# Architecture for Knowledge Exploration of Industrial Data for Integration into Digital Services

Jan Nicolas Weskamp*, Arnab Ghosh Chowdhury †, Florian Pethig*, Lukasz Wisniewski†
*Fraunhofer IOSB-INA
{jan.nicolas.weskmap, florian.pethig}@iosb-ina.fraunhofer.de
†Technische Hochschule OWL - Institut für industrielle Informationstechnik (inIT)
{arnab.ghosh, lukasz.wisniewski}@th-owl.de

***Abstract*—Gaining added value from industrial data is a challenging process. It is often not known what data is available, in what form and what it means. In addition, the infrastructure to integrate the data into appropriate services is often missing. Another challenge is the heterogeneity of solutions, technologies and infrastructure on the shop floor. This paper presents a solution that integrates industrial data based on Industrie 4.0 technologies from the shop floor and provides it to a knowledge platform. The proposed solution is working as autonomous and automated as possible. The existing data and metadata are used to automatically configure and instantiate the components of the platform. Users of the platform can interact with the platform via a web application and use data or services relevant to them. There is also an interface that allows services to search directly for required data. To achieve this, components were developed to collect data from the production. Furthermore there is a knowledge platform where the data is prepared, stored and searched. This platform interacts with a middleware and the services, so that the data arrives automatically at its desired destination. The solution was evaluated through an implementation, showing that the usage of meta information is enabling the fast and easy integration of data in the digital services for Industrie4.0.**

***Index Terms*—Industry 4.0, Asset Administration Shell, Meta Data, OPC UA, Knowledge Exploration, Big Data Platform**

## I. Introduction

Different IoT (Internet of Things) enabled devices, together with other already existing ones are generating and collecting a huge amount of data [1]. According to the study by McKinsey "manufacturing stores more data than any other sector - close to 2 Exabyte of new data in 2010" [2]. This data is unstructured, miscellaneous and complex to handle. However, valuable information can be derived from this data.
The usage, integration of and access of this data can lead to an extensive benefit in areas like production, decision making and prediction. Nevertheless, generating benefit out of such data is challenging. It is due to the heterogeneity of the environment, data access and a lack of interoperability. As a result, a solution is desirable that can index and process large amount of data, respectively metadata, in order to enable user friendly knowledge exploration. Hence, in this paper a concept and implementation for an industrial knowledge platform, capable of locating required industrial data and storing its meaning in a knowledge database is presented. The metadata in the database can be queried by users as well as digital services to find and use appropriate data available from production

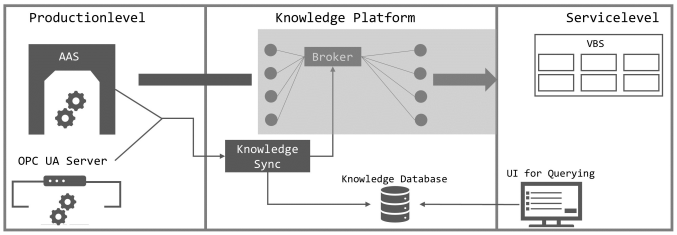


Fig. 1: Overview of Knowledge Exploration

environments. To achieve this, "classic" big data approaches were combined with standardized concepts and developments, that have been established since the launch of the Industrie 4.0 initiative. This can ensures that the solution is able to collect data from industrial environments using I4.0 technologies, index it, make it searchable and provide it to digital services. A digital services in context of this paper is any virtual entity that can serve, work or gaining a benefit out of the available data.
One main contender for standardized communication in I4.0 is the Open Platform Communications Unified Architecture (OPC UA). For this reason, OPC UA is used to gather industrial data. On the classic IT side, technologies such as Docker, Apache Kafka and Database solutions for the operation of different components is used. In summary, the knowledge platform, pursues the following three objectives:

- Provide an interface to users and digital services, which allows to search for data in the platform and fetch it for further processing.
- Provide components required to integrate, transport and pre-process the data automatically.
- Provide a data discovery mechanism, which continuously updates the knowledge database.

Figure 1 sketches the concept of the whole solution and the knowledge exploration.
The remainder of the paper is structured as follows. Section II introduces the state of the art. Section III presents specific challenges from industry and requirements, from which the concept and the functions of the knowledge exploration can be derived. Section IV describes the architecture and introduces technologies used. Section V shows how the data gathering and access has been implemented. This section presents the Proof-of-Concepts implementation of the proposed architec-

ture respectively platform. The final section VI discusses and summarises the results and gives an outlook on future work.

## II. State of the Art

When sensors, actuators and computers are connected via the Internet, they form the so-called IoT. Between these different things information can be exchanged and delivered to digital services [3]. The IoT is available in several application areas like smart manufacturing, smart home, smart city, transportation systems and smart grid. In context of industrial applications manufacturer are implementing the IoT technology for intelligent communications in their applications. So the IIoT (Industrial IoT) is describing the IoT for industrial applications [4]. The term Industrie 4.0 (I4.0) describes how the IoT or more specifically IIoT will improves the production, engineering and management processes of the industry. Real assets are represented by virtual components in the digital world, resulting in permanently connected systems, also known as Cyber-Physical Systems (CPS) [5]. If these assets, in context if I4.0, are production systems, cyber physical production systems (CPPS) are created [6].

The core of distributed industrial components relies on reliable exchange of information. In the beginning the industrial communication was realized based on fieldbus systems. Nowadays the majority of systems are communicating through Ethernet based networks. Since the last years they are gradually extended by wireless networks that become more and more robust and real-time capable. The adoption of the IoT and CPS facilitates the integration of inceased ammount of information flows in the automation industry [7].

A defacto standard communication protocol incorporated within Industrie 4.0 is OPC UA. It enables inter-operable and standardized communication between machines (M2M) and higher systems [8] as well as information modelling. Alternative approaches for I4.0 communication such as MQ Telemetry Transport (MQTT) [9] or Constrained Application Protocol (CoAP) [10] offer low memory requirements and data overhead.

In classical computer science the automation of processes using available semantics is already established. It is possible to generate and execute code from UML models [11]. Furthermore, DevOps automates the processes of development and operation, to enable faster and more efficient implementation of software development projects [12]. Both examples use knowledge or semantics that have been enriched for the respective applications. Projected on industrial applications there are already first suggestions which point in this direction. In [13], a conceptual model and necessary elements for a real world implementation, towards knowledge based and intelligent systems for Industrie 4.0 was proposed. In this approach the CPS and CPPS are generating data that is transmitted in a standardized form to semantic enhanced CPS. Apps and other application are using this information to provide a value out of this data. At the same time CPS on the shop floor communicate with CPS in the cloud and convert signals into knowledge (Figure 2). A more concrete architecture for interactive data

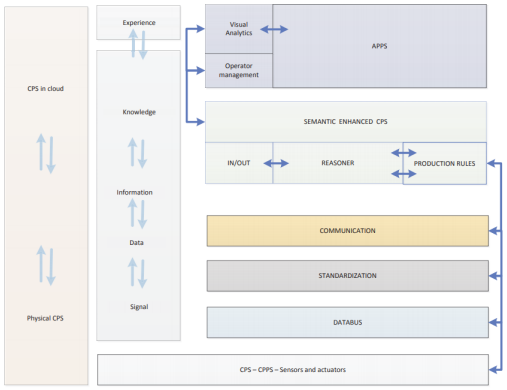


Fig. 2: Architecture proposed by [13]

exploration in a Smart Factory was suggested by [14]. In their architecture, a physical machine is producing data, that is transmitted by a Platform Service Bus. This bus has interfaces to databases, storing meta data, process data and summarised data. On the other side are services like a state detection or data exploration using that data. A GUI is provided to interact with this services.

## III. Challenges and Requirements

In industrial production systems, data is collected/generated by sensors, actuators and PLC programs. This data can offer companies comprehensive added value, if it can be integrated into corresponding services such as AI algorithms, visualizations or systems like an Enterprise-Resource-Planning. In this context, companies face several challenges that are not trivial to solve, such as:

- Companies often do not know what data is available in their production
- The meaning of individual values is not known.
- The semantic connection between the data is not clearly recognizable due to missing semantic annotations.
- The vast amount of data is difficult to handle.
- It is challenging to find the right information in the raw data.
- The use of different technologies makes comprehensive data integration difficult.

The challenges mentioned above can be classified according to different causes. Missing semantic relationships or links between data, that logically supplement each other arise out of insufficient semantic annotation. The lack of access and overview of the existing data, on the other hand, indicates insufficient data integration and preparation on a suitable platform. The first cause can only be fixed by an extensive metadata description, which is presupposed in the context of this paper. Therefore, the solution proposed in this paper has been designed to eliminate the second cause. To achieve this, the following requirements have been formulated:

- **A) The creation and configuration of platform services have to be done dynamically:**
  The knowledge platform does not know which resources and software components have to be available and to what

extent. This information results from the interaction of the users or the input parameters of the digital service requesting certain data. As soon as users or services find suitable information in the knowledge database, various services of the platform must be extended or instantiated in the background. These services ensures that the selected data will be delivered to its target and hence have to be dynamically creatable, depending on which input parameters are used, that are either provided by the user, the service or its metadata.

- **B) The current operation of the platform needs to be registered:**
  As soon as data has to be available for applications, a series of checks must be triggered in the background to prevent the multiple creation of a service or usage of resources for already available data. Data that has already been indexed, for example, should not be collected twice. Data that was already chosen and is available for services does not require another subscription or transmission. On the other hand, data pipelines must be removed if they are no longer used. This ensures that the solution is always up to date and the vitality of the system is maintained.
- **C) The data from production shall be integrated with standardized technologies and existing metadata:**
  As already mentioned in the introduction, OPC UA is the interface to the production and is the main technology, with which the data integration will be realized. In addition the concept of the Asset Administration Shell (AAS) provides information about assets in production environments [15]. The AAS may be the logical representation of a simple component, a machine or a plant at any level of the equipment hierarchy and will contain all information about the entire product life cycle of an assets. Together with the asset it represents the I4.0 component [16]. The structure of an AAS can be mapped with an OPC UA information model. To ensure, that this mapping is standardized, the Plattfrom Industrie 4.0 is running a working group, which works out the necessary Companion Specification for OPC UA and the AAS [17]. Both technologies, OPC UA and the AAS, provides already an extensive amount of information about the production and the data.
- **D) Users needs an interface to interact with the knowledge database:**
  The user needs an interface to interact with the information stored in the knowledge database. In order to guarantee an intuitive operation, a search realized via a web application can be used to search for suitable data within the database. One advantage of the implementation is, that the shop floor or CPPS can be explored from everywhere. The results are displayed in prepared form and the user can select relevant data. The data will be represented by its meta information such as the associated OPC UA server or AAS, the data type or a description. This information will enable the user to choose the appropriate data.
- **E) The data must be stored in such a way that it can be searched quickly and easily:**
  Via the web interface, one can search for the data needed for the current application. A service can communicate directly with the system, but for both it has to be possible to search for specific information. The metadata stored in the Knowledge Database is searched accordingly. This search must be as fast, flexible and accurate as possible. On the one hand, the search needs to provide the possibility to search freely, so that both the metadata values and the metadata names itself can be searched for the keyword(s). On the other hand, it is necessary to be able to define a search as specific as possible. Whether to search in concrete metadata fields, a combination or to search for substrings in the available information.

Finally, the data has to be transmitted to the services. For this a middleware is needed and will not considered in this paper. It is assumed, that the middleware is able to orchestrate the dataflow, providing subscribed data from the shop floor level and can interact with the knowledge platform.

## IV. Platform Architecture

With respect to the introduced requirements the architecture displayed in figure 3 shows the layout of the solution. The architecture is developed to enable knowledge exploration of industrial data, that is realized through the components displayed in the upper boxes.

As already mentioned we assume that the data at the shopfloor is provided via OPC UA or/and the Asset Administrations Shell. Nevertheless the concrete infrastructure of the network (number of servers, endpoints, AAS locations) are not known. So the question is: Where are my OPC UA servers/AAS and how many exists? Therefore the *Shopfloor component* is necessary, to ensure that data can be integrated independently into the platform. OPC UA offers the concept of Discovery Servers to meet this requirement. Discovery servers always running at the same address inside the network and all available OPC UA servers can register to them [18]. Inside the discovery server, the OPC UA servers are registered with their endpoints. This is the only necessary information to get connected to them. For collecting the information of the OPC UA servers, one has then just to ask the discovery server for the registered endpoints. It is then possible to read out the information models via an OPC UA client.

Until now there is no discovery procedure available for the AAS but there are ongoing standardization activities in the Plattform Industrie4.0. In the ongoing standardization three AAS types exists [19], [20]:

1) Type 1 AAS provides a standardized structuring of information belonging to an asset in a file format.
2) Type 2 AAS extends the Type 1 AAS with a standardized interface to external clients. To interact with this AAS type, an application need to support the interface.
3) A type 3 AAS has, in addition to type 2, functionalities to make an I40 Asset an interacting component which can support system integration and system interaction.

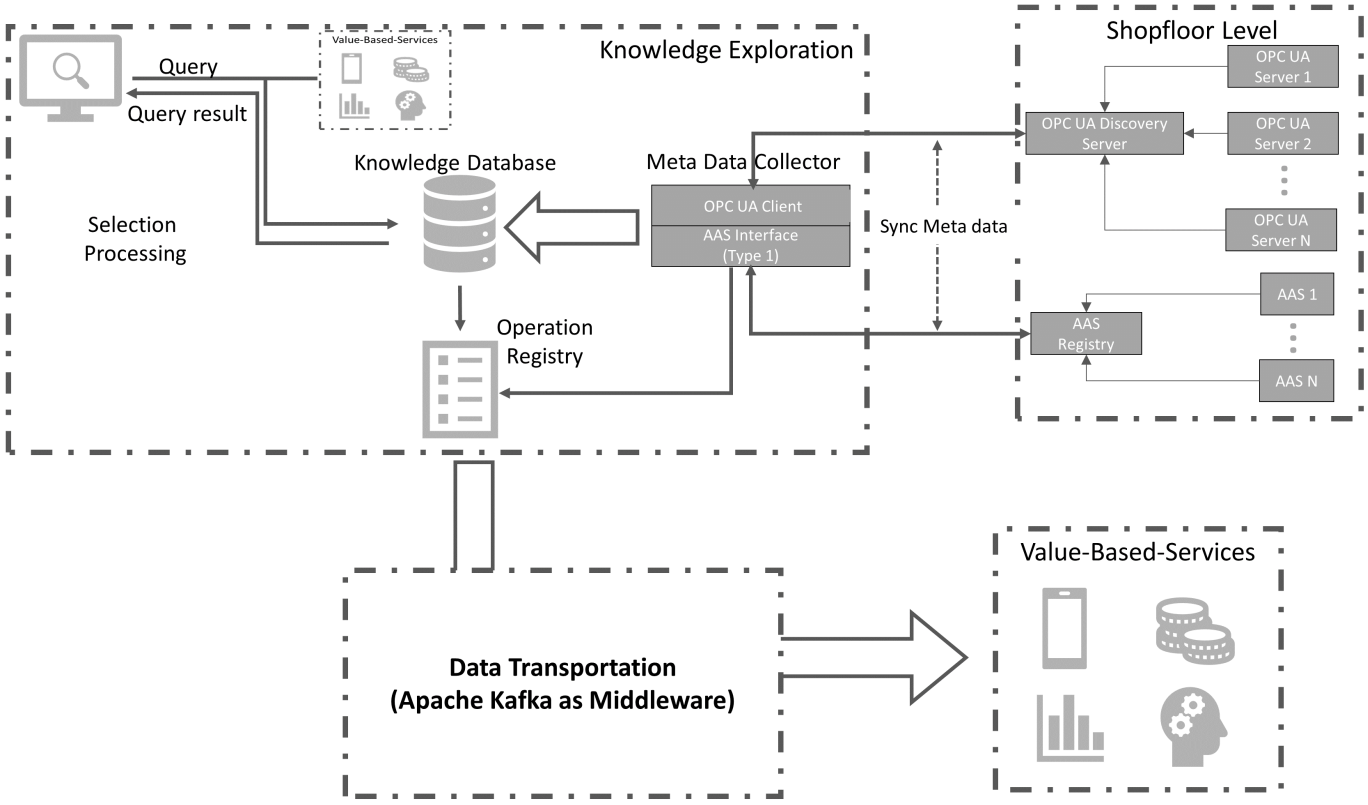


Fig. 3: Platform Architecture

In the architecture we are able to integrate the AAS as a file (Type 1) and AAS modeled via the OPC UA information model. The AAS can either be stored directly in the *AAS Registry* or the path to the AAS is stored accordingly. In the second case, it must be ensured that the Meta Data Collector can reach and read the AAS via the network. The supported AAS file type is JSON as it can be exported from the AAS Package Explorer [21]. The AAS Package Explorer is an open source software, that can be used to model AAS. In addition, it provides functionalities to import, export and serve the modeled AAS. From the registry the procedure is the same as for the OPC UA part. All registered AAS are parsed and will be injected in the knowledge database. This and the OPC UA Discovery is part of the solution linked to requirement **C**. *The knowledge exploration* is responsible for the collection of knowledge from the shopfloor and can be matched to the requirements **A**, **B**, **D** and **E**. First of all, the component contains the central *Knowledge Database* in which all information from the production level is stored. In order to fill this database, the information has to be extracted from the OPC UA servers and AAS. Too access the OPC UA data a corresponding client is implemented. It can be expected that the AAS can also be mapped via an OPC UA information model. In this case the AAS registry need not be concerned. But for the AAS type 1 which are not implemented with OPC UA, the registry has to be queried. The needed service is called *Meta Data Collector* and will interact with the Discovery Server and the AAS Registry. This component will continuously check, if there are changes in the registry and will provide the up-to-date information to the database. In the knowledge database only meta information, which is describing the semantic of the available data is stored. In the end the real process data, that is described through that data is subscribed.

The database can be queried through a web application, looking for suitable data. The web application is offering a simple search form. The concrete queries depend on the database technology used and are described in the upcoming chapter V. The services that want to request data directly must do this also via a query that depends on the database. As soon as the query was processed by the database and results has been found, it will be displayed in the web application. When the query was executed by a services, it has to be capable to handle the answer. Depending on the selected data, the solution must check whether data subscriptions already exist for some of the requested data. This point is very important to ensure that no data is send twice. To ensure that, the *Operation Registry* will hold this information. It has to know about the current status and instances of the relevant components. In general the following information should be available and always up-to-date inside the registry:

- The location (Address) of available OPC UA Discovery Servers
- The already subscribed OPC UA Data
- Parameter for the operation of the platform (i.e. Refresh interval, path to AAS storage)

So the registry is maintained and filled by the Web Interface

and the *Meta Data Collector*. The *Knowledge Exploration* component is sending a JSON, with the necessary information for creating the data exchange between the different platform components, the shopfloor and the services to the middleware.

## V. Implementing knowledge exploration

All software components presented here were implemented in Docker Container.

### A. Shoopfloor level

For this component the OPC UA Discovery Server and the AAS registry were implemented. For the OPC UA Discovery Server implementation the open source Python framework was used [22]. The OPC UA Discovery Server is providing Sign&Encrypt security and is waiting for OPC UA server for registration. In the server just one OPC UA method *GetRegisteredEndpoints* was implemented, returning a list of the currently registered endpoints. This method will be used by the *Meta Data Collector*.
To monitor the behavior of the server an additional OPC UA client is integrated in the discovery service. This client is printing information about the number of currently registered servers, a list of the endpoints and is indicating if the discovery server is running as expected. When the individual OPC UA servers registering to the Discovery Server by calling *RegisterToDiscovery* service, the client gets a list of endpoint URLs (Uniform Resource Locator) of the registered OPC UA servers by calling *FindServer* service [22], [23] from the Python OPC UA framework. A Python job scheduler is running at regular time interval to monitor the availability of the registered OPC UA servers. When a registered OPC UA server is stopped or become unavailable, the server registration expires and the endpoint URL of the server was removed from the registered server list. When the same OPC UA server starts or is available again, it will be registered to the Discovery Server and the monitoring client will display the information. The AAS registry is also a Python implementation. Consisting of a data structure holding the paths to AAS-Files, a static information about the directory where AAS are stored directly and a list of the names of the AAS in this directory. There are two interfaces in the application to the shopfloor side. The first one is providing an endpoint to add a new AAS-Path to the data structure and is listening to incoming calls. The second one is continuously querying the directory where AAS-Files are placed and is updating the the internal list of the files. To the knowledge exploration component an additional interface was implemented, which can be queried by the *Meta Data Collector*. This interfaces is providing the stored information about AAS paths and file names to that component. Figure 4 is sketching the shopfloor implementation.

### B. Knowledge Exploration

The main part was the implementation of the knowledge exploration component. In the upcoming subsection the main implementations, *MetaDataCollector*, *Operation Registry* and *Knowledge Browser* are explained. The database itself is not

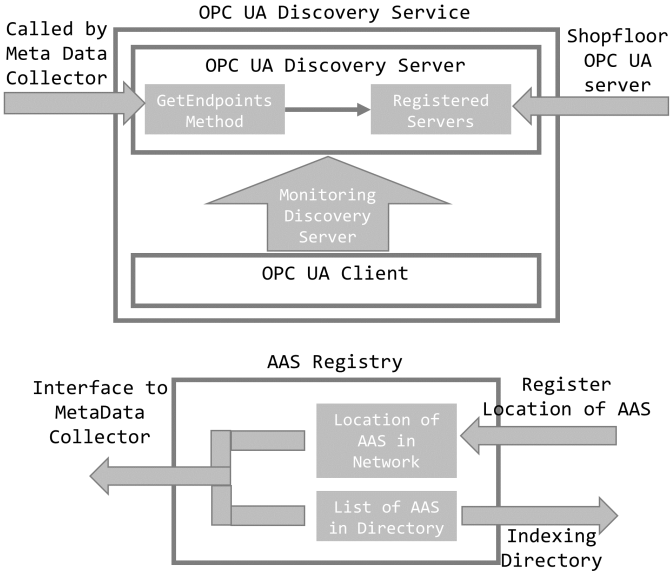


Fig. 4: Implementation Shopfloor components

self developed. Since there are numerous database solutions for storing the meta information of the data, nothing new was implemented, but rather an already well-established and suitable database for the solution was integrated. Due to the fact that the focus of the solution is on indexing and searching of data, Elasticsearch [24] was chosen as database technology. Thus, the API provided by Elastic is used to query for data and integrated into the *Browser Component*.

*1)* ***Meta Data Collector****:* The task of the *Meta Data Collector* is to gather the meta information out of the OPC UA servers and AAS's, transfer them into the knowledge database and make sure that it has always the up-to-date index. This implementation is also developed in Python and is divided into two modules. The OPC UA module is expecting two static parameters on start. The IP-addresses of the OPC UA Discovery Servers and the update interval in seconds. The second parameter indicates how often the *Meta Data Collector* should check if the registered OPC UA servers or AAS´s has changed, to update the knowledge database. The heartbeat determines the execution of the application and is valid for both modules.
Inside the collector an OPC UA client is running, calling the Method *GetRegisteredEndpoints* of every available discovery server every heartbeat. As soon as the application starts, the information models of all registered servers are indexed. Every OPC UA server gets its own index, that is used in the Elastic database and is constructed from OPC endpoint URL by replacing all, from Elastic, forbidden characters ($'/' and' :'$), with an underscore. For example an index can look like: *opc.tcp_127.0.0.0.1_4840*. Since the depth of the information models is not known, they are searched recursively down to the leaves and the Nodes are exported. The *Meta Data Collector* is only indexing the Object, Variable and Property Nodes of the OPC UA information model. The information that is collected form every node and is searchable through Elastic is:

List 1: Listing of OPC UA Meta data

- BrowseName
- DataType
- Description
- NodeId consisting of NamespaceIndex and Identifier
- NodeClass
- NodeIds of childs
- Server endpoint with security configuration

Especially the NodeId´s of child nodes are an important information when searching for suitable data, respectively the concrete value. When looking at the following sample information model (Figure 5), one can see, that the values holding different temperature values of different meaning, are stored in the children of the object Temperature. If one just want to subscribed the temperature-object it will fail.

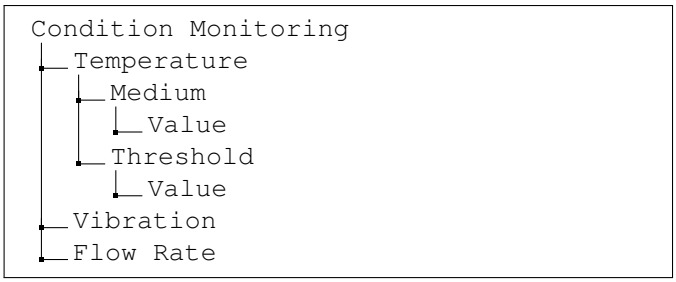


Fig. 5: Sample Informationmodel snippet

As a consequence, this means that a search for ”Temperature” without the knowledge of possible children of this node cannot provide a suitable result. Therefore it is important to store the NodeId's of the children of nodes in the index to include them in the search process or search result.
At the next heartbeat, either the list of registered servers may or may not have changed. If it has changed, it is clear that the index of the knowledge database also needs to be updated by adding or deleting the data of the server that has changed. In addition to these obvious changes, the information model of the already indexed servers may also have changed, because nodes may have been added or removed at runtime. To determine whether the information model of a server has changed, compared to the one already indexed, a naive approach was taken and the number of nodes was compared. This approach is naive in that, it cannot determine whether the values of the individual attributes have changed. This case is deliberately not covered because it requires a comparison of all attributes or their values, which becomes too voluminous, when the information models are of a certain size. Nevertheless it is possible to update just all indexes independently, if the criteria has found changes or not. The sequence diagram in figure 6 is showing the operation of the OPC UA Meta Data collection.

The collection of the AAS information is much easier, than for the OPC UA data. All available JSON´s of AAS are parsed to an JSON Object. This JSON Object is stored in the knowledge database and can be then searched. Additionally

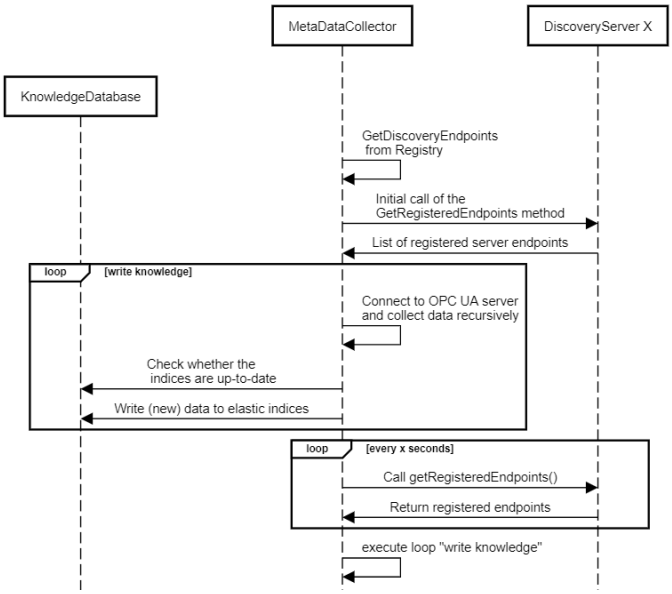


Fig. 6: Operation of OPC UA Meta Data collection

the number of JSON attributes is stored to be able to apply the same criteria for updating the database as for the OPC UA data. If the AAS is not modeled as an OPC UA server it is checked if the AAS Model is somehow linked to an OPC UA server. This information is important to subscribe the data out of the OPC UA Server, that was described in the AAS.

*2) **Schema of Meta Data**:* There is a schema for the OPC UA meta data that is used for storing in the *Knowledge Database*. The schema a is highly connected to the variables listed in List 1. In addition to them, there are fields from Elastic as example the *_index* and fields that getting filled through the *Meta Data Collector*. At the moment these fields area: addressSpaceSize, encryption and secuirtyMode. An example OPC UA JSON using the schema is shown in Listing 1. Some braces and attributes where removed because of space. For the AAS available as a file no schema was defined, because the AAS are directly stored in their JSON form in the Elastic database.

*3) **Operation Registry**:* The *Operation Registry* provides an interface through which the other components of the platform can retrieve the information for a seamless execution and operation. As the other components, this part is written in Python and is offering the following functions:

- **ReturnPlatformHeartbeat**: When receiving the request the registry is returning the heartbeat of the platform. The value has to be set on the startup of the platform via an environment variable. The *Meta Data Collector* is asking for this value on its initialization. It is send in seconds and in a small JSON string. This value will be processed by the *Meta Data Collector* and the value is set accordingly.
- **AddDiscoveryEndpoints**: This is an external interface where new discovery endpoints can be registered at the knowledge platform. It is expecting the data over a REST call as the other functions and as a JSON. The application is checking which endpoints are already there and is

Listing 1: OPC UA Meta Schema

```
"_index": "opc.tcp_10.105.4.41_48520",
"_source": {
  "__module__": "SchemaRepresentation",
  "node": {
    "browseName": "Switching status",
    "dataType": "BOOL",
    "descritpion": "Module active or not",
    "displayName": "Switching status",
    "nodeClass": 1,
    "nodeId": {
      "identifier": 1647,
      "namespaceindex": 2,
      "nodeidtype": 2
    },
    "children": [
      {
        "identifier": 1622,
        "namespaceindex": 2,
        "nodeidtype": 2
      }
    ],
    "server": {
      "addressSpaceSize": 245,
      "encryption": [
      "http://opcfoundation.org/UA/
      SecurityPolicy#None"
      ],
      "endpoint":"opc.tcp://10.105.4.41:48520",
      "securityMode":[
        1,
        3,
        2
      ]
```

adding them to the registry. The opposite call *RemoveDiscoveryEndpoints* is deleting the listed endpoints form the registry.

- **GetDiscoveryEndpoints**: On this call the *MetaDataCollector* is getting the list of the current registered endpoints in a JSON that has the same structure as above. Then the collector is processing the discovery servers as described.
- **AddSubscribedOPCUAData**: Here the already subscribed OPC UA Data is logged to the registry. A JSON containing the NodeID´s and corresponding OPC UA sever endpoints is sent and added to a database maintained by the registry. The *RemoveSubscribedOPCUAData* function behaves accordingly and uses the same JSON format. This endpoint is used by the *Data Transportation* component, which knows if pipelines has been stopped or changed. There is no comparable function for AAS data, as it is assumed that the actual process data is available in an OPC UA server.
- **GetSubscribedOPCUAData**: This method is used by the *Browser component* to find out, if OPC UA data is already subscribed. This information is handed over to the *Data Transportaion* component, which is going to create the new data pipelines.

*4) **Browser**:* The *Knowledge Browser* is a Web Application and is developed with the Django Framework [25]. When opening the App, one can put a query based on the Elastic DSL for Python [26]. It is possible to search in concrete or all fields, for substrings and connecting different search terms with and/or. Sample queries could be:

- node.browseName:Temperature: Result contains all elements where the BrowseName has the text Temperature
- node.browseName:Temperature OR node.description:Celsius: Here the Results has the name Temperature and the string Celsius in their description
- server.endpoint:opc.tcp://a.b.c.d:*: The star is the wildcard indicator in Elastic and a query formed like this will return all elements where the server string has this format. With this for example you can find all data provided on the same machine.

Due to the fact that there is no schema for AAS that have not been modelled via an OPC UA information model, it is possible to search freely in the AAS data.
Elastic is returning the full JSON of every found element of its database. Then the WebApp starts itself, in case of OPC UA Data, to query the OPC UA children of these elements. This is done until all elements and children, that matching the query, are found. The full JSON´s are stored temporary, but in the WebApp just selected information are displayed. Therefore the JSON is parsed for the name of the Value, the NodeID, server endpoint and description. In the result page, all data points respectively the linked meta data of them are displayed, in the same structure as they are in the OPC UA information model. Figure 7 displays the style of the result page for a sample query.

Listing 2: JSON describing selected data

```
[
    {
        "database": "Influx",
        "endpoint": "opc.tcp://10.105.4.41:48520",
        "identifier": "1686",
        "name": "TemperatureValue",
        "namespaceindex": "2",
        "nodeidtype": "2"
    }
]
```

Some of the displayed results has additional checkboxes. The reason is, that one can only subscribe Variables and Properties of OPC UA. The WebApp is already recognizing these elements and enables the selection field only for these kinds of nodes and is disabling it for Objects. With the confirmation by clicking on the "Select" Button a JSON string displayed in Listing 2 is generated and send to the *Data Transportation* component.

## VI. Conclusion and Outlook

In summary, the solution provides components that enable digital services and users to find and receive necessary process data from the shopfloor. It indexes existing data, makes them search-able and passes them on to the final destination. The main technology for the integration of data into the knowledge exploration is OPC UA. Additional knowledge is added through the Asset Administration Shell. It is continuously

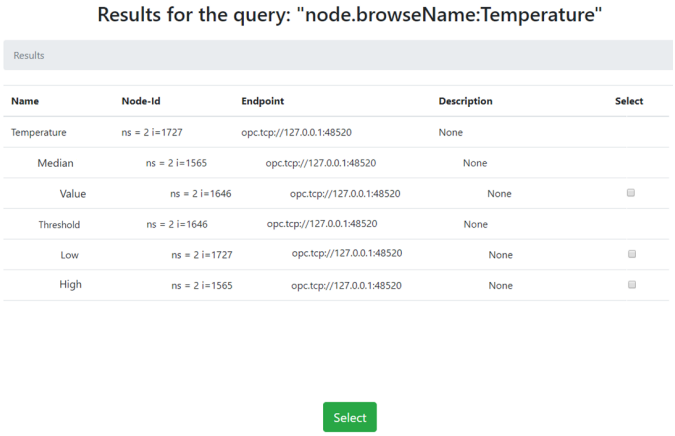


Fig. 7: Resultpage of the WebApp

checked if the data on the shopfloor is changing. These changes are synchronized with the knowledge databases. The Elastic database is storing the knowledge respectively meta data. The Elastic API allows to query the databases by users and services. The services can directly communicate with the databases and Users can use the presented Web Tool. Chosen data, necessary configuration information and the target services is send to the Middleware to establish the data flow. This part is only briefly covered in the paper to depict the whole story and is still under development. Here the containers for the OPC UA data subscription and transport are created dynamically and maintained. This pipeline will ensure that the data will be transmitted from the shop floor to their designated destination. Finally the services can use the selected data for gaining a value. The whole operation of the solution is registered to make sure, that no resources are wasted, by duplicating existing data subscriptions or transmissions.
Next steps will be the integration of AAS of type 2 and 3. For type 2 AAS a dedicated discovery mechanism is necessary to collect information out of them. For type 3 AAS, direct integration into the knowledge platform is being sought. However, the efforts to achieve this are still in their first stages and are current research topics.

## ACKNOWLEDGMENT

Parts of this article were developed within the project Industrial Automation Platform of the excellence cluster itsOWL (Intelligente Technische Systeme OstWestfalenLippe).

## REFERENCES

[1] S. Ferber. (2012) Industry 4.0–germany takes first steps toward the next industrial revolution.
[2] M. Baily and J. Manyika, “Is manufacturing “cool” again,” *Project Syndicate*, vol. 21, 2013.
[3] K. Vandikas and V. Tsiatsis, “Performance evaluation of an iot platform,” in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE, 2014, pp. 141–146.
[4] D. Kiel, C. Arnold, M. Collisi, and K. Voigt, “The impact of the industrial internet of things on established business models,” in *Proceedings of the 25th international association for management of technology (IAMOT) conference*, 2016, pp. 673–695.
[5] F. Pethig, O. Niggemann, and A. Walter, “Towards industrie 4.0 compliant configuration of condition monitoring services,” in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, July 2017, pp. 271–276.
[6] L. Monostori, “Cyber-physical production systems: Roots, expectations and r&d challenges,” *Procedia Cirp*, vol. 17, pp. 9–13, 2014.
[7] M. Wollschlaeger, T. Sauter, and J. Jasperneite, “The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0,” *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, March 2017.
[8] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Springer Science & Business Media, 2009.
[9] “OASIS, MQTT Version 3.1.1,” ”http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html”, ”[Online; accessed May 15, 2019]”.
[10] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (CoAP),” *RFC*, vol. 7252, pp. 1–112, 2014.
[11] W. Sadiq, “System and method for automated code generation using language neutral software code,” Aug. 28 2007, uS Patent 7,263,686.
[12] L. Bass, I. Weber, and L. Zhu, *DevOps: A software architect’s perspective*. Addison-Wesley Professional, 2015.
[13] C. Toro, I. Barandiaran, and J. Posada, “A perspective on knowledge based and intelligent systems implementation in industrie 4.0,” *Procedia Computer Science*, vol. 60, pp. 362–370, 2015.
[14] A. Bagozi, D. Bianchini, V. De Antonellis, A. Marini, and D. Ragazzi, “Interactive data exploration as a service for the smart factory,” in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 293–300.
[15] D. Lang, S. Grunau, L. Wisniewski, and J. Jasperneite, “Utilization of the asset administration shell to support humans during the maintenance process,” in *17th International Conference on Industrial Informatics (IEEE-INDIN 2019)*, Helsinki, Finland, Jul 2019.
[16] J. Bock, C. Diedrich, A. Gössling, R. Hänisch, A. Kraft, F. Pethig, O. Niggemann, J. Reich, F. Vollmar, and J. Wende, “Interaktionsmodell für industrie 4.0 komponenten,” in *Entwurf komplexer Automatisierungssysteme : EKA 2016 ; Beschreibungsmittel, Methoden, Werkzeuge und Anwendungen ; Magdeburg : Inst. für Automation und Kommunikation e.V.*, 2016.
[17] “I4AAS – Industrie 4.0 Asset Administration Shell,” ”https://opcfoundation.org/markets-collaboration/i4aas/”, ”[Online; accessed January 17, 2020]”.
[18] *OPC Unified Architecture - Part 7: Discovery and Global Services(IEC/TR 62541-12:2018)*, IEC Std., 2018.
[19] “Details of the Asset Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0,” Plattform I4.0, 2018.
[20] “Semantik der Interaktionen von I4.0-Komponenten,” AUTOMATION – Leitkongress der Mess- und Automatisierungstechnik, baden-Baden,2018.
[21] “AAS PackageExplorer,” ”https://github.com/admin-shell/aasx-package-explorer”, ”[Online; accessed January 17, 2020]”.
[22] O. Roulet-Dubonnet, “Python OPC-UA Documentation,” Dec. 2018.
[23] S. K. Panda, T. Schröder, L. Wisniewski, and C. Diedrich, “Plug produce integration of components into opc ua based data-space,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sep. 2018, pp. 1095–1100.
[24] “Elasticsearch,” ”https://www.elastic.co/de/products/elasticsearch”, ”[Online; accessed January 17, 2020]”.
[25] “Django,” ”https://docs.djangoproject.com/en/3.0/”, ”[Online; accessed January 17, 2020]”.
[26] “Elasticsearch Python,” ”https://www.elastic.co/guide/en/elasticsearch/client/python-api/current/index.html”, ”[Online; accessed January 17, 2020]”.