Intermediate Integrated
IoTcrawler Framework and Tools

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 779852
This document presents the current state of Task 6.2: Intermediate Integrated IoTCrawler Framework and Tools whose aim is to develop deployable results and tools support the application of the IoTCrawler framework in pilots and further utilization. Furthermore, it aims to guide the contributors of the IoTCrawler core framework towards sustainable, deployable and reusable results, which in a state-of-the-art quality managed and tested software stack.
Keywords:

Framework, Deployment, Tools, KPIs, Component’s responsibilities.

Disclaimer:

The present report reflects only the authors’ view. The European Commission is not responsible for any use that may be made of the information it contains.
# Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
<th>Author (Organization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>14/06/2019</td>
<td>Initial Proposal for Table of Content</td>
<td>Mirko Ross (DW)</td>
</tr>
<tr>
<td>V0.5</td>
<td>14/07/2019</td>
<td>Merging Framework Documentation</td>
<td>Mirko Ross (DW)</td>
</tr>
<tr>
<td>V1.0</td>
<td>17/07/2019</td>
<td>Merging Pilot Components</td>
<td>Mirko Ross (DW)</td>
</tr>
<tr>
<td>V1.1</td>
<td>18/07/2019</td>
<td>Adding descriptions from NEC</td>
<td>Mirko Ross (DW)</td>
</tr>
<tr>
<td>V1.2</td>
<td>22/07/2019</td>
<td>Review and minor changes</td>
<td>Pavel Smirnov (AGT)</td>
</tr>
<tr>
<td>V1.3</td>
<td>23/07/2019</td>
<td>Review and minor changes</td>
<td>Pedro Gonzales Gil (UMU)</td>
</tr>
</tbody>
</table>
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDR</td>
<td>MetaData Repository</td>
</tr>
<tr>
<td>IdM</td>
<td>Identity Manager</td>
</tr>
<tr>
<td>CM</td>
<td>Capability Manager</td>
</tr>
<tr>
<td>CT</td>
<td>Capability Token</td>
</tr>
<tr>
<td>PDP</td>
<td>Policy Decision Point</td>
</tr>
<tr>
<td>PEP</td>
<td>Policy Enforcement Point</td>
</tr>
</tbody>
</table>
Executive Summary

This document presents the current state of Task 6.2: “Intermediate Integrated IoTCrawler Framework and Tools” whose aims to develop deployable results and tools to support the development and deployment of the IoTCrawler framework in pilots. The work results are baseline actions for the future utilization. Furthermore, it aims to guide the contributors of the IoTCrawler core framework towards sustainable, deployable and reusable results, which are represented state-of-the-art quality managed and tested software stack.

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 Content

**Table of content** .......................................................................................................................... 7  
**Table of Figures** .......................................................................................................................... 8  

1. **Methodology** ............................................................................................................................... 9  

2. **Intermediate Framework Tools** ................................................................................................. 12  
   2.1 Organizing the SCRUM structure for development ................................................................. 12  
   2.2 Tools to organize the SCRUM based development .................................................................. 14  

3. **Deployment process** ................................................................................................................. 16  
   3.1 *IoTCrawler Framework Architecture* .................................................................................... 16  
   3.2 *IoTCrawler Core Components Deployment* .......................................................................... 17  
   3.3 *IoTCrawler Core Components* .............................................................................................. 17  
      3.3.1 Meta Data Repository ........................................................................................................... 18  
      3.3.2 PEP Proxy ............................................................................................................................ 20  
      3.3.3 PDP .................................................................................................................................. 22  
      3.3.4 PAP .................................................................................................................................. 23  
      3.3.5 IDM .................................................................................................................................. 23  
      3.3.6 Blockchain Handler ............................................................................................................ 25  
      3.3.7 Semantic Enrichment .......................................................................................................... 26  
      3.3.8 Monitoring .......................................................................................................................... 28  
      3.3.9 Orchestrator ......................................................................................................................... 30  
      3.3.10 Ranking ............................................................................................................................. 32  
      3.3.11 Indexing .............................................................................................................................. 33  
   3.4 *IoTCrawler Core Framework Deployment Process* .............................................................. 35  
   3.5 **Core Components for Pilots** ................................................................................................. 35  
      3.5.1 Pilot CompariSense & Pop-up ............................................................................................... 36  
      3.5.2 Pilot Smart Parking ............................................................................................................... 37  
      3.5.3 Pilot Smart Home .................................................................................................................. 37  
      3.5.4 Pilot Smart Campus: Room Booking ..................................................................................... 39  
      3.5.5 Pilot Smart Home: Elderly Care ............................................................................................ 40  
      3.5.6 Pilot Industry 4.0: Machine Monitoring .............................................................................. 41  
      3.5.7 Pilot Smart Energy: Flexibility trading for small assets ...................................................... 42  

4. **Conclusion** .................................................................................................................................. 43
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Agile SDLC methodologies selected for the IoTCrawler scope</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2</td>
<td>IoTCrawler Development SCRUM Cycle</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Overview SCRUM roles in IoTCrawler development</td>
<td>12</td>
</tr>
<tr>
<td>Figure 4</td>
<td>IoTCrawler Framework Architecture</td>
<td>15</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Principle of deploy management of a single component / enabler</td>
<td>16</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Deployment of use-case / pilot specific core components</td>
<td>34</td>
</tr>
</tbody>
</table>
1. **Methodology**

The IoT Crawler Framework will consist of exiting, adapted and new software components and libraries. Multiple stakeholders are contributing their work, which will be rolled out on different use-case and pilots. This will easily lead into complex conditions and requirements to gain deployable and reusable results. Furthermore, the final build of the framework needs to be maintainable to provide a sustainable utilization. Such software development, operation and maintenance challenges can be represented in the System Development Life Cycle (SDLC) methodology\(^1\). For the IoT Crawler framework the SDLC methodology phase itself is organized in SCRUM methodology, which is extending standard SDLC models towards an agile SDLC model.

This agile SDLC methodology is a combination of iterative and incremental process models. It reflects the needs of adaptability for IoT Crawler use-cases and pilots defined in deliverable 7.1. Also, it allows to break the complex framework into incremental builds. In IoT Crawler the development is a co-working process between international distributed teams. Building the framework in agile SDLC iterations, supports such teams to work simultaneous on their specific tasks and review the results with all project stakeholders at

---

the end of an iteration. The framework functionality (and complexity) will increase step by step on incremental iterations towards the final build, which will include all features required by the use case pilots.

For IoTCrawler, the phases defined in the SDLC can be linked to certain work packages and stakeholders. Before each iteration in the agile SDLC model the tasks on single stage are validated with the conditions of previous work (documented in deliverables), new conditions will be defined (results from new deliverables) or conditions will be changed (as a result from development and testing). The software contributions will be continuously documented in GitLab\(^2\) as collaborative code version management system.

Overview on agile SDLC cycle mapped to the IoTCrawler development:

- **Stage 1: Planning and Requirement Analysis**
  
  Tasks are based on results of WP 2 and are defining the Framework baseline requirements. This information is used to describe IoTCrawler basic approach and conditions to fulfil the technical, legal and economic needs. Results are documented in the deliverables:

  D2.1 Requirements and Design Templates for IoT Crawling

  D2.2 Security and Privacy aware IoTCrawler Framework

- **Stage 2: Defining Requirements**
  
  Tasks are defined by the results of Stage 1 deliverables. In this stage the overall requirement analysis is used to build the specific product requirements for the IoTCrawler Framework. Results of Stage 2 are building the Software Requirement Specification (SRS) and are documented in the deliverables:

  D3.1 Enables for IoT Security and Privacy Baselines

  D3.2 Secure IoT information Access and Privacy Preservation

  D7.1 Co-Creation of use-cases with stakeholder

- **Stage 3: Designing the Product Architecture**
  
  Based on the SRS of Stage 2 the product architecture will be defined and documented in specific Design Document Specifications (DDS). This DDS will be reviewed during iteration cycles by all product stakeholders (developers and pilots) towards risks, robustness, modularity and the project timeline. The DDS of IoTCrawler is represented in the deliverables:

  D4.1 IoT Data Attributes and Quality of Information

  D4.2 Large Scale IoT Crawling Indexing and Ranking

  D4.3 Adaptive and Secure Publication and Subscription in Highly Dynamic Environments

\(^2\) [https://about.GitLab.com/](https://about.GitLab.com/)
D 5.1 Enablers for Sensing and Understanding IoT Context

D5.2 Enablers for Machine Initiated Semantic IoT Search

D5.3 Monitoring and Fault Recovery in Highly Dynamic IoT Environments

- **Stage 4: Developing the Product**

In this stage of SDLC the IoTCrawler Framework is built by the project developers. Programming code is adopted, created and documented by GitLab. As the IoTCrawler Framework consist of code contributions from multiple partners, the project did not limit contributions on a specific coding language. To avoid friction losses the partners, agree on common methods for interoperability on modules (API’s). The development process is managed and documented in the deliverables:

D 6.1 Intermediate IoTCrawler Benchmarking and Testing Environment

D6-2 Intermediate Integrated IoTCrawler Framework and Tools

D 7.2 Develop and deploy hands-on prototypes

- **Stage 5: Testing**

Testing is a subset of all stages in modern SDLC models and therefore for IoTCrawler the testing is mostly involved on stages 3 (Design evaluation), stage 4 (quality management in development e.g. by unit tests) and stage 6 (feedback from pilots). The testing procedures are documented in the deliverables:

D6.3 IoTCrawler Benchmarking and Testing Environment

D7.3 Evaluation of user experience and business opportunities

- **Stage 6: Deployment and Maintenance**

Once IoTCrawler modules and components are tested, they will be deployed for usage in the framework and for specific pilots. This will be done via GitLab in defined iterations. The framework baseline components will be released finally on public GITHUB, when they have reached a sufficient maturity based on the DDS and testing results. The documentation will be done in the deliverables:

D 6.4 IoTCrawler Framework and Tools

D 7.2 Prototypes for End User Evaluation

D7.4 IoTCrawler Show Cases for Business Development
2. **Intermediate Framework Tools**

The SDLC method will be applied agile with SCRUM methodology. This is extending the method by an adaptive approach. As IoTCrawler is facing research challenges we need a flexible method to account for future tasks that technologies, conditions or features can be changed. Using an agile method also supports to show and check results after every iteration (Sprint). Furthermore, agile methods as SCRUM do support the work with distributed developer teams by giving a common legacy structure on team-based software development.

For IoTCrawler the agile project development teams are working on two main topics:

- **Topic 1**: Teams developing and adapting core framework components (Enablers) of the IoTCrawler
- **Topic 2**: Teams deploying and integrating the IoTCrawler framework for pilot use-cases

The development in specific topics is organized in SCRUM cycles:

![Figure 2: IoTCrawler Development SCRUM Cycle](http://scrummethodology.com/scrum-sprint/)

2.1 *Organizing the SCRUM structure for development*
Working in SCRUM methodology on the development tasks, requires structuring the development process and the roles of involved stakeholders. For IoTcrawler the structure has been defined as follows:

<table>
<thead>
<tr>
<th>SCRUM Role</th>
<th>Responsibility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCRUM Master</td>
<td>digital worx</td>
<td>Organizing the SCRUM development process</td>
</tr>
<tr>
<td>Product Owner</td>
<td>Single lead on each partner in Pilots / Enablers / Components</td>
<td>Defines the product specific properties</td>
</tr>
<tr>
<td>Developer</td>
<td>Team of developer on each Pilot, Enablers, Components</td>
<td>Working on Components, Enablers, Pilot deployments</td>
</tr>
<tr>
<td>Clients</td>
<td>Lead of Pilots</td>
<td>Customer involved in pilots / use-cases</td>
</tr>
<tr>
<td>Backlog</td>
<td>Product owner</td>
<td>Defining the specific functionalities and tasks</td>
</tr>
<tr>
<td>Retrospective</td>
<td>digital worx</td>
<td>Organizing the review of sprint cycles, providing tools for planning, documentation and review. Retrospective is done on weekly base with the development team by ZOOM Video conference.</td>
</tr>
</tbody>
</table>

*Figure 3: Overview SCRUM roles in IoTcrawler development*
2.2 **Tools to organize the SCRUM based development**

The work in sprint cycles is organized in distributed international teams. This requires tools to support the collaborative working process, tracking progress and the documentation of results. For the IoTCrawler development this SCRUM based process is organized with various tools:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Slack</strong></td>
<td>Collaborative Messenger Platform used for permanent communication with all members of development teams. URL: iotcrawlereu.slack.com</td>
</tr>
</tbody>
</table>

---

![Slack Chat](image-url)
**Zoom**

Video conferencing used for weekly retrospective calls and during sprints.

**GitLab**

Platform for source code and version management for collaborative development teams.

URL: GitLab.iotcrawler.net
3. DEPLOYMENT PROCESS

3.1 IoT Crawler Framework Architecture

The IoT Crawler Framework Architecture consists of components and enablers, linked by functional API’S as referenced in Deliverable 6.1:

![IoT Crawler Framework Architecture Diagram]

The core architecture can be clustered into 11 components (enablers), which are combined and used in specific configurations for the pilot use-cases. The IoT Crawler use-case are characterized by domain specific technical requirements and user experiences as described in deliverable 7.1. By that, the Pilots have different needs on using the core architecture components, domain specific configurations of components and interaction with use case specific sub-components. This is leading into a flexible architecture, where use-case can select required components of the core architecture and will deselect components not required. Therefore, a deployment of the Core Framework will be almost unique in its specific configuration of components and configuration per Pilot, while the core of a single component will be provided on a common source base.
3.2 **IoTCrawler Core Components Deployment**

In general, each component has an individual deployment process, which is depending on the specific development history and status. Components shall be deployed in virtual containers (e.g. Docker⁴), while existing or adopted containers will be deployed on instances (e.g. virtual machines or virtual containers). Each core component is developed or adopted, tested and published on own quality management cycles.

![Principle of deploy management of a single component / enabler](image)

*Figure 5: Principle of deploy management of a single component / enabler*

GitLab is the central repository to keep the sources and developer information on core components (enablers) The component’s product owner will decide, if a specific development version is ready to go for a release in a container version. The testing procedures and quality management will be done after the build of a release and the procedure is described in deliverable 6.1.

3.3 **IoTCrawler Core Components**

The IoTCrawler core architecture consists of 11 core components (enablers) which will be linked by functional API’s. Understanding the interactions between components and enablers is important when creating a deployable framework for specific pilot use-cases. For a better understanding, the dependencies of core components have been documented and shared on GitLab towards the development teams.

---

⁴ [https://www.docker.com/resources/what-container](https://www.docker.com/resources/what-container)
3.3.1 Meta Data Repository

<table>
<thead>
<tr>
<th>Component Name: Meta Data Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributor: UMU / NEC</td>
</tr>
<tr>
<td>GITLAB Link: <a href="https://GitLab.iotcrawler.net/metadata-repository/ngb-image">https://GitLab.iotcrawler.net/metadata-repository/ngb-image</a></td>
</tr>
</tbody>
</table>

**Functional Description of the Component**

This component is the core of the IoTCrawler platform since it is responsible for registering the information provided by IoT platforms, gateways or IoT devices in the NGSI-LD format and make it available to the other modules in order to provide semantic enrichment and also to control the way this information is distributed and/or provided to the users. It provides and NGSI-LD API with also support for a publish/subscribe pattern.

**Known dependencies**

PEP_Proxy, Semantic Enrichment

**Deployable Container (e.g. Docker)**

Deployment of this component is performed (by now) by means of automated scripts that launch the required software modules (JAR files and/or virtual machines). In the near future we expect to have a docker image available, which would allow us to integrate it with other deployed components by means of docker compose or other technologies.

**Description of Testing Procedure (e.g. Unit Tests)**

To be defined.

**Workflow Diagram**

Direct Interactions

- Create/Update/Delete NGSI-LD Operation
- OK/Error NGSI-LD Response
- Query NGSI-LD Operation
- Context/Error NGSI-LD Response

---
Provider Interactions

Context Registration
- Register Context NGSI-LD Operation
- OK/Error NGSI-LD Response

Context Querying
- Query NGSI-LD Operation
- Context Retrieval NGSI-LD Operation
- Context/Error NGSI-LD Response

Pub/Sub Mechanism

Context Subscription
- Context Subscribe NGSI-LD Operation
- OK/Error NGSI-LD Response

Context Updating
- Update/Create NGSI-LD Operation
- Context/Error NGSI-LD Response
- Context Notification NGSI-LD Message
3.3.2 PEP Proxy

**Component Name:** PEP Proxy

**Contributor:** OdinS

**GITLAB Link:** [https://GitLab.iotcrawler.net/xacml/pep_proxy](https://GitLab.iotcrawler.net/xacml/pep_proxy)

### Functional Description of the Component

This component is responsible for enforcing the policies generated by the Authorization Enabler - Capability Manager through the corresponding Capability Token (CT). This component receives the NGSI-LD messages aimed at the MDR which contains a CT. So, the PEP_Proxy validates this Capability Token, and after a positive verdict, it forwards these requests to the MDR, forwarding the responses too. Additionally, this component allows for CP-ABE\(^5\) encryption over specific attributes.

### Known dependencies

- PDP, MDR, IdM
- Deployable Container (e.g. Docker)

### Not defined yet

**Workflow Diagram**

PEP - enforcing AuthZ Enabler policies

---

**PEP - enforcing AuthZ Enabler policies plus cyphering data with CP-ABE policy**

---

\(^5\) https://ieeexplore.ieee.org/document/8119793
3.3.3 PDP

**Component Name:** PDP

**Contributor:** UMU

**GITLAB Link:** https://GitLab.iotcrawler.net/xacml/xacml_pdp

### Functional Description of the Component

This component of the XACML framework process XACML authorization request and validates against the XAML policies document generated by the PAP.

### Known dependencies

- Capability Manager, PAP
- **Deployable Container (e.g. Docker)**

Not defined yet

### Workflow Diagram

[Workflow Diagram Image]

**AuthZ Enabler**
- AuthZ Enabler – Cap. Manager
- AuthZ Enabler XACML PDP
- Blockchain Handler

**XACML – AuthZ request**

**Validate(XACML req)**

**Check XACML policies**

**Checked**

**Check**

**XACML Verdict**

---

**IQT CRAWLER**
3.3.4 PAP

Component Name: PAP
Contributor: UMU
GITLAB Link: https://GitLab.iotcrawler.net/xacml/xacml_pap

Functional Description of the Component

This component is responsible for managing XACML policies. It does not interact directly with other components since only writes the XACML policies in a document which is checked later by the PDP.

Known dependencies

PDP
Deployable Container (e.g. Docker)

Not defined yet

Workflow Diagram

```
AddNewPolicy(subject, resource, action)

... 

UpdateNewPolicy(subject, resource, action)

... 

RemoveNewPolicy(subject, resource, action)

... 
```

3.3.5 IDM
<table>
<thead>
<tr>
<th>Component Name:</th>
<th>IdM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributor:</td>
<td>UMU</td>
</tr>
<tr>
<td>GITLAB Link:</td>
<td></td>
</tr>
</tbody>
</table>

**Functional Description of the Component**

This component is responsible for managing the identities of the IoTCrawler platform. It is also responsible for authenticating these entities in the system. Additionally, it stores all the attributes corresponding to each entity and provides an API to access to them.

**Known dependencies**

- Pep Proxy, PDP
- Deployable Container (e.g. Docker)

Not defined yet

**Workflow Diagram**

```
Component    IdM

Auth Request(credential)  Validate(credential)  Response (Auth Token)
```
3.3.6 Blockchain Handler

Component Name: Blockchain Handler

Contributor: NEC
GITLAB Link: https://GitLab.iotcrawler.net/blockchain-handler/blockchain-handler

Functional Description of the Component

This component is responsible for handling blockchain network, executing smart contracts for inter-domain policy management. IoT components connect to the blockchain network via Router/Handler using NGSI-10 standard, while the Handler connects to the Blockchain backend using Fabric CLI\(^6\). The Blockchain will add policies, validate them and evaluate them upon request from authorization component.

Known dependencies

PEP, authorization components

Deployable Container (e.g. Docker)

Not defined yet

Workflow Diagram

\(^6\) https://jira.hyperledger.org/browse/FAB-1073
3.3.7 Semantic Enrichment

**Component Name:** Semantic Enrichment

**Contributor:** Siemens

**GITLAB Link:** [https://GitLab.iotcrawler.net/semantic-enrichment/semanticenrichment](https://GitLab.iotcrawler.net/semantic-enrichment/semanticenrichment)

**Functional Description of the Component**

The Semantic Enrichment is responsible for rating data sources and their delivered data for quality of information. For doing this the component subscribes to the MDR for newly registered data sources. While receiving metadata of new sources through notifications and using the Monitoring component to access data of the sources, the enrichment component calculates their QoI. The calculated QoI is saved back to the MDR as an additional input for the Ranking component.

**Known dependencies**

MDR, Monitoring

**Deployable Container (e.g. Docker)**

**Workflow Diagram**

[Workflow Diagram Image]
3.3.8 Monitoring

Component Name: Monitoring

Contributor: UASO
GITLAB Link: https://GitLab.iotcrawler.net/monitoring

Functional Description of the Component

All the data provided by every data provider will pass through the Monitoring Component. Monitoring is also responsible for the following operations:
- Monitoring will pass this data to the Semantic Enrichment component for QoI calculations.
- Monitoring will forward this data to the subscribed application
- It will detect faults in data streams and use recovery techniques, included in its sub component "Fault Recovery".
- It will also accept the requests for virtual sensor creation through its subcomponent "Virtual Sensors"

Known dependencies

Ranking, Indexing, Authorization Enabler
Deployable Container (e.g. Docker)

Docker Workflow Diagram
3.3.9 Orchestrator

**Component Name:** Orchestrator

**Contributor:** AGT

**GITLAB Link:** https://GitLab.iotcrawler.net/orchestrator/orchestrator

**Functional Description of the Component**

The orchestrator is responsible for handling the following tasks:

1) Serving requests from applications: Orchestrator is responsible for handling search and subscription queries for entities stored in metadata repository (e.g., streams, sensors, observable properties, etc.). Internally, orchestrator communicates with a number of other IoT Crawler components, but this communication is totally hidden from applications.

2) Initializing of virtual sensors when it is possible (in cases when none of existing sensors are matching the query requirements).

3) Handling fault-tolerance

**Known dependencies**

Ranking, MDR, Authorisation enable, PEP Proxy

**Deployable Container (e.g. Docker)**

Docker image: iotcrawler/orchestrator

**Workflow Diagram**

[Workflow Diagram Image]
Component Name: Ranking

Contributor: Siemens
GITLAB Link: https://GitLab.iotcrawler.net/ranking/ranking

Functional Description of the Component

This component is responsible for:
- Forward the query requests from the Orchestrator to the Indexing
- Receive the search results back from the indexing
- Rank the results and send them to the Orchestrator

Known dependencies

Orchestrator, Auth Enabler, Indexing

Deployable Container (e.g. Docker)

Workflow Diagram

```
Interactions Ranking

External network  IoT network  PEP Proxy  Indexing

Application  Orchestrator  Auth Enabler  Ranking

getStreams(query, rankingCriteria)  getStreams(query)
rankedList<ioTStream> by a criteria  List<ioTStream>
```
3.3.11 Indexing

Component Name: Indexing

Contributor: Surrey
GITLAB Link: https://GitLab.iotcrawler.net/indexing/indexing

Functional Description of the Component

This component is responsible for indexing the IoT resources based on their attributes and meta-data descriptions and also indexing and discovery of patterns using machine learning and clustering methods. Overall the indexing is provided in two ways:

- Indexing the resource description attributes; e.g. using geohashing for location data and long/lat attributes;

- Pattern creation and indexing using clustering methods for the content of the data streams.

Known dependencies

Ranking, MDR, Authorisation enable

Deployable Container (e.g. Docker)

Docker

Workflow Diagram
3.4 IoTCrawler Core Framework Deployment Process

The core framework will be deployed by combining the core components needed for the specific use-case and deploying the specific configuration and settings for the functional interaction of core components and use-case specific components. By that, specific use-cases can use a different set of core components in the pilots.

Figure 6: Deployment of use-case / pilot specific core components

3.5 Core Components for Pilots

The tables below indicating the core architecture components intended for the application in the IoTCrawler pilots. Components will be deployed in containers with pilot specific on configurations and settings.
### 3.5.1 Pilot CompariSense & Pop-up

<table>
<thead>
<tr>
<th><strong>Pilot Name:</strong></th>
<th>CompariSense &amp; Pop-up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Partner</strong></td>
<td>Aarhus Municipality</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core Architecture Components in use, mark with X</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Meta Data Repository (MDR)</td>
</tr>
<tr>
<td>X Pep Proxy</td>
</tr>
<tr>
<td>X PDP</td>
</tr>
<tr>
<td>X PAP</td>
</tr>
<tr>
<td>X IDM</td>
</tr>
<tr>
<td>Blockchain Handler</td>
</tr>
<tr>
<td>X Semantic Enrichment</td>
</tr>
<tr>
<td>X Monitoring</td>
</tr>
<tr>
<td>X Orchestrator</td>
</tr>
<tr>
<td>X Ranking</td>
</tr>
<tr>
<td>X Indexing</td>
</tr>
</tbody>
</table>

**Domain specific components interacting with Core Architecture Components**

The CompariSense concept is an IoT Product Validation platform that provides reference data sets (real and virtual) by crawling all available IoT devices across a city. Using the data from these IoT devices allows IoT startups and their customers to validate the data being generated from their own solutions. Pop-up Experimentation Space: This concept is an IoT experimentation platform where flexible urban experimentation spaces in a city can be created by geo-fencing an area. The experimentation space then automatically integrates any IoT devices being implemented in the space from approved citizens or companies. Both concepts have similarities with the Smart Home pilot from AGT, but will be operating in an urban environment, therefore we expect to use similar components, but may also include ranking.
### 3.5.2 Pilot Smart Parking

<table>
<thead>
<tr>
<th>Pilot Name:</th>
<th>Smart Parking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>Murcia, OdinS and University of Surrey</td>
</tr>
</tbody>
</table>

**Core Architecture Components in use, mark with X**

- **X** Meta Data Repository (MDR)
- Pep Proxy
- **X** PDP
- **X** PAP
- **X** IDM
  - Blockchain Handler
- **X** Semantic Enrichment
- **X** Monitoring
- **X** Orchestrator
  - Ranking
- **X** Indexing

**Domain specific components interacting with Core Architecture Components**

This pilot will allow search and discovery of the sensory resources within the city and will also allow to find information and insights and identify the events from the sensory observations and measurements. The sensory resources (traffic, pollution, noise, parking, etc.) will be registered at the IoTBroker. The MDR implementation with the IoTBroker crawling is needed to allow search and discovery of the resources based on their meta-data and semantic descriptions. Additionally, Semantic Enrichment will be used to extract higher-level descriptions of the underlying raw data. The pilot will also use the indexing and orchestration components to crawl the raw data streams and search and discover patterns for continuous data.

### 3.5.3 Pilot Smart Home
**Pilot Name:** Smart Home  
**Partner** AGT

<table>
<thead>
<tr>
<th>Core Architecture Components in use, mark with X</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Meta Data Repository (MDR)</td>
</tr>
<tr>
<td>Pep Proxy</td>
</tr>
<tr>
<td>X PDP</td>
</tr>
<tr>
<td>X PAP</td>
</tr>
<tr>
<td>X IDM</td>
</tr>
<tr>
<td>Blockchain Handler</td>
</tr>
<tr>
<td>X Semantic Enrichment</td>
</tr>
<tr>
<td>X Monitoring</td>
</tr>
<tr>
<td>X Orchestrator</td>
</tr>
<tr>
<td>Ranking</td>
</tr>
<tr>
<td>X Indexing</td>
</tr>
</tbody>
</table>

**Domain specific components interacting with Core Architecture Components**

This pilot will crawl Smart Home networks and semi-automatically integrate discovered devices into the IoTcrawler infrastructure. This will be facilitated by the SmartConnect UI Tool that recommends how raw device metadata and data can be semantically enriched. The user makes the final decision on the semantic enrichments and automates the process of exposing semantically enriched IoTStreams in the MDR. IoTStreams can be accessed using the orchestrator and index. The Monitoring component can be used for notifying about changes between IoTStreams (e.g. faults, appearance on new ones).
3.5.4 Pilot Smart Campus: Room Booking

<table>
<thead>
<tr>
<th>Pilot Name:</th>
<th>Smart Campus: Room Booking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>Aarhus University</td>
</tr>
</tbody>
</table>

Core Architecture Components in use, mark with X

- X Meta Data Repository (MDR)
- Pep Proxy
- X PDP
- X PAP
- X IDM
  - Blockchain Handler
- X Semantic Enrichment
- X Monitoring
  - Orchestrator
- X Ranking
- X Indexing

Domain specific components interacting with Core Architecture Components

This room booking pilot will allow search and discovery of the sensory resources within the university buildings and will also allow to find information and insights from the sensory observations and measurements. The sensory resources (PIR, heat, humidity) will be registered at the IoTBroker. The MDR implementation with the IoTBroker crawling is needed to allow search and discovery of the resources based on their meta-data and semantic descriptions. Additionally, Semantic Enrichment will be used to extract higher-level descriptions of the underlying raw data such as user ratings and user location. The pilot will also use the indexing, ranking and orchestration components to crawl the raw data streams and search and discover patterns and trends of rooms booking and users preferences/location to make more accurate recommendations for room booking.
### 3.5.5 Pilot Smart Home: Elderly Care

<table>
<thead>
<tr>
<th><strong>Pilot Name:</strong></th>
<th>Smart Home: Elderly Care</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Partner</strong></td>
<td>University of Surrey</td>
</tr>
</tbody>
</table>

#### Core Architecture Components in use, mark with X

- X Meta Data Repository (MDR)
- Pep Proxy
- X PDP
- X PAP
- X IDM
- Blockchain Handler
- X Semantic Enrichment
- X Monitoring
  - Orchestrator
  - Ranking
- X Indexing

#### Domain specific components interacting with Core Architecture Components

The Smart Home: Elderly Care pilot will search the home for sensory resources (movement, light, energy monitoring) that are registered at the IoTBroker. The MDR implementation with the IoTBroker crawling is needed. Additionally, Semantic Enrichment will be used to extract higher-level descriptions of the underlying raw data. The monitoring component will be also used to monitor the existing and underlying resources. The pilot will also use the indexing and orchestration components to crawl the raw data streams and search and discover patterns for continuous data.
Pilot Industry 4.0: Machine Monitoring

<table>
<thead>
<tr>
<th>Pilot Name:</th>
<th>Industry 4.0: Machine Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>Digital Worx</td>
</tr>
</tbody>
</table>

Core Architecture Components in use, mark with X

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Meta Data Repository (MDR)</td>
</tr>
<tr>
<td></td>
<td>Pep Proxy</td>
</tr>
<tr>
<td>X</td>
<td>PDP</td>
</tr>
<tr>
<td>X</td>
<td>PAP</td>
</tr>
<tr>
<td>X</td>
<td>IDM</td>
</tr>
<tr>
<td></td>
<td>Blockchain Handler</td>
</tr>
<tr>
<td>X</td>
<td>Semantic Enrichment</td>
</tr>
<tr>
<td>X</td>
<td>Monitoring</td>
</tr>
<tr>
<td></td>
<td>Orchestrator</td>
</tr>
<tr>
<td></td>
<td>Ranking</td>
</tr>
<tr>
<td>X</td>
<td>Indexing</td>
</tr>
</tbody>
</table>

Domain specific components interacting with Core Architecture Components

The Industry 4.0 is indexing machining sensor data of registered sources at IoTBroker. To create valuable search results to data is enhanced by the Semantic Enrichment with context data of the production process. The search pilot aims to get better search results for incidents on shop floor to speed up resolving of problems on production chains and to support predictive maintenance based on data search results of machining condition monitoring.
3.5.7 Pilot Smart Energy: Flexibility trading for small assets

<table>
<thead>
<tr>
<th>Pilot Name:</th>
<th>Smart Energy: Flexibility trading for small assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>Siemens</td>
</tr>
</tbody>
</table>

Core Architecture Components in use, mark with X

- X Meta Data Repository (MDR)
  - Pep Proxy
- X PDP
- X PAP
- X IDM
  - Blockchain Handler
- X Semantic Enrichment
  - Monitoring
- X Orchestrator
- X Ranking
- X Indexing

Domain specific components interacting with Core Architecture Components

The Smart Energy pilot scans the network for assets (EVs, buildings, homes) that are registered at IoTBrokers. For that, the MDR implementation with the IoTBroker crawling is needed. Additionally, Semantic Enrichment will be used to derive high level descriptions of the underlying data. From the application point of view, the pilot searches for assets that can be used for flexible energy trading by means of control power. Search results might be ranked based on quality attributes or assets’ origin and location. For that, the Orchestrator is first contacted, which then forwards the search request to the Ranking and Indexing components.
4. CONCLUSION

Task 6.1 aims to provide deployment packages of the IoTCrawler Framework, which will support a manageable way to deliver, roll out and configure the framework in pilots of WP7. To achieve this, the development process has to be structured by methods, development tools have to be established and the development cycles needed to be initiated.

For the intermediate report of WP6 the project partners are working together on a collaborative development process to build the components of the framework. The progress of work is documented and supervised with installed tools as GitLab, Slack and Zoom. This established agile development cycles and collaboration is the foundation for the structured roll out of virtual component containers.

The blueprints of components and of the pilot framework are prepared and will be released in the upcoming development cycles to provide finalized IoTCrawler framework deployments for the pilots.