scenarios.
3	KPI_{IOP} (Compatibility)	MDR, Ranking, Orchestrator, Search Enabler	The MVP needs to be interoperable with the most relevant types	KPI_{IOP} = 1 The MVP is interoperable with all desired flexible asset

			of flexible assets	types
4	KPI_{MAT} (Reliability)	MDR, Ranking, Orchestrator, Search Enabler	For a convincing impression to the stakeholders, the MVP needs to have a certain level of maturity	KPI_{MAT} = TRL6 The MVP is able to operate in a real-live environment

Functional completeness

The functional requirements which form the basis for this evaluation were defined during Task 7.1 and refined during further progress of this project. This was achieved by a continuous stakeholder process with several feedback loops. For the final evaluation a number of meetings were organized where the concept of small flexibilities utilization was presented, and the MVP was demonstrated to different representatives from the Smart Infrastructure Digital Grids business unit of Siemens. Table 3 shows a comparison of the defined functional requirements and how they were realized by the MVP.

Table 4: Requirements and feature realization

Functional Requirement	**Realization in MVP**
Search, filter and rank flexibilities by various attributes	Orchestrator, Search Enabler operating on MDR
Web-based GUI with input form, map with asset icons, results table, trend view	Web frontend implemented based on Angular framework
Integration of real-life data sources via Webservice	Testbed equipped with typical types of distributed energy resources (PVs, BESSs, Heatpumps, EV Chargers)
Provide a reasonable quantity of data sources	Complementation of real-life assets with a scalable asset simulator

A KPI to express the degree of completeness is defined in the following. It evaluates to 1 if all required features are achieved and 0 if none of the required features were achieved.

$$KPI_{FC} = \frac{Number\ of\ achieved\ features}{Number\ of\ required\ features} = \frac{4}{4} = 1$$

As it can be seen, all desired features have been realized.

Performance efficiency

The relevant properties in this category are Time Behaviour and Scalability. In the context of this MVP this means that search requests on flexibilities need to terminate within a certain time interval where the search space consists of a reasonable number of instances. These two quantities follow from the use case scenario which was assumed under the conditions a transmission service operator (TSO) usually requests an automated frequency restauration reserve (aFRR) provider for activation. The time interval (full activation time, FAT) is typically 5 minutes in central Europe[4]. To reach this limit for the whole activation process, a search request for flexibilities should terminate and deliver results within t_{max} = 30 seconds. The other performance indicator is the number of NGSI-LD instances in the MDR which correlates to the number of potential flexible assets integrated in the system. Studies showed that these numbers are for Austria around 200.000 and for Germany around 3 Mio assets [4-9]. A degree of participation in an aggregator solution was assumed 30%.
To evaluate the performance of the MVP against these requirements a series of experiments have been carried out. The response times of sequential query requests issued to the Search Enabler were measured for various setups differing in the counts of asset instances in the MDR. The $KPI_{RT\ <Setup,Experiment>}$ is defined as follows:

$$KPI_{RT\ <\text{asset instance count}>} = \frac{Number\ of\ search\ requests\ terminated\ within\ t_{max}}{Number\ of\ issued\ search\ requests}$$

[4] Impact of Merit Order activation of automatic Frequency Restoration Reserves and harmonised Full Activation Times, ENTSO-E 2016

It evaluates to 1 if all search requests terminate within the defined interval t_{max} and 0 if no search request terminates within t_{max}.

For the performance evaluation following setups (S1…S7) varying in the counts of assets have been defined for being tested in different experiments (E1…E3). By the nature of the IoTStream ontology, the count of NGSI-LD instances is by factor 10 higher than the number of asset instances.

- S1: 10 assets
- S2: 100 assets
- S3: 250 assets
- S4: 500 assets
- S5: 1000 assets
- S6: 10000 assets
- S7: 100000 assets.

In each setup, for a 20% ratio of the instances the geolocation properties are set to different coordinate values than the remaining 80%. This is done to show the influence of filtering by geolocation to the search performance.

On these setups, three different experiments have been carried out:

- E1: Query assets and observations without filter
- E2: Query only for observations
- E3: Query assets and observations with geo filter set to the location of the 20% subset

For each combination of experiment and setup 20 query requests have been issued to the system. The following Table 4 shows the mean and the maximum values of response times in milliseconds over the number of runs for each combination of setup and experiment:

Table 5: Response times on different instance counts

Experiment	**E1**		**E2**		**E3**	
Setup	**Mean**	**Max**	**Mean**	**Max**	**Mean**	**Max**
S1: 100 NGSI-LD instances	948	1634	545	782	580	846

S2: 1000 NGSI-LD instances	1410	2590	815	1144	1314	1943
S3: 2500 NGSI-LD instances	2435	6004	1309	1587	2350	6032
S4: 5000 NGSI-LD instances	110716	110755	10031	10040	110651	110681
S5: 10000 NGSI-LD instances	Exception	Exception	Exception	Exception	Exception	Exception
S6: 10^5 NGSI-LD instances	Exception	Exception	Exception	Exception	Exception	Exception
S7: 10^6 NGSI-LD instances	Exception	Exception	Exception	Exception	Exception	Exception

This leads to following observations:

- Queries up to 250 assets (2500 NGSI-LD instances) have a good performance with 2,4 seconds average evaluation time
- Queries on a larger dataset than 500 assets (5000 NGSI-LD instances) lead to errors in the Scorpio Broker (org.apache.kafka.common.errors.RecordTooLargeException)
- The speedup when only querying for observations compared to full query (assets + observations) is factor 2
- Geo filters have a minimal effect in speed up currently, since geo-filtering happens at the Search Enabler component and not by the Broker

The KPIs for each setup and experiment therefore evaluate to the following:

Table 6: Performance efficiency KPIs

$KPI_{RT\ <Setup,Experiment>}$	**E1**	**E2**	**E3**
S1: 100 NGSI-LD instances	$KPI_{RTS1,E1} = 1$	$KPI_{RTS1,E2} = 1$	$KPI_{RTS1,E3} = 1$
S2: 1000 NGSI-LD instances	$KPI_{RTS2,E1} = 1$	$KPI_{RTS2,E2} = 1$	$KPI_{RTS2,E3} = 1$
S3: 2500 NGSI-LD instances	$KPI_{RTS3,E1} = 1$	$KPI_{RTS3,E2} = 1$	$KPI_{RTS3,E3} = 1$
S4: 5000 NGSI-LD instances	$KPI_{RTS4,E1} = 0$	$KPI_{RTS4,E2} = 0$	$KPI_{RTS4,E3} = 0$
S5: 10000 NGSI-LD instances	$KPI_{RTS5,E1} = 0$	$KPI_{RTS5,E2} = 0$	$KPI_{RTS5,E3} = 0$
S6: 10^5 NGSI-LD instances	$KPI_{RTS5,E1} = 0$	$KPI_{RTS5,E2} = 0$	$KPI_{RTS5,E3} = 0$
S7: 10^6 NGSI-LD instances	$KPI_{RTS5,E1} = 0$	$KPI_{RTS5,E2} = 0$	$KPI_{RTS5,E3} = 0$

As these results show, the current setup does not cover the requirements on scalability which would reflect a scenario based on the potential market in Austria. However, a reasonable performance was shown for a count of up to 250 assets. This scale would at least cover the next steps in MVP development where the integration of assets via real-live vendor infrastructures would be addressed. When a further uptake of this concept takes place, the bottleneck in this architecture needs to be identified and depending on the

outcome either distinct components need to be optimized or the complexity of the IoTCrawler ontology representing the assets could be reduced.

Compatibility

The field of application of the current use case scenario lies in a highly heterogeneous landscape of flexible assets. Therefore, Interoperability was considered most important here. A KPI to express this property is defined as follows. It evaluates to 1 if all required asset type interfaces are supported and 0 if none of the required asset type interfaces are supported.

$$KPI_{IOP} = \frac{Number\ of\ supported\ asset\ type\ interfaces}{Number\ of\ required\ asset\ type\ interfaces}$$

The most asset types relevant for provision of small flexibilities are photovoltaic inverters, battery energy storage systems, heat pumps and electric vehicle charging stations (wall boxes). One reason therefor is that these assets cause a significant load of the electrical grid which will even drastically increase in the next few years. The other reason is that these assets are more and more integrated in a monitoring and control infrastructure operated by the device vendors and are therefore relatively easily accessible by an aggregator solution like presented here.
The goal for this MVP was to show the integration of each of the before mentioned asset types by means of semantic asset models, flexibility calculation models and live data points. Since this goal has been fully achieved the KPI_{IOP} evaluates to:

$$KPI_{IOP} = \frac{4}{4} = 1$$

Usability

Appropriateness, Recognizability and Operability of the Web-based user frontend are evaluated by means of online demonstrations and workshops with stakeholders. The results will be documented in deliverable D7.3.

Reliability

To be able to present a meaningful and expressive implementation, a certain Maturity level needs to be reached. A metric (KPI_{MAT}) to describe maturity of a software artefact is the Technology Readiness Level (TRL). According to its definition, a prototype which operates in its relevant environment corresponds to TRL 6. Since this is the case for the current MVP, which operates on data from real-live assets KPI_{MAT} evaluates to the following:

$$KPI_{MAT} = TRL\ 6$$

5 Urban Data Missions

5.1 Description

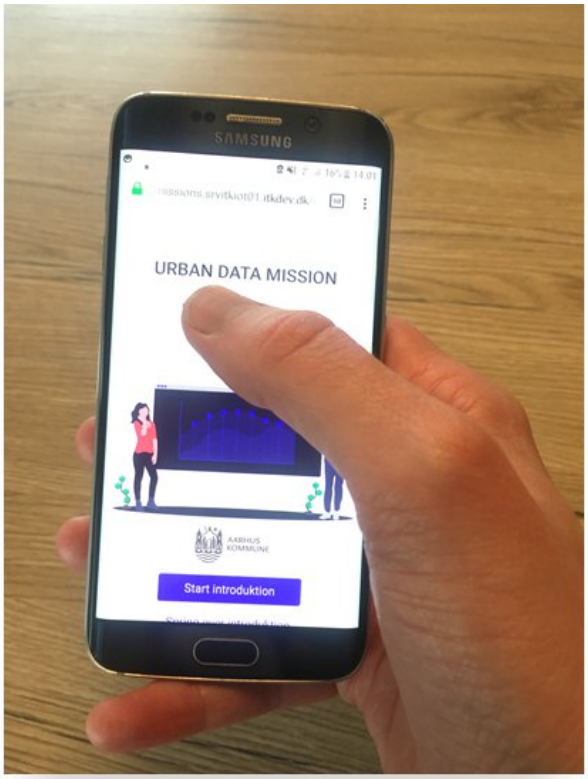

Figure 12: Urban Data Mission splash screen

Urban Data Mission is a citizen-science platform, which invites citizens on "data missions", where they can learn more about the environmental condition of their city.

The missions in Urban Data Mission can be well-defined with specific tasks to carry out e.g. "find the healthiest route to your school" or more open, where citizens can explore the topic that interests them the most e.g. monitoring green areas of the city. To carry out the missions on the platform the citizens can borrow a sensor-kit from the municipality, so they can carry out their own measurements, however the Urban Data Mission platform also contains data from official environmental sensors of the city. This enables the citizens to compare and correlate data within the Urban Data Mission platform. When carrying out the missions citizens have an online diary (Mission log) as part of the mission, where they can capture their own insights and observations e.g. "I think the reason for the high pollution levels in this area of the city is due of the many trucks driving on the main road".

By connecting Urban Data Mission to the IoTCrawler search-engine the citizens can find the relevant data sources and compare them to each other e.g. compare air quality levels between two different neighbourhoods in the city or even across different cities in EU. The societal benefit Urban Data Mission is aiming for is to

increase data literacy for citizens and include them in a more democratic way into the development of the Smart City.

Below in Figure 13 there is an example of the data view and mission log, which gives alerts and where users can keep a data diary of insights about their data mission:

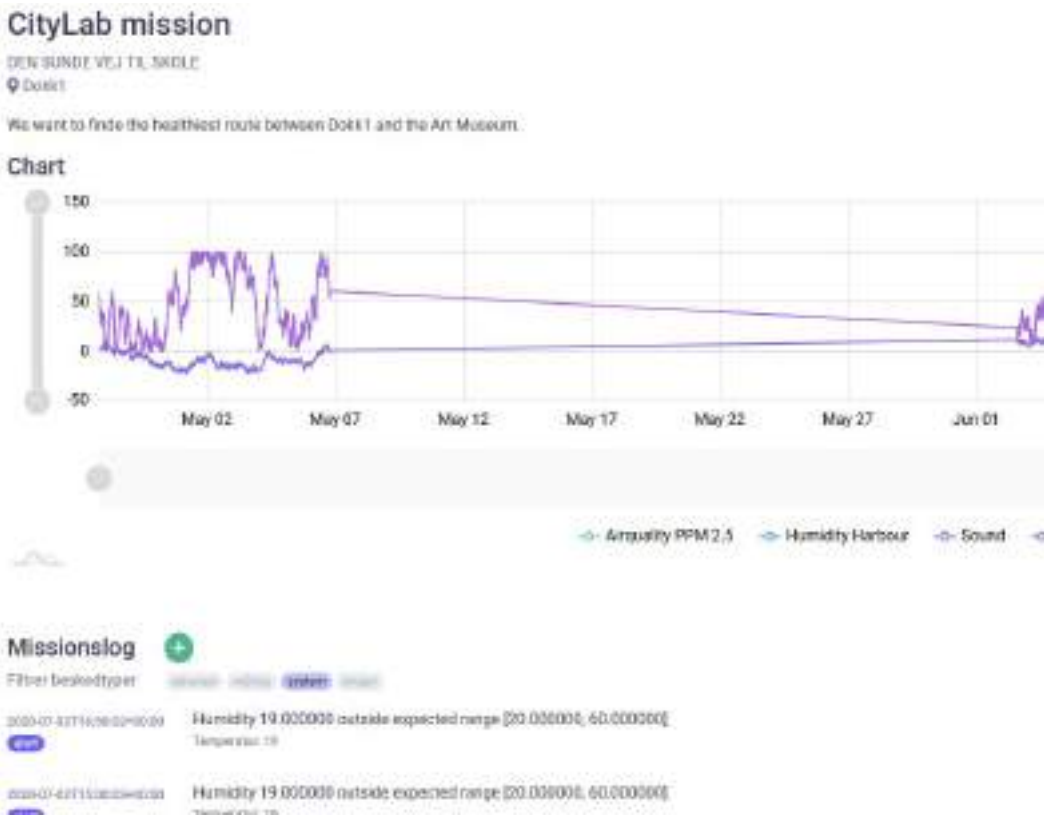

Figure 13: An active data mission in the application

Below in figure 14 is an example of the map view of different data missions taking place in the city:

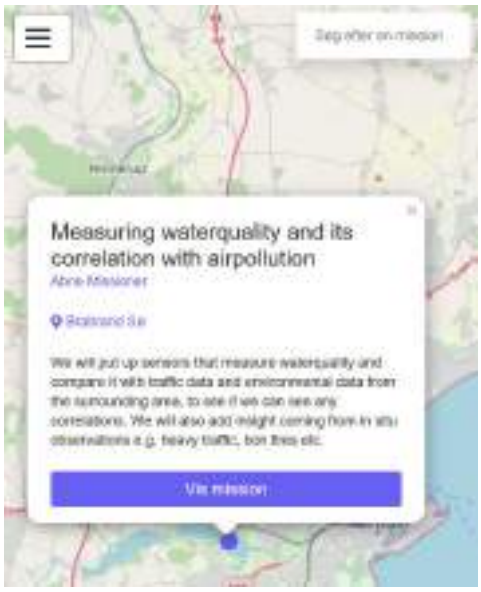

Figure 14: Map View in Urban Data Mission

5.2 Implementation Details

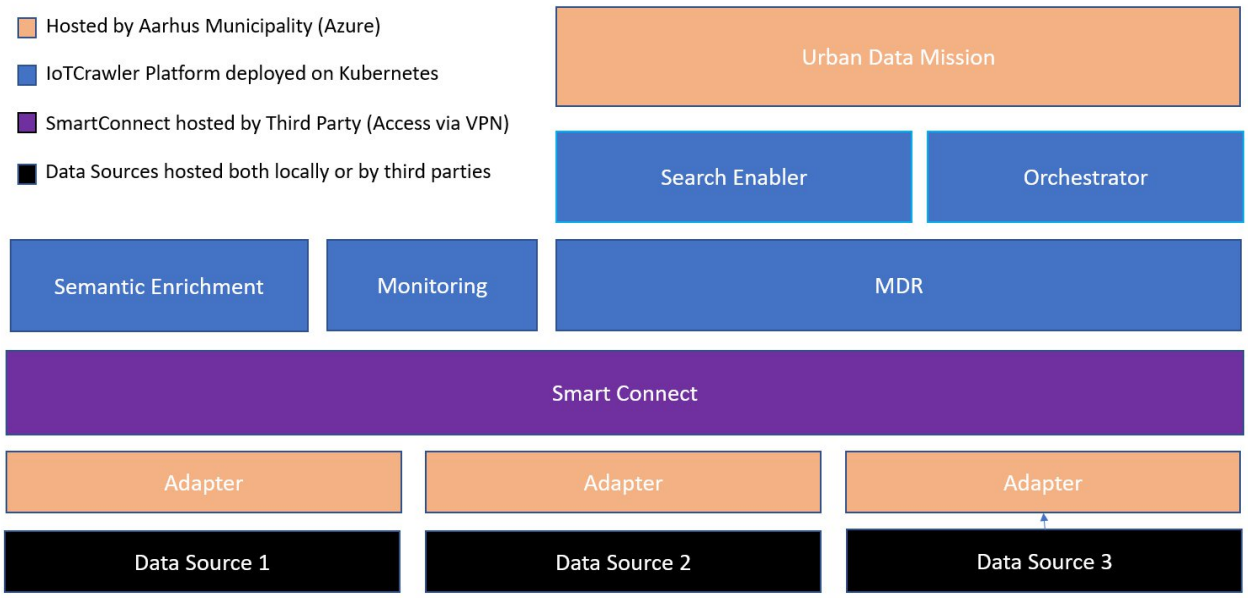

Figure 15: Architecture of Urban Data Mission MVP.

Above in Figure 15 the architecture of the Urban Data Mission MVP is presented. The application layer is the first orange box, below it in the blue boxes are the IoTCrawler components. The layers following this (purple, orange and black)

handles the process of getting data into IoTCrawler. Each component and their interactions will be described in the following paragraph.

The Urban Data Mission application searches for and subscribes to the relevant data in the MDR via the Search Enabler and Orchestrator; The Search enabler is used to search for sensors by their type and provides the correct id, which is used via the Orchestrator to make the subscriptions. The data comes from relevant environmental monitoring platforms and devices hosted by Aarhus Municipality and third parties. For instance, Smart Citizen Kit devices from SmartCitizen.me are used for engaging the users in the urban data missions. For each data source an adapter has been developed that brings data into the SmartConnect tool developed by AGT[5]. SmartConnect can identify the type of sensor and semantically annotate it. The monitoring component within the IoTCrawler deployment provides fault detection and recovery meta data via the subscriptions. This metadata is shown in the mission log of the data mission and on the graph view. The semantic enrichment component provides the application with Quality of Information data that is displayed when a user searches for sensors to add to their Data Mission, this way it can guide their choice for selecting the data sources.

5.2.1 Deployment

The components from the IoTCrawler framework are hosted in a Kubernetes Cluster following the same principles as described in chapter 8.

The application and the adapters are running on an Azure server outside of the Kubernetes cluster and managed by Aarhus Municipality. The SmartConnect tool is hosted by AGT and is accessed via a VPN connection.

5.2.2 Source Codes

The Urban Data Missions source code can be found here:
https://github.com/itk-dev/urban-data-missions/

Adapters for the MVP can be found here:

[5] The SmartConnect tool will be described in detail in D7.4

https://github.com/itk-dev/iot-crawler-adapter/blob/main/README.md#iot-crawler-adapter

5.3 Technical KPIs and evaluation

A number of technical KPIs have been identified for Urban Data Missions MVP according to specifics of this application scenario. Table 7 represents a list of KPIs as well as measured results and their explanations.

Table 7: KPIs for the Urban Data Missions MVP

#	KPI and Category	Components involved	Rationale	Results
1	The MVP should be able to collect data from at least two other IoT network or platforms to demonstrate the interoperability of IoTCrawler. The KPI for this will be two have developed at least two adapters to different data sources. (Compatibility)	Adapters, SmartConnect, MDR, Orchestrator, Search-Enabler	The idea behind the concept is to show that the MVP can make it relatively easy for citizens to make use of open data provided by the city, themselves or any third party that is connected to a global IoT Search engine. There we want to show this capability by addint at least two	3 adapters have been created: one for Smartcitizen.me, one for the City of Aarhus' Loriot server (containing data from LoRa sensors), and the last one is an adapter for experimental environmental sensors set up in the city by a local startup.
2	Users should be able to search for sensors with human understandable terms instead of sensor id's E.g. "Temperature". The KPI will be either True or False. (Usability)	Search-enabler, MDR, SmartConnect.	We want to democratize IoT and the Smart City, so therefore it is important to go beyond code and sensor id's and present the information in a more human-readable way.	True. This was enabled via the SmartConnect tool, where the data was annotated with the corresponding sensor and property types, which made it possible to search for this via the search-enabler in the MDR
3	Users should be able to have access to Quality of	Semantic Enrichment,	The rationale behind presenting users with	True. The Semantic Enrichment component

	Information when selecting sensors for a data mission. KPI is either True or False. (Functional suitability)	MDR, Search-Enabler	Quality of Information is the provide an additional layer to discuss when selected data sources for a data mission and this way increase the data literacy skills of citizens involved.	delivers Quality of Information details in the application's search interface and in the listing of selected sensors for the urban data mission.
4	If a fault occurs during a data mission the users should be alerted, and a calculated value should be presented. KPI is either True or False. (Reliability)	Monitoring, Semantic Enrichment, MDR	A good way to learn about sensor data is also to know what happens, when it fails to do so. We want to be transparent about this process by giving alerts to the users and showing calculated values in the mission log and in the graph view.	Partially True. The Fault Detection works, however as of the time of writing this the Fault Recovery experiences an issue in the training phase in the online environment. It works in a local setup. Besides fixing this in the next iteration we also want to be able to add virtual sensors.
5	Users should be able to view their sensor data in the application without too much delay, when having deployed a sensor in the field. The KPI for this is <30s. (Performance efficiency)	Adapters, SmartConnect, MDR, Search-Enabler	The rationale behind this is to give the users a good onboarding experience, when creating their missions. The rationale behind the 30s is based on the anticipated time it will take to setup a mission in the application for the user with title, description, and location before ending at the stage where they search and add the sensors to the mission.	Currently above 2 mins because of SmartConnect's update frequency. This time can be lowered in next iterations.

6 Room Booking

6.1 Description

This MVP has been developed to showcase how to leverage an IoTCrawler ecosystem through an IoT application to solve the problem of room booking based on real time data insights to make a university campus truly smart without the need of any manual intervention.

For any study/business meeting, lecture and project work, there is always a need to book a room in advance (usually via any organizational email integrated room booking system) to avoid any hassle, chaos or interruption by/to others. However, it has lots of flaws in certain scenarios like assuming all rooms are showing booked but in realty some of these are not occupied/used like shown in Figure 16. So, to validate if a booked room is being used at that point in time, it needs manual intervention & effort spent in different directions. Therefore, by digital intervention one can know, easily & in a hassle-free manner, the real time availability of a room without interfering & interrupting others. Otherwise, this leads to loss of resources productivity, waste of time, false impression & efforts. Additionally, there is a low level of infrastructure resource utilization happening due to the lack of real time insights & lack of data to generate useful insights over the period of time to analyse situations & demands in a better way to take well timed and informed administrative actions. Therefore, highly optimized utilization & management of infrastructure and ability to serve custom student/staff requirements is a need for future smart university campuses.

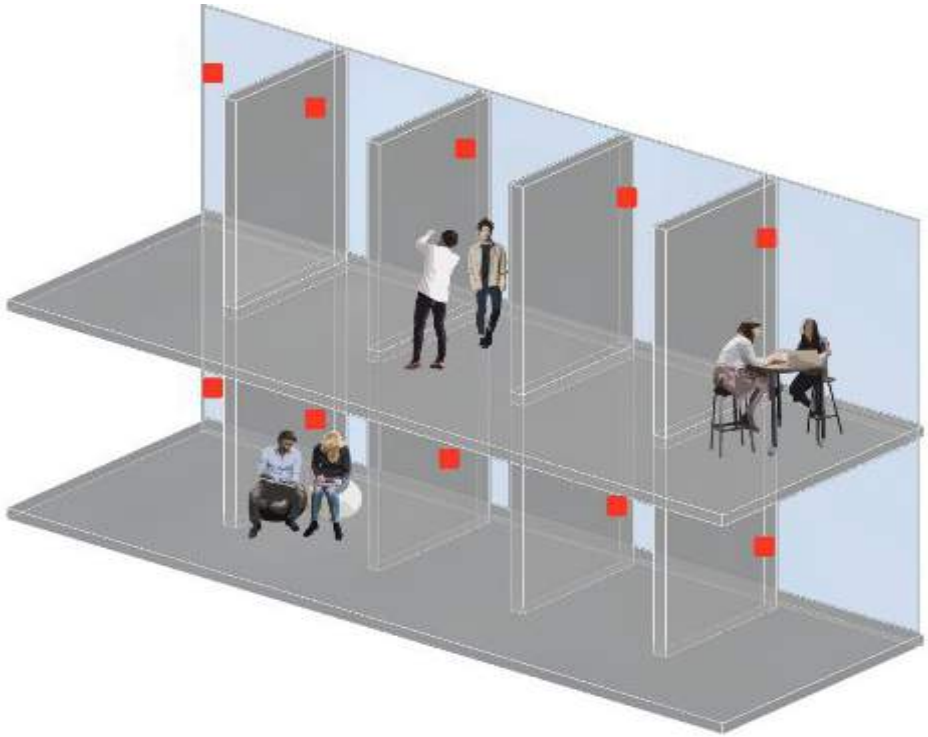

Figure 16: University Campus Infra – Per Floor Meeting Room View

The above problem scenario is addressed by the Smart-Room-Booking MVP, which is a digital intervention, that consists of a mobile app front-end and a dynamic dashboard UI that calls IoTCrawler NGSI-LD standard compliant APIs to access real time data context (generated by sensors) for different events and actions associated to booking management. To generate real time data context associated to physical environment of related room infrastructure, each room is installed with different types of IoT devices i.e. sensors such as humidity, temperature, light, motion detection (indicates physical presence) context. This real time data is fed into the IoTCrawler for semantic context and then the front-end application fetches the real time information from IoTCrawler to provide semantic and relational information for an end user to take certain actions related to room booking in real time.

This MVP concept will enhance the productivity, infrastructure utilize optimization and enables better decision making for the different stakeholders. Thus, offers small footprints on how to transform a traditional university campus into a modern smart campus with the help of the IoTCrawler framework.

6.2 Implementation Details

The architecture is shown in Figure 17. This MVP consists of three layers

- ***IoT Edge*** layer which consists of Lora Sensors and gateways - to generate real time data or useful insights of the physical environment using different types of sensors.
- ***IoT platform*** layer which consists of IoTCrawler framework- to collect, allow semantic enrichment & search, transform, process & store the data/metadata context and
- ***IoT Application*** which consists of mobile App & web services connected to IoTCrawler - to provide a mobile or web-based UI to allow interfaces for the search and query of metadata and data context for the realization of an end user related booking management task. Please see in-depth explanation in next related "Deployment" section.

This MVP offers mobile app & web interface to search information in real time. In the backend this MVP uses IoTCrawler interfaces & components like Orchestrator & Search Enabler to search room associated meta data & data (like specific sensor for specific room or all sensors of a given room or aggregated information of all rooms with all sensors etc.) context and MDR to query and subscribe and push and federate sensor data (like value of humidity, temperate or motion detection sensor, timestamp value etc..) for different physical contexts in real time like which rooms are available and which are occupied currently or what is the current environment status within a room or if someone is present in the room or not. Additionally, this MVP can leverage IoTCrawler related NGSI-LD compliant API interfaces for its temporal queries and geo-spatial query requirements to see the location of rooms. Though, in a small campus context it has little significance to navigate to the exact location of the room in a large campus. However, for multiple geo-distributed locations related informational queries, it becomes very relevant & significant.

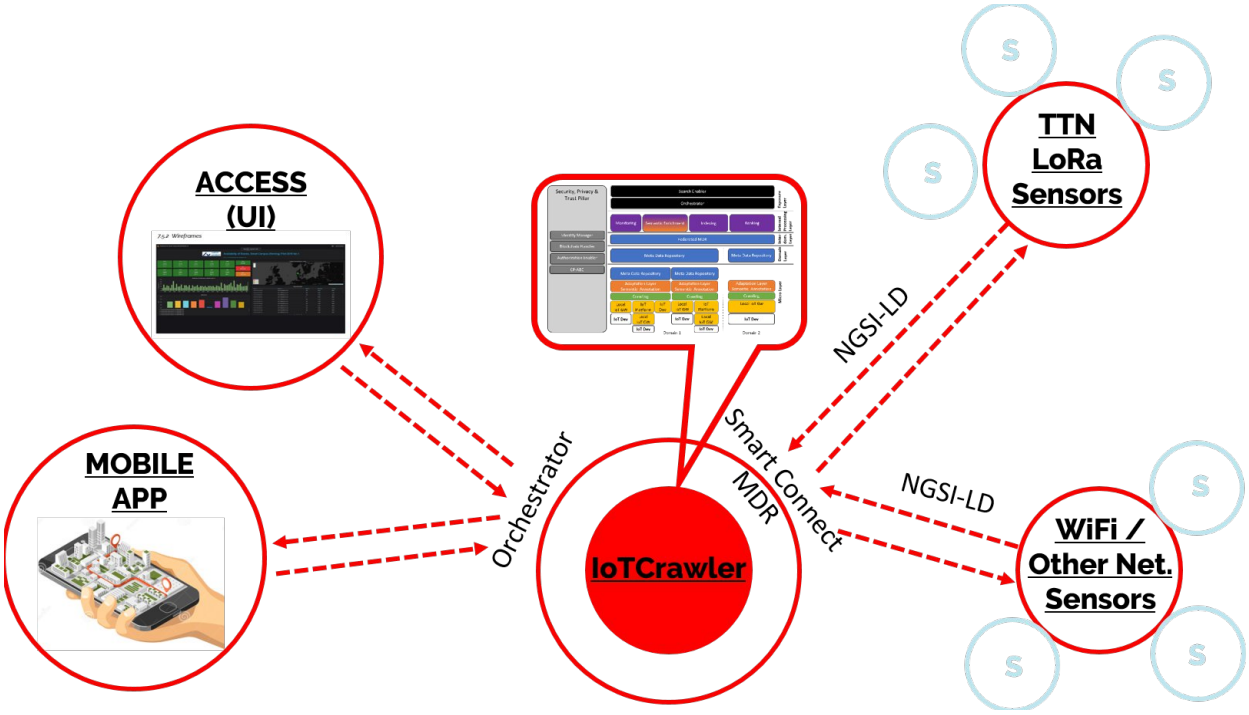

Figure 17: High Level Solution Architecture of Smart Room Booking MVP

6.2.1 Deployment

The Smart Room Booking MVP's deployment architecture is shown in Figure 18. The processing of this MVP consists of three layers given as follows:

(1) ***IoT Edge layer***- This layer is responsible for generating real time data or useful insights of the physical environment using different types of sensors (humidity, temperature, motion detection sensors) that are installed in each room. The sensors capture the environmental data and sends this data to the IoT Gateway (over Lora radio interfaces) which also acts as a MQTT server that is running inside the university campus. This gateway sends the data to SmartConnect service which in turn feeds the data into the local IoTCrawler after semantic transformation. The SmartConnect tool offers IoTCrawler related semantic abstract (metadata and data context) generation, based on IoTCrawler core information model i.e. NGSI-LD compliant context, in an automated manner. The smart connect service & local instantiation of IoTCrawler is running in the (AGT) partner's premises data centre.

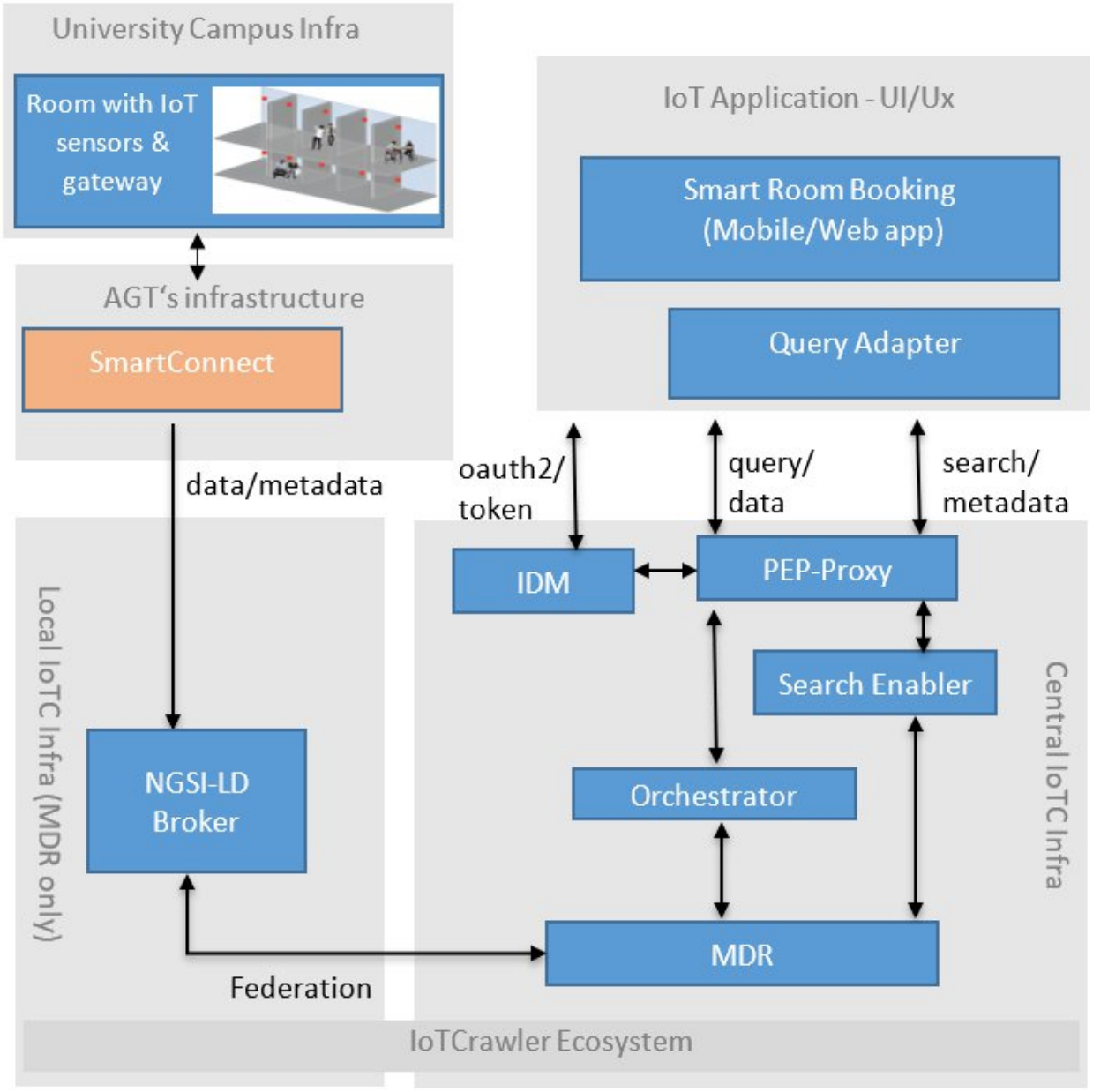

Figure 18: Smart-Room-Booking MVP Technical Architecture

(2) ***IoT platform layer*** - This is the IoTCrawler core framework that comprises of different components like MDR, Orchestrator, Search Enabler, IDM, Ranking & Indexing etc. This layer is responsible for performing certain IoT functions on our data like to collect data through MDR data federation, indexing of data for fast retrieval, enable semantic search for data or metadata, allow subscription of our data and metadata context, transformation of raw data into enriched semantic

annotated data i.e. NGSI-LD data based on IoTCrawler core modelling, persistence of the data and metadata context for real & temporal query requirements. For security, the platform consists of Identity Manager (IDM) service which provides OAuth2 based JWT (Json web Token) authentication and authorization to access the APIs and resource protected behind those APIs. The MVP calls this IDM service to get the token based on the provided credentials and use this token in the IoT platform service REST API calls for authentication and authorization validation at the platform side. The orchestrator and Search enabler is already offering HTTPS interfaces to be accessed by MVP and it offers the required data protection (e.g. PKI based encryption) mechanism in standard compliant way.

(3) ***IoT Application*** - In the end, application layer consists of mobile app (running in android & iOS-based mobiles) & web based view (built based on Grafana accessible through a browser) which acts as UI for the end user. The backend services of mobile and web apps is running in university cloud server. It is used to provide mobile and web-based UI to offer the search & query of meta data & data context for the realization of an end user related booking management task such as to display graphically all rooms if they are currently vacant or occupied, room booking flow, room/sensor specific captured environment parameters view etc. The related UI view is shown in Figure 19. The query adapter component of this layer help fronts end UI (mobile/web) services to perform wide range of queries of data & meta data based on the IoTCrawler search engine in real time.

Figure 19: Smart Room Booking Application (UI/UX) View

6.2.2 MVP Functional Flow

Below figure 20 is showing the sequence flow of Smart Room Booking MVP functionality. Every service call from the MVP to IoTCrawler components needs a JSON Web token (JWT) issued by Auth Enabler as per the OAUTH2 flow. This token is then embedded in each service request's HTTP header by the MVP. The platform service components is forefront by the PEP-Proxy filter that intercepts all the incoming request and validates the token with the IDM (Identity Manager i.e. Auth Enabler). The validated token request then allows to be processed by the relevant components (like Search Enabler & Orchestrator) in this case.

The event is triggered when a user perform room booking related tasks in the MVP. Then the MVP in turn initiates the request (after following token validation flow with IDM and PEP-Proxy explained above already) to fetch the information about all rooms available (from MDR via Orchestrator) in the campus to perform booking tasks. For each room, the MVP then collects room specific sensor level information

(via Search Enabler which in turn make subsequent calls to MDR in a semantic relationship manner). This room environment sensing information enable the MVP to take certain decisions about the room e.g. if the room is available for booking or already occupied by someone in real time.

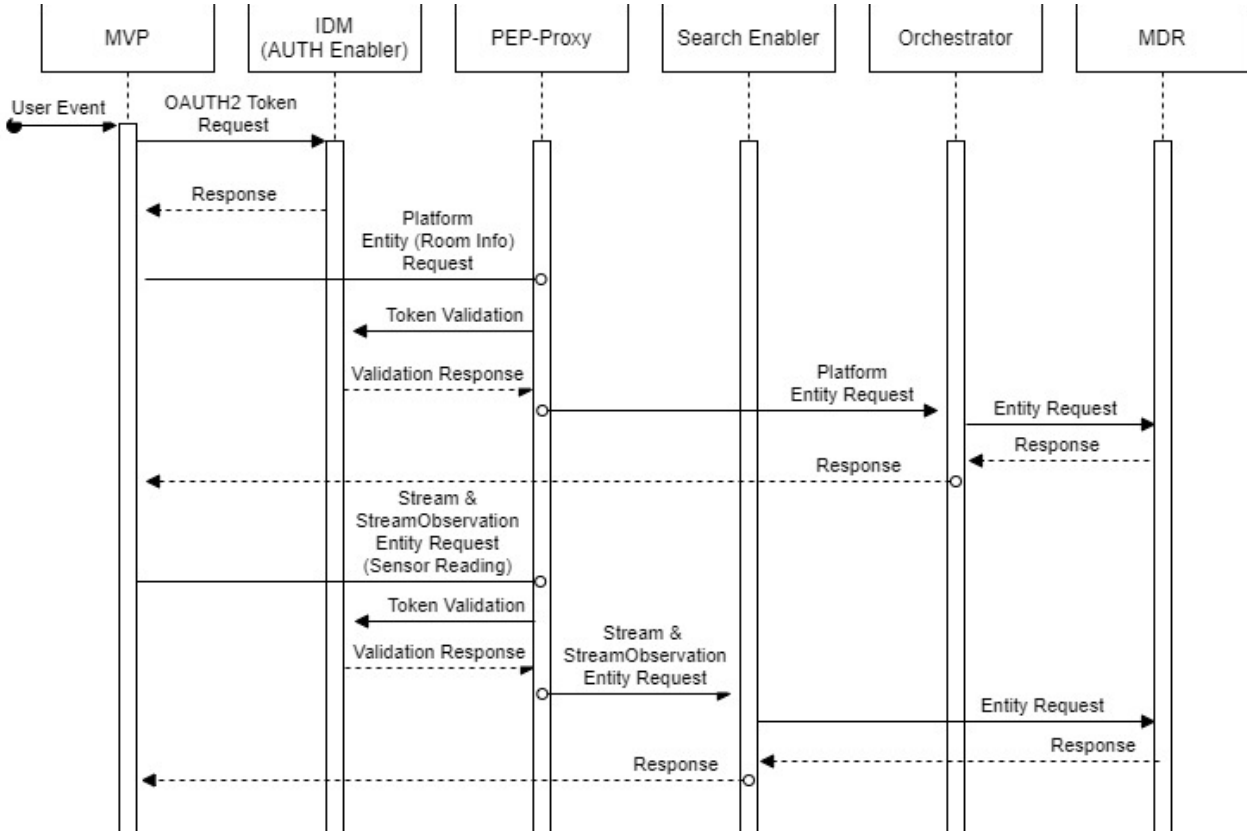

Figure 20: Sequence Diagram for Room Booking MVP

6.2.3 Source Codes

The source code for the MVP is available under

https://gitlab.iotcrawler.net/demontator-au/smart-room-booking/

6.3 Technical KPIs and evaluation

Room booking related KPIs have been captured in the table 8 below. There are 10 iterations performed to measure the KPIs. The functional flow related to KPIs measurement consists of platform entities (5 platform entities of size ~5KB) fetched from MDR directly. Additionally, there were entities (around 30 entities, in total, related to sensor, stream and stream-observation having size ~13.5 KB) fetched from

search enabler via GraphQL interface to get data & metadata in semantic relationship manner.

Table 8: KPIs for the Room Booking MVP

#	KPI and Category	Components involved	Rationale	Results
1	Latency on finding information about specific room with related sensors and measured values, based on which its occupancy can be decided, should be less than 5 seconds. (Functional Completeness & Time behaviour)	Search Enabler	Specific room related information is useful to decide the occupancy availability of that room very quickly without the need of manual intervention.	1.587 seconds
2	Latency on finding information about what all rooms (booked or empty) are available in campus for meeting purposes should be less than 2 seconds. (Functional Completeness, Correctness & Time behaviour)	MDR & Orchestrator	Room booking app should provide information about all meeting rooms by discovering them in the campus in real time.	0.4096 seconds
3	Total latency to generate useful information for an end user to take decision for a room booking, that includes app functional processing, network latency & all data fetching latency from IoTCrawler, should be less than 5 seconds. (Functional Completeness, Correctness & Time behaviour)	MDR, Orchestrator & Search Enabler, Application itself	Room booking app on-demand needs to sense all rooms for physical environment in order to decide about the occupancy of room booking is available or not as quickly as possible.	2.2191 seconds

All KPIs described above is of high importance in nature. The IDM related OAUTH2 token validation latency overhead is not accounted for in the above KPIs in order not to clutter the true picture of KPI measurements.

7 Machine Monitoring

7.1 Description

Industry 4.0 has been one of the major highlights of the year, 2020 in technological space. It has been driven by innovations and rapid technological advancements in the Internet of Things (IoT). A typical Industrial Internet of Things (IIOT) use case consists of multiple sensors, actuators, controllers and digital platforms. In the most cases, these are employed for monitoring, automation, optimization and maintenance of machines, logistics, processes, data etc. in an industrial setup. The data generated from these IoT devices and applications is contextual. It shows the characteristics and attributes of the applications, machines and their surroundings.

Figure 21: Manufacturing Machine IoT Network

The IoT systems and applications used in the industries are desired to be real-time and reactive. The IoT applications based on the traditional non-context aware databases are not efficient in that regard. Therefore, a Minimum viable product (MVP) named Machine Monitoring which utilizes more advanced spatial, temporal and keyword queries to resolve and discover real time information, is designed and developed by digital worx under the IoTCrawler project.

The objectives of the Machine Monitoring application to facilitate real time machine data securely. This data will include machine status and data from all sensors embedded in a machine. For an example, typically a packaging machine on an industrial floor has red, green, yellow light sensors, welding bar temperature sensor, welding bar open/close sensor, packaging material status sensor, level sensor, machine on/off status. All of these data will be collected, formatted and sent to IoTCrawler framework. Their this data will be enriched and stored securely.

The machine data is very useful to monitor the performance of the machine and its components, predictive maintenance and optimization of the machine output. In a typical settings, this will concern to the floor manager, machine operators, maintenance engineers, and production managers. The MVP application will prove to be significantly useful for all its stakeholders.

It is also imperative to secure the data during transmission and storage. The machine data is quite critical for any organisation. The data breach and stealing might have significant consequences for an organisation. The IoTCrawler framework utilizes multiple tools to maintain the data integrity and privacy. The following section gives more information about the IoTCrawler security components used in the MVP.

7.2 Implementation Details

In the Machine Monitoring MVP, the controllers installed on smart machines in a manufacturing plant feed data to the MDR in the IoTCrawler. This data includes machine status, position, sensors details etc. The MVP application uses core components of the IoTCrawler framework. They are depicted in the Figure 22.

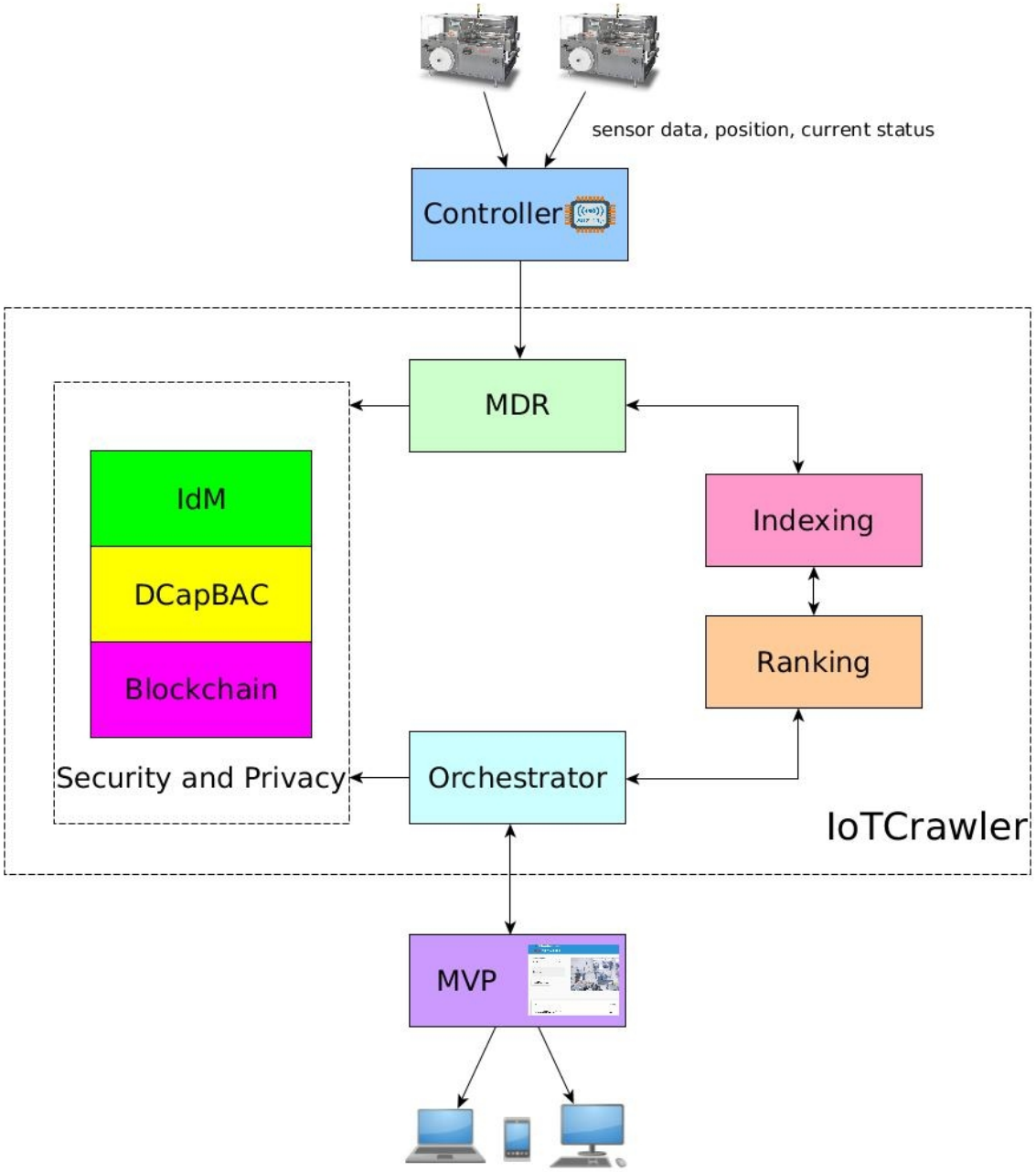

Figure 22: High Level Solution Architecture of Machine Monitoring MVP

The machine monitoring MVP application interacts with the IoTCrawler framework using the Orchestrator. It is the first contact point for it. The MVP application sends user queries and receive results from the Orchestrator. The results are displayed in the application with digital media such as graphs, table and charts.

The security is the core aspect of the IoTCrawler framework and the MVP application. The MVP application employs the security components of the

IoTCrawler framework. These components enable secure data access for controller and MVP application. The access policies are defined to put access constraints on data available in the IoTCrawler framework.

MVP Application

The MVP application is developed as a web application written in typescript using Angular framework. It is a Single Page Application (SPA). It interacts with a web browser by dynamically rewriting the current web page based on the data received from the server. These are the following benefits of the SPA architecture.

1. Fast and responsive
2. Linear user experience
3. Caching capabilities
4. Less bandwidth

Figure 23: Comparison of Traditional and Single Page Application

The same application can be compiled as an Android or IoS mobile application. It has very intuitive interface and material design[6]. The web application is modular by design. The MVP can also be built as a docker image and run on any platform. We have deployed the MVP on Kubernetes cluster running on Google Kubernetes Engine (GKE). There, it runs in an isolated container. It is exposed outside the cluster using ingress resource and Nginx ingress controller is used as load balancer.

[6] https://material.angular.io/

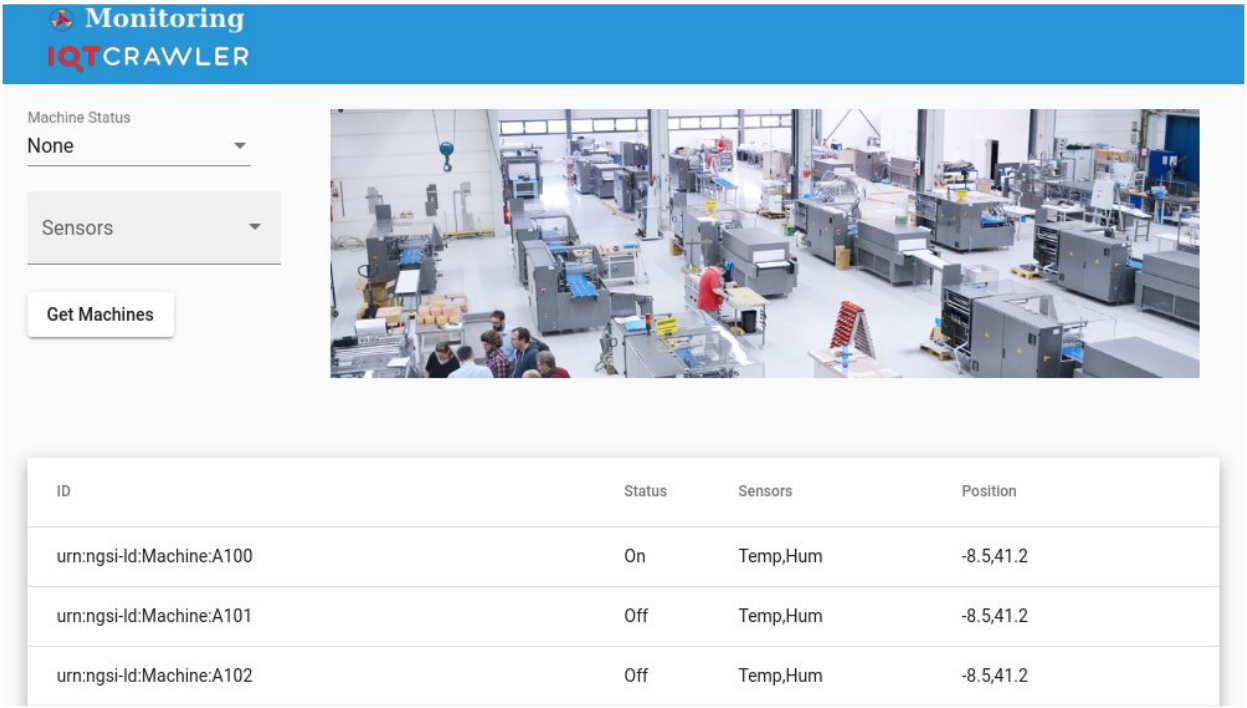

Figure 24: Machine Monitoring MVP Interface

The application is not connected to any database. It does not store any data about IoT devices and the application. It can be used to make different types of queries to access context data stored on the IoTCrawler platform. Once the data is received, it is illustrated on the application with intuitive graphics.

Controller

The controller is an embedded system which is installed on an industrial machine in a manufacturing plant. The controller houses a microcontroller, input/output peripherals, memory and WiFi communication module. It does not run an operating system. It executes a single threaded program which iteratively checks, collect and send sensors data to IoTCrawler. The controller collects machine data such as machine status, red, green, yellow light status, welding temperature etc. This data is formatted and sent to Orchestrator in the IoTCrawler platform. The controller has small footprint and low power consumption which are ideal features of an IoT device.

IoTCrawler Core Components

The MVP uses 5 core components of IoTCrawler. They are Orchestrator, Ranking, Indexing, MetaData Repository and security components. The machine related data

is sent to the MDR using the REST API endpoints. The MVP application then can be utilized to get and visualize the machine data from the Orchestrator.

7.2.1 Deployment

There is a separate IoTCrawler instance for the machine monitoring MVP deployed on a Google Kubernetes Engine (GKE) cluster (described in Section 8). The machine monitoring application also runs on Kubernetes cluster.

7.2.2 Source Codes

The source can be found in the following gitlab repositories.
https://gitlab.iotcrawler.net/dw/machine-mvp

7.3 Technical KPIs and evaluation

The machine monitoring MVP deals with data from the machines installed in an Industry 4.0 manufacturing plant. Typically, a plant host thousands of machines which function around the clock. These machines are embedded with sophisticated sensors. The volume of the data generated by these machines are quite high. It is imperative for Production managers, floor managers and maintenance engineers to get virtually real time information of these machines. Because a failure in dozens of machine could also have significant impact on the production of that day.
Therefore, a constant machine monitoring is required to mitigate the risk and take preventive decisions. The MVP aims to achieve the real time monitoring. Hence low latency of responses is quite critical in the machine monitoring application. This is reflected in the KPIs. A thorough testing was done on the IoT Controller and MVP application to have low latency.
The MVP application was designed and developed keeping the performance as the core principle. It is a single page application. Consequently, it has multiple benefits[7]. The web resources such as HTML, Scripts etc. are only loaded once throughout the

[7] https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

lifespan of the application. Which has significant improvement on the performance of the application.

The table 9 list the KPIs required by the Machine Monitoring MVP application from the IoTCrawler platform. The KPIs are use-case specific and are affected by certain IoTCrawler components.

Table 9: KPIs for the Machine Monitoring MVP

#	**KPI and Category**	**Components involved**	**Rationale**	**Results**
1	Unauthorised access to the machine data should be prohibited in the IoTCrawler framework (Security)	Identity manager, Blockchain	The data breach of the machine data on the IoTCrawler framework might have severe consequences for an organisation	Reached by using identity manager and access policies in Blokchchain
3	Time taken to add machine data* in the IoTCrawler frame by the IoT controller should be less than 1000ms (Performance Efficiency)	MDR, IoT Controller	For a real time monitoring solution it is imperative to have low latency	Response time was measured to update the machine data* on the IoTCrawler framework average: 670ms
3	The time taken to fetch data* for a given machine ID from IoTCrawler framework should not be larger than 1000ms (Performance Efficiency)	Indexing, Ranking, Orchestrator, MDR, MVP App	The MVP is aimed to show real time machine data*	Response time was summarized for multiple requests sent to the IoTCralwer framework to fetch machine data* average. 630ms
4	The time taken to fetch data* of all machines having Temperature sensor from IoTCrawler framework should not be larger than 1000ms (Performance Efficiency)	Indexing, Ranking, Orchestrator, MDR, MVP App	It is critical to provide machine data in real time to able to take decision on preventive maintenance and production quantity	Response time was summarized for multiple requests sent to the IoTCralwer framework to fetch machine data average. 620ms

5	The time taken to fetch data* of all machines based on machine status from IoTCrawler framework should not be larger than 1000ms (Performance Efficiency)	Indexing, Ranking, Orchestrator, MDR, MVP App	It is important for production manager to know the number of machine working and not working in real time to take quick decision on production volume	Response time was calculated for multiple requests sent to the IoTCralwer framework to fetch machine data* average. 610ms
6	The time taken to fetch data* of all machines based on queries such as machine type and sensor type Accelerometer from IoTCrawler framework should not be larger than 1000ms (Performance Efficiency)	Indexing, Ranking, Orchestrator, MDR, MVP App	It is crucial for floor manager in a manufacturing plant to able to fetch machine data take quick decision	Response time was calculated for multiple requests sent to the IoTCralwer framework to fetch machine data* average. 790ms

* The machine data in the MVP includes the machine ID, types of sensors e.g. Level, Accelerometer, Temperature, Humidity etc, location,

8 Deployment

The deployment mechanism of the MVP components is similar to the deployment mechanism of the core IoTCrawler components: using the Gitlab CI/CD pipelines source codes are built to container images, which when deployed in the Kubernetes engine hosted in Google Cloud (see Figure 25).

Figure 25: CI-CD pipeline status, as shown in Gitlab dashboard.

Each MVP is composed of one or several containerized applications, which interact with IoTCrawler components within the cluster. To avoid side effects, some MVPs are using dedicated instances of the IoTCrawler platform, while other MVPs share the same set of IoTCrawler components.

The Gitlab/Kubernetes stack was selected to make all IoTCrawler platforms and MVPs highly available and scalable. The horizontal container (pod) scaling feature can be used for the components and applications on the Kubernetes cluster. Which means that number of pods running a component is dynamic. It changes based on the CPU and memory usage of the pod. If the traffic increases for a component or the application, the number of pods are scaled up to handle the traffic efficiently. This method controls the traffic congestion. Interaction from the outside world, with the different components contained in the cluster, is performed though different Ingress Points in the Kubernetes cluster. This is also configured and set up thanks to Gitlab's CI-CD tooling, which provides means to configure the SSL Certificates required for the different Ingress points of the cluster, which are connected to the respective internal components, providing HTTPS termination, as well as other services, such as load balancing.

As we can see in Figure 26, the cluster in which the components are built and tested, as well as many of the MVPs components are hosted, is a Google

Kubernetes Engine instance, running two nodes of e2-standard-8 machines, with eight virtual CPU cores and 32GB of RAM.

Cluster	iotcrawler
Node version	1.15.12-gke.20

Size

Number of nodes	2
Autoscaling	Off
Node zones	europe-west1-b

Nodes

Image type	Container-Optimized OS with Docker (cos)
Machine type	e2-standard-8
Boot disk type	Standard persistent disk
Boot disk size (per node)	100 GB
Boot disk encryption	Google-managed
Preemptible nodes	Disabled
Maximum Pods per node	110 (inherited from iotcrawler)

Figure 26: IoTCrawler cluster specifications in GKE.

9 Best Practices

This section summarizes the practices of using IoTCrawler for different MVPs and provides a guideline how a potential MVP owner should develop/integrate his application on top of the IoTCrawler platform. In this guideline we assume, that the IoTCrawler components are already deployed and integrated together, so the MPV owner builds his application using the required IoTCrawler services. By MVP owner (application developer) here we consider a technically skilled person.

MDR

The MDR is the central part of the architecture and, as such, it is used in all of the MVPs. Its purpose is to act as a data broker, providing a central point from which to query information and where clients can subscribe to get notifications. In order to be able to receive those notifications, clients must provide an URL for an REST API endpoint where notifications will be pushed. That REST endpoint must be accessible and available to the MDR, in order for those notifications to correctly reach the client.

The MDR can be deployed as a federation of NGSI-LD brokers, offering some beneficial features such as improved privacy/security and improved scaling/performance. Some MVPs don't require such complexity (at least at the time of writing this document), but others do; such is the case of as the SmartHome MVP do. The federation of brokers allows to have a central point (the federation broker) to act as a single broker representing all the information contained in all of the remaining ones (federated brokers). This way, even though information is actually stored in one of the federated brokers, IoTCrawler can still leverage that information through its different components (Indexing, Ranking, Semantic Enrichment, etc...).

MVP owners should not integrate their application with the MDR, but through the Orchestrator. If such integration was still a requirement, such as addressing a specific need not already covered or solvable through the means provided by IoTCrawler, the integration should be performed through the Policy Enforcement Point (PEP-Proxy) provided by the DCapBAC component, ensuring in this way that the interaction would take place within the policies set in place in the Policy Administration Point (PAP), further described in the following Security best practices section.

The interaction with the MDR would follow the usual steps required for a regular NGSI-LD interaction, preceded by the security steps described in the next section. In

the case of performing a subscription, special care must be taken to ensure that the MDR is able to reach the endpoint to which the notifications should be addressed.

Security

The security pillar comprises both Identity Management, Capability Manager, XACML framework and Blockchain. These components must be used together for authorising users or services to access to the information stored in the IoTCrawler framework either for reading it, or for integrating new sources of information.

These components have been used in the Smart Parking MVP for integrating different sources of information with two different permissions, this way, RPZ information and private parking site information is available with two different access control policies, preventing non-authorised users to access this information. Additionally, these components have been used for the Smart Home Crawler with a similar approach for defining different privileges in the access of the information discovered inside a home domain.

In order to integrate their applications with IoTCrawler, MVP owners will have to follow these steps:

1. Define and create the user-base in the IdM, together with the relevant attributes that will later ease the definition of policies, such as roles, organizational structures etc…
2. Define the policies that will rule the authorization mechanism, following the XACML standard, through the Policy Administration Point. These policies will define which queries can be made to the different REST API endpoints that are protected in IoTCrawler, such as the MDR and Orchestrator.
3. In order to interact with protected components, the application will have to provide a Capability Token as a header on the REST request to the protected endpoint. This token is provided by the Capability Manager upon request, by specifying the desired endpoint and operation to be performed, together with the Identification Token obtained after login-in with the IdM.

Indexing

The Indexing component enables applications and services to discover IoT data streams and sensors more efficiently and allows clients to subscribe to StreamObservation entities from the MDR (or via the Orchestrator) when streams of interest have been identified. Indexing of entries from an MDR begins as soon as the

Indexing component is subscribed to that particular MDR. Meaning that currently, any previous entry in the MDR would not be indexed, and therefore should be taken into account when deploying the Indexing component.

For persistence, the Indexing component makes use of local database sharding, which allows the distribution of stream and sensor descriptions across multiple instances of the local store. Upon initial deployment, MVP owners should estimate the expected number of sensors and streams in their system and based on that, determine the number of required shards. Sharding configuration is done through the execution of the shard script which can be found under the "/MongoDB/scripts" subdirectory.

The indexing mechanism applied is based on geographical parameters, in the form of polygonal regions, each of which correspond to a country. If an MVP is limited to a particular country or region, then the polygonal coordinates of the regions for that country can be used instead, provided the format is in GeoJSON. The geoBoundaries[8] website provides boundaries on the level of country regions, by selecting ADM (Administrative Divisions) level 1 or above. For example, if the Smart Parking service operates within the region of Murcia, then the a GeoJSON dataset with ADM level 2 could be used. The new dataset must replace the dataset that is available under "/src/data/". Also, the script that reads the dataset uses the property name "ISO" to differentiate between regions. The MVP owner could wither replace the name of the field containing the region name to "ISO", or else rename the variable representing that field in the source, which can be found under "src/util/populateFeatureGeometry.js", to that being used in the new dataset, e.g. "shapeName".

Since entities stored in the Indexing component are indexed based on location, it is imperative that they have location attributes or are linked to an entity that does. In IoTCrawler ontology, location is directly bound to the Sensor, and so any entry for a Sensor must contain the location attribute. Also, according to IoTCrawler, IoTStreams are considered as a core concept and therefore, for Sensors to be discoverable, they must be linked to an IoTStream entity.

[8] https://www.geoboundaries.org/downloadSimple.html

Also, indexing is applied on a subset of IoTCrawler entities (IotStream, Sensor, Point, and QoI) based on geolocation, and parses a limited number of attributes for efficiency. In the case of:

- Sensors: "type", (GeoProperty) "location", (Point) "location", and "observes".
- Streams: "type", "generatedBy", and "hasQuality".

An MVP owner is not targeted to interact with the indexing component directly, but it still might be useful to have an understanding how the data can be optimized and which queries can be used to get the maximum performance of a data retrieval from IoTCrawler via Search Enabler or Ranking components.

Ranking

The goal of the ranking component is to offer a way to order the results from a search request. While this is crucial in large deployments, spanning over a large number of devices, ranking the result set might also be proven useful even in smaller use cases.

The ranking component takes as input an NGSI-LD query and an additional parameter containing tuples of ranking features and their weights. Currently the ranking component supports ranking of entities by their attributes (NGSI-LD Properties), for example, direct properties of iotStream entities. Queries are then forwarded to the underlying component, which is either an index or an MDR, depending on the deployment. Upon receiving the query results, the ranking component computes, for each result, its ranking score, based on the given features and weights. This score is then attached to the result as an additional property, which can then be consumed by the upper layer. The component is stateless, meaning it can scale across different servers. Applications can refrain from using the ranking in cases it's not required. In this case, applications specify only the query, without the ranking parameter, and the ranking simply acts as a proxy between the upper and lower layers, with minimum delay. In order to get ranked entities for their applications, app developers should:

1. Look up entities and their attached numeric NGSI-LD properties to be ranked by (e.g., in the MDR)
2. Define rank weights for these properties and encode them in the query to the ranking component
3. Optional: Encode rank weights in GraphQL queries to the Search-Enabler

4. Integrate queries in your app and process query results by the supplied rankScore property values

Search Enabler

The Search Enabler component is used by a number of MVPs (SmartHome, Urban Data Missions, Smart Energy, Smart Campus) for returning metadata and data via GraphQL interface. The power of GraphQL is in a query language composed out of data types defined in schema files. The core IoTCrawler data model (entities of the Iot-stream and SOSA ontologies) are already defined in Search Enabler schemas, but in case if MVP owner wants to introduce new domain-specific types used in his application (and expose them to others), then it is possible encode them into a separate schema file and to deploy own instance of Search Enabler[9] or to contact a platform administrator to apply the new schema file to already running Search Enabler.

To integrate the potential MVP application with search enabler the following steps are required:

1. Design and debug target queries using GraphiQL
2. Integrate GraphQL queries into the application

Let's describe the aforementioned steps in detail. The search-enabler is a first step from which MVP owner should start searching relevant data streams, from which his application wants to get data. For searching streams (sensors, and any other entities) the MVP owner should open the GraphiQL environment, which is available online[10]. After putting a pair of curly brackets "{ }", putting cursor in between and pressing "Ctrl + Space" user is able to see all the data types currently loaded into Search Enabler via schema files. After selecting a query type (e.g. streams, sensors, observable properties) from the list and putting a pair of brackets "()" it becomes possible to specify filters relevant for the selected type (e.g. id, alternativeType, offset, limit, etc.). Some filters are simple (e.g. id, label, offset, limit), others are

[9] https://iotcrawler.readthedocs.io/en/latest/getting-started/start-search-enabler.html

[10] http://search-enabler.iotcrawler.eu

hierarchical – representing a hierarchy of types associated with the target entity type. For example, user can filter streams by properties of sensors, by which these streams are generated. When MVP owner specified filters, the next step is to specify the output of the query by using the same hierarchical approach as was used for filters specification. It is possible to return all types of metadata stored in MDR only in one GraphQL query (but using offset and limit might be required). More details about encoded data types and possible queries are provided in the tutorial[11].

Then a query specification process is done, the query might be encoded into the app using different GraphQL client libraries (e.g. Apollo Client[12]). A simple GraphQL client for java is provided by the Core Library[13] which requires a GraphQL endpoint to be specified. Using a GraphQL interface (a client and queries) to interact with IoTCrawler, application developer can avoid using the NGSI-LD interface to getting the metadata from IoTCrawler. Being a GraphQL server, the Search Enabler has its own NGSI-LD client to Ranking and Indexing components (which represent data stored in MDR). For resolving a single GraphQL query the Search Enabler it automatically performs all the necessary NGSI-LD requests.

Orchestrator

The orchestrator component is used by a number of MVPs to mediate NGSI-LD requests from applications to further components of IoTCrawler (Ranking and Indexing or directly to MDR). Orchestrator is targeted to solve two problems:

- extend the push-based subscribe mechanism of Scorpio Broker by optional use the AMQP protocol to let applications run without any network configuration (relevant for Smart Home users or organizations)
- maintain the cache of active subscriptions to avoid duplicating subscriptions if all the critical parameters (entity, watched attributes, reference URL) are the same. This avoids broker to be overloaded by duplicating subscriptions coming from different applications (or the instance of the same application in

[11] https://iotcrawler.readthedocs.io/en/latest/tutorials/graphql_search.html

[12] https://www.apollographql.com/docs/react

[13] https://github.com/IoTCrawler/Orchestrator/tree/master/com.agtinternational.iotcrawler.core

time). Also, the orchestrator component has a potential to notify remote applications about changes, which might affect data quality. For example, orchestrator potentially might be used for notifying an application about virtual sensor creation, changing of quality of already subscribed streams.

In order to integrate the application with orchestrator, MVP owner have to perform the following sequence of steps:

1. Define of relevant streams to subscribe to (e.g. using GraphQL)
2. Encode a subscriptions requests logic
3. Encode and debug notifications handlers
4. Make the app more robust using environment variables

Let's describe all of these steps are describe in detail. Interactions with the orchestrator component becomes relevant when a application has identified a set of streams to subscribe to. For finding relevant streams application owner can used Search Enabler and GraphQL client described in previous section.

Having a list of relevant streams, the MVP owner need to make application to initiate subscriptions to them. For subscribing to streams, application code should perform an NGSI-LD subscription request to the Orchestrator component. The application developer can use the java library with NGSI-LD client[14] and a valid orchestrator's endpoint[15]. The NGSI-LD and GraphQL clients also bundled together into IoTCrawlerClient provided by the IoTCrawler Core library[16], which MVP owner can use for performing steps (1) and (2).

When subscriptions logic is implemented and tested, the MVP owner has to define notification processing handlers. In other words, application developer has to teach application to react on a incoming notifications with actual sensor data values. For example, the application can update it's state or run some analytics over the latest data. The default publish-subscribe mechanism expects a REST endpoint (a HTTP server) to be exposed by the application to receive notifications from the MDR. Exposing a publicly accessible endpoint requires network configuration, which is not

[14] https://github.com/IoTCrawler/Orchestrator/tree/master/com.agtinternational.iotcrawler.fiware-clients

[15] https://orchestrator.iotcrawler.eu/ngsi-ld/

[16] https://github.com/IoTCrawler/Orchestrator/tree/master/com.agtinternational.iotcrawler.core

possible in many situations. For avoiding that, application can instruct the Orchestrator component to receive all the notifications to his (orchestrator's) endpoint and forward them into AMQP-queue. Knowing the corresponding AMQP-queue name (received as a result of subscription request) the application needs to be connected to it as a client. The IoTCrawler Core library[17] for Java has the IoTCrawlerRPCClient - an implementation for doing that allows to create AMQP connections easily. After notification is received by the app (via HTTP or AMQP) the asynchronous callback defined on this step would perform a processing logic over the received data.

After finishing all the described steps, the application is ready for interactions with remote the Orchestrator component. To make the application more robust and flexible, it is recommended to introduce externally configurable variables for at least the following endpoints: orchestrator's NGSI-LD endpoint, AMQP server endpoint (if used) and GraphQL Endpoint (if used). Having these variables externally configured would allow to easy switch application between different IoTCrawler instances, which might be required in future.

Semantic enrichment

The Semantic Enrichment (SE) component of the IoTCrawler framework is responsible for the annotation of data sources with Quality of Information (QoI). This additional information provides an idea about the quality of a data source and provides an indication, if a data source is getting worse, e.g. by decreasing battery levels or misconfiguration.

In addition to the environment variables a data source should be annotated with some meta data information. This consists of an update interval, and a minimum and maximum value a data source can provide, e.g. reflecting the measurement range of a sensor or the usually measured interval and are described in more detail below with an example for a data source providing temperature data:

- qoi:max

  The qoi:max value defines the maximum value a data source should provide in its related StreamObservations received by the Semantic Enrichment. An

[17] https://github.com/IoTCrawler/Orchestrator/tree/master/com.agtinternational.iotcrawler.core

example can be a temperature sensor inside a building that is defined to deliver values up to 50°C or a sensor to provide the current capacity of a parking garage that should not go above the total number of spaces inside the garage.

- qoi:min

  The qoi:min value is the opposite of qoi:max and defines the minimum value that can be provided by a data source. For example, a minimum of 10°C for an indoor temperature sensor or 0 for the capacity of a parking garage. Together with qoi:max it is therefore possible to define the measurement range of a data source. Values outside from this range can be assumed as error or abnormal behaviour.

- qoi:updateInterval

  The qoi:updateInterval defines the regular interval when a new value from a data source should be provided. If a sensor should provide a value every 5 minutes this should be annotated with 300s.

- qoi:valuetype

  This defined the type of a measured value. It support types like integer or float but in case of type string it is also possible to provide a regular expression with qoi:regexp.

- qoi:regexp

  The qoi:regexp can be set if the valuetype is of type string. In this case the provided value can be checked for a regular expression provided with qoi:regexp.

In an optimal case all of these metadata fields are provided for a data source. In case these values are not provided, the SE provides a simple backup solution for typical fields like temperature or humidity or also a mac address to show the use case of qoi:regexp:

- mac {'regexp': '(?:[0-9a-fA-F]:?){12}', 'valuetype': 'string'}
- temperature {'min': -20, 'max': 50, 'valuetype': 'float'}
- humidity {'min': 0, 'max': 100, 'valuetype': 'float'}

These examples are used when no metadata is provided and the ObservableProperty of a data source can be identified as a temperature, humidity or mac sensor. It is also possible to provide additional types via the GUI of the SE.
For proofing that the component is running and for simple debugging tasks in a scenario, the SE features a simple web interface as a GUI, where a user can get information about active subscriptions, locally stored information from received notifications, a page to configure the fallback metadata, a log file viewer, and a simple configuration page. One example instantiation of the SE can be accessed for the Urban Data Mission MVP[18].
The calculated QoI is used by other components as an indicator to start extended fault detection (Monitoring component) or to select data sources with the highest possible QoI (Ranking component). As QoI provides a meaningful insight to the quality of a data source, it might be useful to not only use QoI internally but also by external users or the providers of a data source. Therefore, they can subscribe to the QoI data sources. This can on the one hand provide a user of data sources with information about its quality and provide a hint, if it is necessary to search for alternatives when a required quality metric, e.g. the frequency of updates, gets too low. On the other hand QoI also offers additional analysing features for the owners of a data source. By subscribing to QoI they can instantiate a basic monitoring system for their data sources. As an example it is possible to subscribe for plausibility to get informed for faults regarding measured values or subscribe to frequency, e.g. with a certain threshold, to get informed about variances regarding the update interval, which might also be caused by a used communication infrastructure.
Currently the SE is running in the Smart Parking and Urban Data Mission MVPs, where it can provide additional information used by the Monitoring or to show QoI values to developers or data source owners in the Urban Data Mission MVP. Best practice for other use cases would be to annotate the available data sources with the metadata fields that have been shown. If this is not possible from the beginning, the fallback solution can be used to get some initial Quality of Information annotations. Finally in short the steps an MVP owner should follow are:

[18] https://staging.urban-data-mission.semantic-enrichment.iotcrawler.eu/semanticenrichment/index

1. Annotate provided data sources with metadata
2. Provide fallback metadata in case metadata annotation is not possible for all data sources
3. Subscribe for QoI information to get informed for a decreasing QoI

Monitoring & Virtual Sensors

The Monitoring component in IoTCrawler framework is responsible for the detection and recovery from faulty values within a data stream as well as the creation of Virtual Sensors. Fault detection and recovery techniques implemented in the Monitoring component are designed to work with any given data stream. However, as the accuracy of detection varies between data streams (depending on data patterns and distribution), caution has to be taken in sensitive environments. Similarly, in industrial environments, such as Machine Monitoring (section 8), a wrong prediction can cause more delays in the system resulting in an increase of unnecessary expenditures. For such scenarios, it is highly recommended for the users to not to rely entirely on the Monitoring component and instead create custom fault detection and recovery solutions catering to their needs.

Currently, the Fault Detection and Recovery techniques are applied in SmartParking and Urban Data Missions MVPs. These MVPs have sensors deployed in harsher environments than other MVPs and are also very little impacted by a false prediction, which provides good criteria to utilise the Fault Detection and Recovery. To use the Fault Detection and Recovery, a stream needs to enable the component and provide unlabelled historical data, which the algorithms need to learn the patterns and distributions. For this, the historical data should be free of faults, outliers and gaps. Furthermore, if the data follows a seasonal pattern, the historical data should cover at least one complete cycle.

The Virtual Sensor component is responsible for creating virtual sensors in case of sensor failures. The virtual sensors are created by learning the relationships between the failed sensor and its nearby sensors. In case of absence of correlated sensor data, the Virtual Sensor component is unable to learn these relationships and hence cannot create a virtual sensor. Hence, the Virtual Sensor component can best be utilised in environments with high coverage of correlating sensors, e.g. sensors measuring the same phenomena. Currently, the Virtual Sensor component is

planned to be used by Urban Data Missions MVP in the next iteration, which consists a large number of sensors and hence provides a good number of sensors to use for the creation of a virtual sensor.

In order to get the most out of the component, a data provider should:

1. Provide historical data
2. Provide sensor location
3. Turn on Monitoring component
4. Subscribe to MDR to get result of Monitoring component

Below each of these steps is described in detail.

The historical data should be unlabelled and consists of only the normal behaviour. The number of samples should be enough to capture the dynamics of the environment the sensor is placed in.

The Virtual Sensors and some algorithms in the Fault Recovery with better predictive capability need the information about the sensors location so they can find other correlated sensors whose data will be used for predictions. In cases this information is not available, we implemented alternative algorithms in the Fault Recovery that can work without this information.

It should be noted that when this information is provided, the data provider should still be mindful of the constraints mentioned above.

Monitoring component should be turned on within the meta data (using "fd:Enable") for any stream that wants to use it. The value can be set to 1 to turn on and 0 to turn off. The functionality is provided so that in cases where the component is not needed or does not give good predictions, it could be turned off to save computing resources.

Users can subscribe to MDR to get the results of the Monitoring component, namely the properties "hasVerdict" and "hasEstimatedResult" in the StreamObservation entity, which will inform whether a reading is faulty or not and what is the estimated value from the Fault Recovery subcomponent, respectively.

10 Conclusion

The document concludes the results of task T7.2 devoted to development of technical prototypes demonstrating Minimum Viable Products, for a number of use-case scenarios. In total 6 MVP belonging to 5 different domains have been demonstrated: Smart City, Smart Home, Smart Energy, Industry 4.0 and Social IoT. The described MVPs have been developed and deployed using the IoTCrawler resources: a Kubernetes Cluster with Gitlab-based CI/CD procedures and the core IoTCrawler components being already deployed in the cluster. Developed MVPs demonstrate different level of integration with the core components depending on use-case needs: e.g. some MVPs such as Smart Parking or Smart Home rely on security components, while others do not have security as a requirement. MVP developers have been technically supported by developers of core IoTCrawler components, who also formulated a best practices of usage the components for MVP creation.

The results of the task T7.2 provides the testing foundation for the next step - task T7.3 user experience evaluation business model testing. The T7.2 contributes to main objectives of the whole WP7 – to provide understanding how IoT Search can deliver value into a different aspects of business and society.

11 References

[1] Impact of Merit Order activation of automatic Frequency Restoration Reserves and harmonised Full Activation Times, ENTSO-E 2016

[2] Smart Home Crawler: Towards a framework for semi-automatic IoT sensor integration M Strohbach, LA Saavedra, P Smirnov, S Legostaieva - 2019 Global IoT Summit (GIoTS), 2019

[3] P. Smirnov et.al., "D5.2 Enablers for Machine Initiated Semantic IoT Search," 15.01.2021 [Online] https://iotcrawler.eu/wp-content/uploads/2020/07/D5.2_Final.pdf

[4] ÖKOSTROMBERICHT 2019, E-Control, 2019; Innovative Energietechnologien in Österreich Marktentwicklung 2019,

[5] P. Biermayr et al., 2019; Marktentwicklung von PV-Heimspeichersystemen in Österreich, Fischer, Leonhartsberger, 2019;

[6] Statistische Zahlen der deutschen Solarstrombranche (Photovoltaik), Bundesverband Solarwirtschaft e.V., 2020;

[7] Deutscher Markt für Wärmepumpen legt 2019 leicht zu, pv magazine, 2019;

[8] Statistische Zahlen der deutschen Solarstrombranche (Speicher/Mobilität), Bundesverband Solarwirtschaft e.V., 2020;

[9] Das »Barometer der Energiewende«, Frauenhofer IEE, 2018

IOTCRAWLER

@IoTCrawler

IoTCrawler EUproject

/IoTCrawler

www.IoTCrawler.eu

mail@IoTCrawler.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779852