ACTA TECHNICA JAURINENSIS
Vol. 16, No. 2, pp. 34-41, 2023
10.14513/actatechjaur.00689

Data Integration Framework to Collect Data from OT/IT Systems

Balázs Szűcs

AUDI HUNGARIA Zrt
Audi Hungária út 1., 9027 Győr, Hungary
e-mail: balazs.szucs@audi.hu

Submitted: 27/02/2023 Accepted: 10/05/2023 Published online: 31/05/2023

Abstract: Industry 4.0 and industrial data processing, due to its inherent possibilities, is gaining more and more emphasis in production companies these days. In a corporate environment, the age of equipment is extremely heterogeneous, in addition to state-of-the-art equipment, legacy systems can also be found in the machine park, which do not have appropriate communication protocols. Also, with the increase in the number of data sources, the management of data is becoming more and more challenging. Not only the operational technology, but also the connection of different IT systems and the extraction of data pose challenges. The different data processing use-cases using partly or entirely the same data sources, so it is necessary to extract and transmit the data to the target systems in a standard way, and avoiding an increase in the number of point-to-point interfaces. In this work we present a possible framework, to solve the above mentioned problems in industrial environment, with the introduction of standardized naming conventions, OT/IT gateways, data integration and distribution layers.

Keywords: Data acquisition; data integration; industry 4.0; MQTT; operational technology

I. INTRODUCTION

Industrial data acquisition involves collecting and analyzing data from various industrial sources such as sensors and machines to improve productivity, efficiency, and safety. Predictive maintenance, lifetime prediction and intelligent quality assurance systems represent an enormous opportunity for manufacturing companies. Significant financial savings can be achieved by reducing maintenance and scrap costs and increasing quality. Advanced technologies and methods are available to clean, process and analyze data, but the data must first be collected.

The data collection methods can include direct connections between sensors and data acquisition systems, as well as wireless or wired communication protocols [1]. Data can also be collected and processed using edge computing technologies, where data is processed at the edge of the network, closer to the data source [2]. Industrial data acquisition methods also utilize data storage systems, such as databases and data lakes, to store and manage large volumes of data [3]. By implementing effective industrial data acquisition methods, organizations can improve their ability to monitor and optimize industrial processes in real-time. Message brokers such as Apache Kafka [4] and RabbitMQ [5] are commonly used in industrial data acquisition due to their high throughput and low latency. These message brokers provide features such as message persistence, fault tolerance, and scalability, which are important for handling large volumes of data in industrial settings. By using message brokers in industrial data acquisition, organizations can improve their ability to monitor and optimize industrial processes in real-time[6].

Besides the challenges of data acquisition methods, the age of equipment is extremely heterogeneous. The life cycle of a product can reach up to 10-15 years, and this often means the age of the production line as well. In addition to state-of-the-art equipment, the machine park also includes legacy systems that do not have the necessary communication capabilities for large-scale data collection. One of the challenges is extracting data from legacy systems and converting it into the appropriate form described by data governance.

With the increasing number of data sources the identification of the data and their sources becomes challenging. The age of the machine park also has an impact on the identification of machines. As a result of poor change tracking of the naming of machines, their physical identification and the identification of the machines in the IT system can differ. On the other hand, the ID of a specific machine can be

different and ambiguous in different IT systems. These factors have a great influence on the identification and assignment of the related data sources and datasets. Industrial equipment naming conventions provide a standardized approach for naming equipment in an organization. One method is the use of a code or numbering system that identifies equipment by type, function, and location [7]. Another method is to use acronyms or abbreviations that have meaning for members of a group or organization [8]. A third method is the use of a descriptive name that is based on the function of the equipment [9]. A combination of these methods can also be used to provide a unique and standardized name for equipment [10]. The selection of a naming convention method depends on the specific needs of the organization and the industry it operates in.

Another relevant aspect of the data collection is the manageability of interfaces of the IT systems and the network performance. Different types of datasets are stored in different IT systems, like part tracking systems, machine and process data systems, and quality databases. The different data processing use-cases use partly or entirely the same data sources. If we use separate interface between every system and use-case, the number of connections can grow rapidly, according to equations (1), where n is the number of nodes.

$$n\frac{(n-1)}{2} \tag{1}$$

To avoid the point-to-point interfaces between the separate data processor applications and the different IT systems, the data extraction and distribution need to be standardized. In the next section we describe the architecture and the rules of the data integration framework.

II. Data Integration Framework

The data integration framework is a set of architectural components and rules which propose a solution to the collection, identification and distribution of data. The main parts of the architecture (**Fig. 1.**) are:

- Controller level
- Gateway level
- Distribution level
- Application level

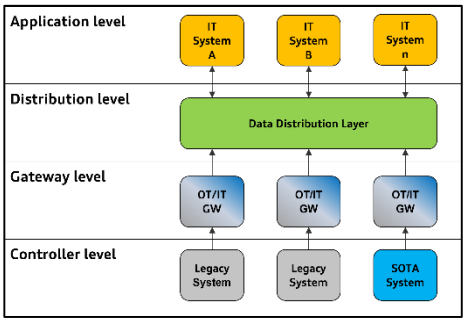

Figure 1. *Data integration framework architecture*

The communication of the components can be bidirectional, but this is not mandatory.

The controller level contains the source systems like programmable logic controllers (PLCs), numerical control units (NCUs), Internet of Things (IoT) capable devices and other automation hardware. The state-of-the-art (SOTA) components are capable to communicate event driven, with higher level protocols like MQTT, but in some cases polling and transformation of the messages are necessary.

The function of the gateway level is to physically separate the operation technology network from the corporate network. Besides security, this level can have other responsibilities, like buffering the incoming messages in a case of network failure, or polling the legacy devices and hosting data acquisition agents and translation of the protocols.

The goal of the distribution layer is to forward the messages to the target systems. One advantage of this architecture, that the source system does not need to know the receiver, it has only send the message to the distribution service and it forwards the message to the designated system, which is subscribed to the data source.

The application level contains the legacy IT systems and other use-cases, which processes the data from the controller level and related IT systems. These components communicates through the distribution layer, this way the point-to-point interfaces between system can be avoided and the

Table 1. *Standardised naming convention*

Global ID	*Business Unit ID*	*Domain*	*Unit*	*Subunit*	*Component*
01	P	Domain A	MG0012	MA001	MS01
02	L	Domain B	TU0123		
01	F	Domain C	AE0200	CP012	SS01

data can be used by other systems as well, no need the duplicate the data through distinct interfaces.

The architecture alone cannot guarantee the reliable and manageable message flow between the components, further rules are needed to manage the communication. These rules are defined by the data governance and are the following:

- Standardized naming convention of data sources
- Standardized message structures
- Distinct channels for predefined message types

1. Standardized Naming Convention

The source of the data needs to be clearly identified[11][12] to forward the information to the corresponding data processor and to connect the related data. To achieve this behavior, we introduced a standardized naming convention of the source components. The nomination of the components not only identifies the source, but contains additional information like the location, hierarchy and type of the unit. The naming convention capable to identify other assets too, like buildings, halls, facility equipment, logistical vehicles, storages. The asset management is not scope of this work.

The coding consists of six level arranged to tree structure (**Fig. 2.**), each level identifies a separate entity of the hierarchy. The usage of every level is not mandatory, but the notation must follow the top-down structure, starting with the global ID. **Table 1.** shows the structure of the standardized naming convention and three examples (top-down, read left to right):

- Measuring system 1 of Machine 1 in Machine Group 12 of Domain A, Business Unit: Production, Factory: 01.
- Tow unit 123 of Domain B, Business Unit: Logistics, Factory: 02.
- Speed sensor 1 of Compressor 12 in Air Engineering 200 of Domain C, Business Unit: Facility Management, Factory: 01.

The notation of components is standardized in a code library. The delimiter of the sections is arbitrary, depends on the use-case or the system, which processes the data.

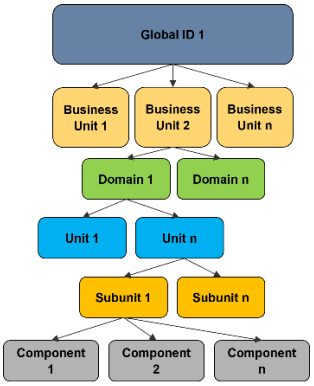

Figure 2. *Tree structure of the standardized naming convention*

2. Standardized Message Structure

In addition to the standard naming convention, a standard message structure is strongly recommended. The uniform structure of the messages makes the data acquisition and the message protocols independent, allows the exchange of the underlying transmission protocol without disrupting the data flow, and makes it easier to manage the collected data.

To connect to the data distribution framework, the participating system only have to utilize the standardized message structure. If the source system meet the requirements of the message structure, the technology of the data collection and the transmission protocol can be arbitrary.

The standardized message structure is based on the JavaScript Object Notation [13] (JSON) format. JSON is a lightweight, self-describing textual object. The textual format makes it possible to interpret the data in a programming language independent way, therefore it is used to store or send data between computers or programs.

The mandatory content of all of the messages are the source identification, the message timestamp, the message version and the counter of lost messages. The related dataset are assigned to predefined channels or topics, thus all other content depends of the message type. An example of the JSON message shown on **Fig. 3**.

Table 2. *The use of standardised naming convention in message topics*

Topic subscription	***Meaning***
01/P/#	Subscription to all elements and topics in Factory 1, Business Unit Production
01/P/DomainA/#/#/Energy	Subscription to Energy topic, all units and subunits in Factory 1, Business Unit Production, Domain A

```
{
„SourceID": „01/P/DomainA/MG0012/MA001",
„TopicRelatedData1": „Data1",
„TopicRelatedData2": „Data2",
„TopicRelatedData3": 123,
…,
„MessageTimeStamp": "2022-12-11T22:54:00.000+01:00",
„MessageVersion": „Example_01.0",
"LostMessages":"0"
}
```


***Figure 3**. An example of the JSON message*

Another good practice is to organize the message topics in a way, which utilize the standardized naming convention. With this method all hierarchy level of the naming convention and all message types are accessible. The selection of multiple elements is possible with wildcards (#). **Table 2.** shows different topic subscriptions and their meanings.

3. Data Distribution Layer

In order to transfer the collected data between the source and the destination, a data distribution layer is needed. The industrial use-cases requires scalable, loosely coupled and dynamic network topology, thus the publish–subscribe (pub-sub) messaging pattern [14][15] is used.

In pub-sub messaging, the publisher (source) does not need to know, who is the subscriber (destination), it only has to publish the messages to the data distributions layer in to the related topic, then the service forwards the messages to the corresponding subscribers who subscribed to that specific topic. This features ensures loose coupling and scalability of the pub-sub systems. Topics[4][5] are logical channels of related datasets, a subscriber receives all the data, which are published to the subscribed topic. The participants can be publishers and subscribers at the same time or only one of them. Messaging actions are not restricted to one topic, as well as publishing and subscribing can also be done on different topics.

The main advantage of this architecture, that the number of communicating system is highly scalable without the introduction of further point-to-point interfaces, thus the architecture remains transparent and manageable. **Fig. 4.** shows conventional interfacing (left) and an interfacing with pub-sub data distribution layer.

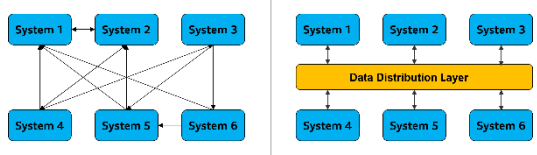

***Figure 4.** Conventional interfacing (left) and interfacing with data distribution layer (right)*

III. Practical Implementation of the Framework

In this section we present a practical implementation of the data acquisition method described in the previous sections. In the experimental setup we collected the number of the working tool, the desired and remaining workpiece count of the tool, the z-axis position, the main spindle feed and current of an 3-axis turning machine.

The framework is also usable with state of the art and legacy IT systems. Some of the system are natively capable to communicate with message brokers, but there are cases when protocol translation and interfacing required, thus the usage of agents cannot be avoided.

There are other cases, when the source IT system cannot provide the source ID in accordance to the standardized naming convention, in this case the translation of the source ID requires the usage of agents too.

The following practical implementation presents the framework usage in case of legacy OT systems, but in case of legacy IT architectural setup of the framework is the same except from the source system. If the source IT system is capable to communicate with the message broker and can also provide the source ID in accordance to the standardized naming convention, the usage of the agents are avoidable.

1. Architecture of the Experimental Setup

The architecture of the experimental (**Fig. 5.**) setups contains the elements described in Section II.

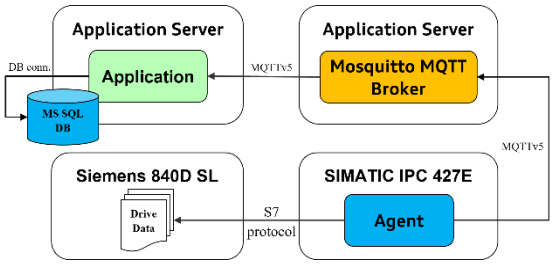

***Figure 5.** Architecture of the practical implementation*

In the controller level we used a Siemens 840D SL [16] NCU with integrated S7-300 PLC. This device is capable to communicate through Profinet on Industrial Ethernet.

For the OT/IT Gateway we chose a SIMATIC IPC 427E [17] industrial PC with Ubuntu 20.04.5 LTS operating system, which besides of the hosting of the data acquisition agent, responsible for physical separation of OT and IT systems. The OT/IT Gateway can host multiple agents in different containers like Docker or LXC containers, and the

gateway can be a remote server too. For the simplicity, in this work we used one physical hardware and one agent.

The data collection agent is a .NET [18] application written in C#, for the communication we used the Sharp7 [19] library, which implements the S7 Protocol [20].

For the data distribution layer we used an Eclipse Mosquitto MQTT broker [21] hosted on a Windows application server.

To store the data we created a simple .NET middleware, which subscribes to the specific topics and stores the data to a Microsoft SQL Server 2019 [22]. The middleware runs on a separate Windows server.

2. Data Collection

In the experimental setup we collected the tool number of actual working tool, the desired and remaining workpiece count of the actual tool, the main spindle feed and current in percentages, referring to the maximal current of the drive, and the position of the Z-Axis. The collected variables [23] shown in **Table 3**.

The sampling speed is based on the speed of the communication, which depends on the hardware type of the numerical control unit, and the user program, but cannot be faster than the smallest theoretical cycle time of the PLC. In this case, the sampling rate of the data acquisition agent is 500 milliseconds. The agent sends the collected data to the MQTT broker, the topic where the agent publishes the messages are the subtopics of "01/P/TestDomian/MG0001/MA001/…", namely "DesiredWorkpieceCount", "RemainingWorkpieceCount", "MainSpindleFeed", "MainSpindleCurrent" and "ZAxisPos".

3. Data Integration Layer

In the data integration layer we used an Eclipse Mosquitto MQTT Broker, hosted on an application server. For the simplicity of the setup, we only utilized the minimally necessary settings of the broker. We used the standard ports, 1883 for unsecure, 8883 for secure connection with Transport Layer Security. For the client to connect to the broker, we created an username and a password, and in the access control list (ACL) we defined which topics can the client access. The broker operates in retain mode, which means if a new client subscribes to a topic, the broker sends the last received message to the client in that topic. The quality of services (QoS) is set to QoS 0, which means "fire and forget", the broker sends the messages to the clients exactly once, without the need of confirmation if the message is arrived. This setting enables to communicate with the lowest latency. QoS 2 and QoS 3 are also available, with QoS 2 the message will be delivered at least once with the need of confirmation, with QoS 3 the broker send the message exactly once and requires a handshake mechanism with the clients. To ensure transparency and to help the debugging, logging is also enabled on the broker.

4. Middleware and Database

The middleware is a .NET application written in C#. For the MQTT connection we used the MQTTnet [23] library. The middleware subscribes to the corresponding topics and writes the data in an Microsoft SQL Database. The application uses the Entity Framework [25] and Data Transfer Objects (DTOs) to map the classes of the application to the database tables.

The database is "code first", which means the database tables are created based on the classes of the application, this feature and the Entity Framework also enables to the usage of the strongly-typed access to the data with LINQ [26]. With LINQ the data is easily accessible and manipulatable from the code. In this case the application stores the data without manipulation. The data from different topics are stored in different tables in the SQL Database. The database tables columns are ID (incremental ID as primary key), SourceID (Client ID based on standardized naming convention), TimeStamp (the timestamp of the data from the MQTT message) and the Value itself.

Table 3. *Collected NC Data*

Data	*Variable*	*Parameter*	*Machine Data*	*Format*
Actual tool number (ToolNo)	/Channel/State/actTNumber	-	$P_TOOLNO	UWord
Desired Workpiece Count	/Tool/Supervision/data[x,y]	ToolNo, 6	$TC_MOP13	Double
Remaining Workpiece Count	/Tool/Supervision/data[x,y]	ToolNo, 4	$TC_MOP4	Double
Main Spindle Feed	/DriveHsa/State/actualSpeed	-	$MD_1701	Float
Main Spindle Current	/DriveHsa/State/actualCurrent	-	$MD_1708	Float
Z-Axis Position	/Nck/MachineAxis/measPos1[axis]	3	-	Double

5. The Collected Data

As we previously stated in the beginning of section three, we collected the number of the working tool, the desired and remaining workpiece count of the tool, the z-axis position, the main spindle feed and current of an 3-axis turning machine. The following diagrams shows the above mentioned dataset of the tool T6013 during of four consecutive machining.

The main spindle feed is shown on **Fig. 6.** The X-Axis of the diagram represents the timestamps of the data in date and time format, the Y-Axis represents the feed of the main spindle in mm/s.

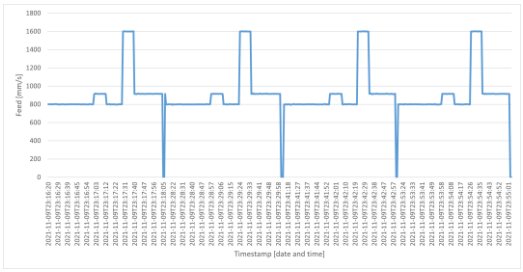

Figure 6. *Main spindle feed (X-Axis: date and time, Y-Axis: feed [mm/s]*

The absolute position of the Z-Axis is shown on **Fig. 7.** The X-Axis of the diagram represents the timestamps of the data in date and time format, the Y-Axis represents the absolute position in millimeters.

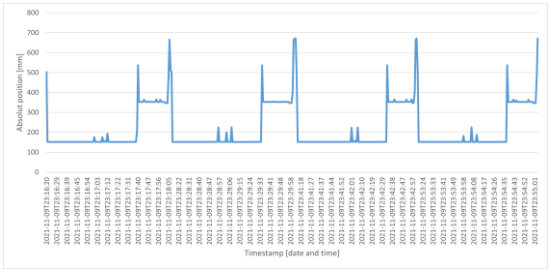

Figure 7. *Z-Axis position (X-Axis: date and time, Y-Axis: absolute position [mm])*

The main spindle current in percentage of the maximal drive current is shown on **Fig. 8.** The X-Axis of the diagram represents the timestamps of the data in date and time format, the Y-Axis represents the spindle current in percentages of the maximum current of the drive.

The count of the remaining workpiece count is shown on **Fig. 9.** The X-Axis of the diagram represents the timestamps of the data in date and time format, the Y-Axis represents the remaining number of machinable workpieces for the specific tool.

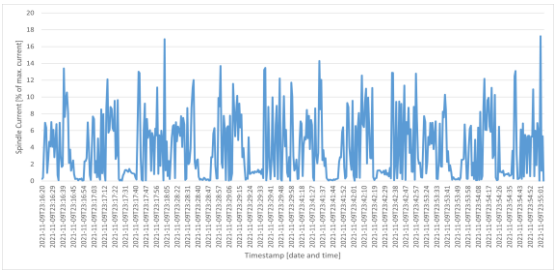

Figure 8. *Main spindle current in percentage of maximum current of the drive (X-Axis: date and time, Y-Axis: % of maximal current)*

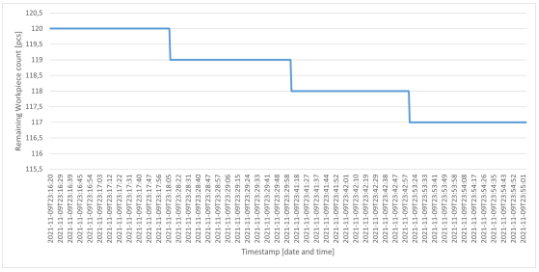

Figure 9. *Remaining workpiece count (X-Axis: date and time, Y-Axis: remaining workpiece count [pieces])*

The desired workpiece count is a specific, constant value for each tool, thus it is no depicted.

IV. Results

Based on the results of the practical implementation of the data integration framework, we rolled out the solution to an entire production line of the AUDI HUNGARIA Zrt. The pilot production line includes 36 machines, each are connected to the data distribution layer through agents as in the previous section presented.

We collected the data of the machine states, machine information like part counters and cycle time measurements, workpiece movements, operator identification information, error messages, the energy consumption, feed override of the machines and the MQTT State of the agent. The topics, where the agents are publishing the data, are based on the standardized naming convention. The base of the topics is the machine ID within the hierarchical structure of the factory, business unit, production domain and the production line, which follows the pattern: "Factory ID/Business Unit /Domain/Production line/Machine ID/Topic". The average daily number of messages for each topic and the size of each message are shown in **Table 4**.

Table 4. *Collected NC Data*

Topic	*Message Count*	*Avg. size (Byte)*
MqttState	2	128
PartMovement	10010	234
Energy	42657	147
MachineState	15504	131
MachineInfo	10010	240
Messages	37053	93
Operator	2	188
Override	251	130

The average daily message count of the 36 machines is 115000 messages. The messages are stored in an SQL Database for further analysis and visualization tasks.

V. Summary and Future Work

In the previous sections we presented a data collecting framework to collect data from OT/IT systems and prevent interface jungle, thus simplify the architecture of a corporate network and enable new data processing use-cases. The framework enables to collect data from legacy OT and IT system, that are unable to use state-of-the-art communication protocols or meet data governance requirements.

With the proposed elements, like the standardized naming convention and the usage of data collection agent and the data distribution layer, the connection of related data can be simplified and the difficulties caused by the poorly managed system and the lack of change management can be eliminated. The standardized naming convention can also be used as a part of asset management.

With the introduction of the data distribution layer, the point-to-point interfaces can be avoided, thus the network management and operations becomes simpler. The data distribution layer also provide transparency and traceability trough data access policies, user management and logging. Specific users or clients can only access to topics, which are enabled in the access control list of the broker, read and write privileges can be set up also, and the connection attempts of client are also logged. These functionalities also enable the conformity to IT security rules.

The framework enables the data collection from legacy systems, thus the operational and process data can be collected from heterogenous systems in a standardized way. The standardized message structure makes the data handling and storage easier, the newly connected clients only have to meet the requirements of the standardized naming convention and message structure to send data to the broker. This feature enables data storage without any further customization of the data sources. The standardized message structure also specifies the topic for the data. This property enables the clients to subscribe only to that topics, what it really needs. This function also eliminates the need for data lakes, each use-case only have to collect the data, what they really need.

In case of a new use-case needs access to the data which available on the message broker, a new user must be created on the broker and after the access right granted on the topic which the new client needs, it can subscribe to the topic and can start the data collection from the broker. This feature enables fast on-boarding of new data processing use-cases, such as machine learning models, artificial intelligence (AI) based data processors and predictive systems.

The data integration framework provides a good starting point for industrial artificial intelligent applications through simplifying the data collection, management and distribution of process and machine data, and new data collections can be easily introduced to the data distribution layer.

Further research in the processing of the collected data, for example predictive maintenance systems and AI backed quality assurance systems strongly advised.

AUTHOR CONTRIBUTIONS

B. Szűcs: conceptualization, proof of concept setup, programming, writing and editing.

Disclosure Statement

The author declare that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ORCID

B. Szűcs https://orcid.org/0000-0002-2273-027X

References

[1] Verma, N., Jain, M., & Agrawal, R. (2018). Wireless sensor networks for industrial automation: Challenges and solutions. Journal of Sensor and Actuator Networks, 7(3), 33. https://doi.org/10.1002/dac.4074

[2] Shi, W. et al. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637-646. https://doi.org/10.1109/JIOT.2016.2579198

[3] Oussous, A. et al. (2018). Big data technologies: A survey. Journal of King Saud University - Computer and Information Sciences, 30(4), 431-448. https://doi.org/10.1016/j.jksuci.2017.06.001

[4] Apache Kafka. (2021). Retrieved from https://kafka.apache.org/ [cited 2023-01-22]
[5] RabbitMQ. (2021). Retrieved from https://www.rabbitmq.com/ [cited 2023-01-22]
[6] Riedel, E. (2022). MQTT protocol for SME foundries: potential as an entry point into industry 4.0, process transparency and sustainability. Procedia CIRP. 105. https://doi.org/10.1016/j.procir.2022.02.100.
[7] Li, Z., Li, B., & Li, J. (2019). Research on equipment coding and naming system for power plants based on the characteristics of the equipment. IOP Conference Series: Materials Science and Engineering, 664(1), 012138.
[8] Das, A., & Akbar, R. (2019). A comprehensive naming convention for industrial components. In Advances in Manufacturing and Mechanical Engineering (pp. 281-289). Springer, Singapore.
[9] Gómez, J. A., & Santana, R. (2020). Development of a descriptive naming convention for industrial equipment. Industrial Management & Data Systems.
[10] Chung, T. W., Chen, C. M., & Yeh, C. T. (2015). Development of a rule-based naming convention for equipment in a semiconductor manufacturing fab. IEEE Transactions on Semiconductor Manufacturing, 28(3), 307-314.
[11] Berners-Lee, Tim, RFC 3986: Uniform Resource Identifier (URI): Generic Syntax (2005) DOI:10.17487/RFC3986
[12] Libes, D., Choosing a name for your computer, FYI 5, RFC 1178, (1990). https://www.rfc-editor.org/info/rfc1178
[13] Bray, T., Ed., The JavaScript Object Notation (JSON) Data Interchange Format, STD 90, RFC 8259 (2017). https://www.rfc-editor.org/info/std90
[14] K. Birman, T. Joseph, Exploiting virtual synchrony in distributed systems. In Proceedings of the eleventh ACM Symposium on Operating systems principles (SOSP '87). Association for Computing Machinery, New York, NY, USA, 123–138. (1987) https://doi.org/10.1145/41457.37515
[15] Yusuf, S., Survey of publish subscribe communication system, Advanced Internet Application and System Design (2004)
[16] Technical Documentation for SINUMERIK 840D sl, Version 4.92, Retrieved from https://support.industry.siemens.com/cs/document/109768584/technical-documentation-for-sinumerik-840d-sl-version-4-92?dti=0&lc=en-DE [cited 2023-01-22]
[17] SIMATIC Industrial PC SIMATIC IPC427E Operating Instructions A5E37454814-AE, Siemens (2021)
[18] .NET, Retrieved from https://dotnet.microsoft.com/en-us/ [cited 2023-01-22]
[19] Sharp7 Library, Retrieved from https://snap7.sourceforge.net/sharp7.html [cited 2023-01-22]
[20] S7 Protocol, Retrieved from https://wiki.wireshark.org/S7comm [cited 2023-01-22]
[21] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," The Journal of Open Source Software, vol. 2, no. (2017) https://doi.org/10.21105/joss.00265
[22] Microsoft SQL Server 2019, Retrieved from https://www.microsoft.com/en-us/sql-server/sql-server-2019 [cited 2023-01-22]
[23] SINUMERIK 840D sl/840Di sl, SINUMERIK 840D/840Di/810D List of System Variables, Parameter Manual (2006), Retrieved from https://cache.industry.siemens.com/dl/files/272/28713272/att_92132/v1/PGA1_1106_en.pdf [cited 2023-01-22]
[24] MQTTnet Library, Retrieved from https://github.com/dotnet/MQTTnet [cited 2023-01-22]
[25] Entity Framework, Retrieved from https://learn.microsoft.com/en-us/ef/ [cited 2023-01-22]
[26] Language Integrated Query (LINQ), Retrieved from https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ [cited 2023-01-22]