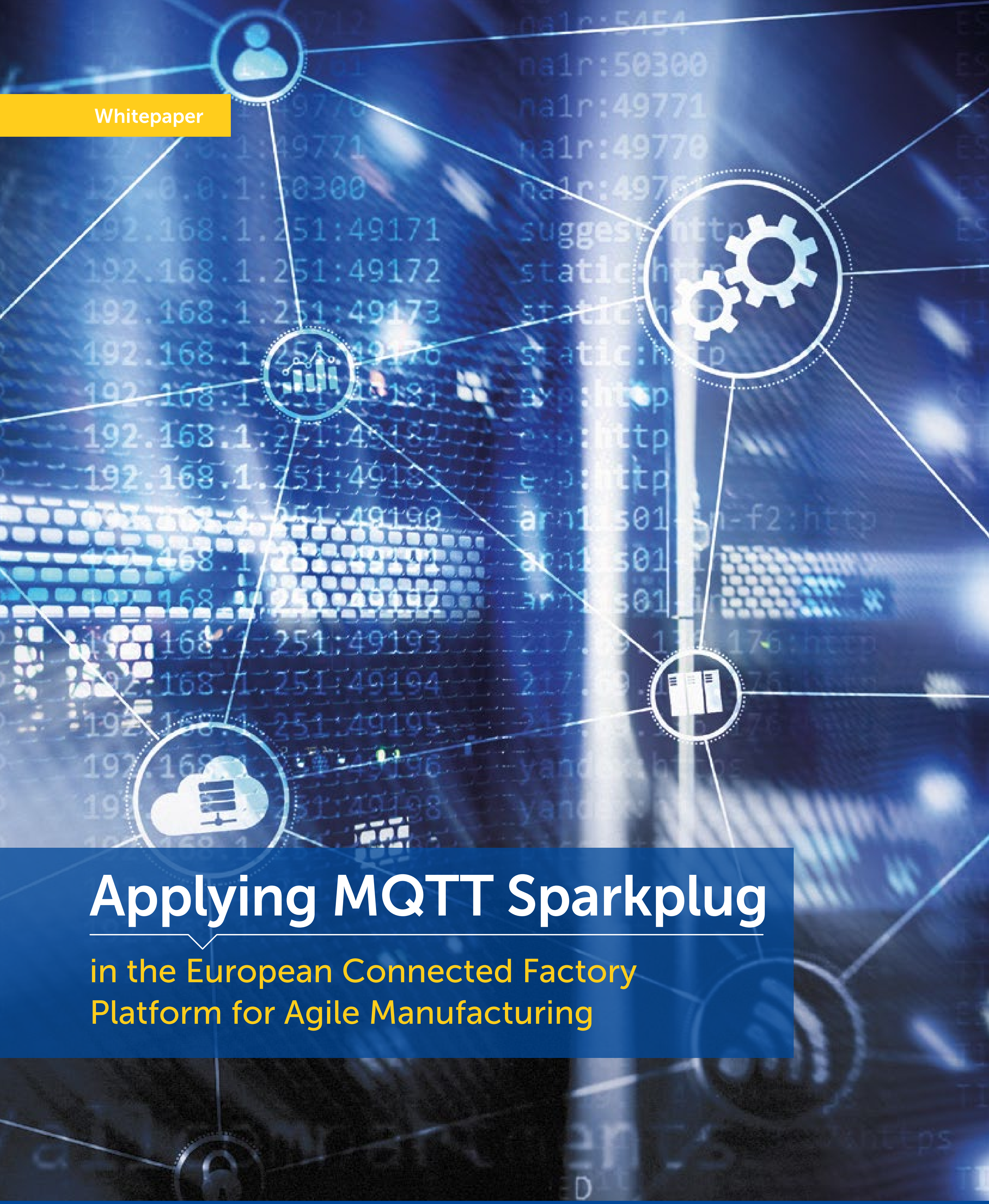
Whitepaper



Applying MQTT Sparkplug

in the European Connected Factory Platform for Agile Manufacturing

fortiss

Authors

Nisrine Bnouhanna
fortiss GmbH
Munich, Germany
bnouhanna@fortiss.org

Rute C. Sofia
fortiss GmbH
Munich, Germany
sofia@fortiss.org

Edoardo Pristeri
LINKS foundation
Turin, Italy
pristeri@links.foundation.com

Abstract – This white paper provides an overview on the need and application of MQTT Sparkplug within the context of the European Connected Factory Platform for Agile Manufacturing. The white paper is directed towards researchers, developers, system integrators interested in understanding the role of MQTT Sparkplug and its operational use in manufacturing environments.

Keywords – Industrial Internet of Things; Networking Architectures; Network Protocols; Operational Technology - Information Technology; Semantic Interoperability; EFPF

Contents

Introduction

Production environments are currently supported by multiple heterogeneous *Industrial Internet of Things (IIoT)* platforms. These platforms comprise different technologies (e.g., semantic technologies) to assist different communication protocols interoperably to allow data exchange from Edge to Cloud *Machine Learning* to support automation aspects.

Data exchange is supported by different communication protocols, and the most popular protocols in manufacturing environments include the *OPC Unified Architecture (OPC UA)* [1] and the *Message Queuing Telemetry Transport (MQTT)* protocol [2].

Both protocols exhibit advantages and disadvantages considering design and performance [3, 4]. Overall, OPC UA has been developed to support in-plant communication, while MQTT has been originally established to interconnect oil pipes over unreliable satellite networks. This motivation produced a lightweight, bandwidth-efficient protocol that can integrate some levels of *Quality of Service (QoS)*.

MQTT is intentionally adapted to support an end-to-end IoT interconnection, where the goal is to interconnect machines between a shop floor and the Cloud. Meanwhile, OPC UA has been devised to support communication within the shop floor based on a client–server approach. A key drawback of OPC UA is that it will not intentionally allow the isolation of a shop floor network when interconnecting to an IIoT system: an OPC UA client must open a firewall port to access data from a machine on the shop floor (OPC UA server), thus possibly exposing critical environments to attacks.

Therefore, the usual approach followed in industrial environments is to consider an OPC UA to another protocol translation via a software-based gateway (e.g., OPC UA to MQTT).

By contrast, several (claimed) MQTT disadvantages in comparison to OPC UA include insufficient encoding, limited integrated security, and lack of uniform information models, which can be solved by implementing MQTT together with Sparkplug (MQTT Sparkplug).

With the current technological evolution of IIoT systems, where intelligence is being driven to the Edge [5], assessing the best protocols for different environments is important.

Regarding an end-to-end (from Edge to Cloud) production environment, MQTT Sparkplug emerges as an interesting solution because it combines the key advantages of both protocols. As an IP-based, message-oriented lightweight protocol, MQTT provides interoperability based on the broker abstraction concept to allow for the support of mission-critical, real-time applications. Sparkplug, particularly Sparkplug B specification, introduces support for *Operational Technology (OT)* data modeling, which is essential to a flexible and interoperable shop floor interconnection.

Moreover, Sparkplug is a new specification provided by Eclipse[1], which defines a standard MQTT topic namespace, that is, payload and session management for IIoT applications, while demonstrating real-time interconnection to *Supervisory Control and Data Acquisition (SCADA)* implementations.

This white paper debates and explores the use of MQTT Sparkplug on a heterogeneous, large-scale factory platform, namely the *European Connected Factory Platform for Agile Manufacturing (EFPF)* . The white paper is directed to researchers, developers, engineers, and system integrators aiming at understanding the role of MQTT Sparkplug in the context of industrial applications.

The white paper is organized as follows. Section II provides the reader with relevant pointers to understand effectively communication protocols in IIoT, particularly MQTT Sparkplug. Section III introduces the EFPF platform and its main components. Section IV provides a brief introduction to MQTT Sparkplug, while Section V explains the EFPF MQTT Sparkplug namespace proposal. Section VI presents the use of MQTT Sparkplug on two different pilots: i) support a broad interconnection between the shop floor and different data analytics tools and ii) facilitate the interconnection of different shop floors and EFPF for environmental monitoring with the EFPF TSMatch component. The white paper concludes in Section VII, presenting challenges and advantages of the integration of MQTT Sparkplug in EFPF.

Related Work

Different IIoT applications introduce specific communication requirements and should dictate the choice of a specific IoT protocol. For instance, the reader can refer to Ramson et al. to obtain an overview of different IIoT applications [6]. Sofia et al. provide a systemic description of communication key performance indicators for IIoT applications [7].

Another category of related work focuses on the performance analysis of IIoT protocols. Silva et al. provide a comparative analysis of the most popular IIoT protocols considering networking features and present a testbed-based comparison of MQTT and OPC UA in terms of latency [4]. Naik et al. provide a broad comparison among the four established messaging protocols for IoT systems, namely MQTT, CoAP, AMQP, and HTTP, and perform a relative analysis based on some interrelated criteria to gain insight into the strengths and limitations of these protocols [8].

Nitulescu et al. present the concept and implementation of a Web-based SCADA based on Node-RED[3] [9]. The project implements an IoT system that allows the components to transfer information via Modbus/TCP and MQTT and develops the basic SCADA features for process supervision. They also mention Sparkplug B and its availability in Node-RED[4]. However, they do not explain the application of Sparkplug and its implementation.

Nipper et al. address the problem of brownfield devices integration into IT, i.e., the challenges with the integration of *OT* to *Information Technology*, and *IT* with MQTT Sparkplug, providing examples for data modeling based on MQTT Sparkplug [10].

The reader is further directed to the Eclipse Sparkplug working group[5].

Overall, MQTT Sparkplug is relevant to address the end-to-end support of IIoT flexibly. Understanding the application of MQTT Sparkplug, and determining which performance it can support is necessary.

This white paper contributes to this gap by explaining the integration of MQTT Sparkplug in IIoT use cases of the EFPF platform.

1 https://sparkplug.eclipse.org/}

2 https://www.efpf.org

3 https://nodered.org/

4 https://flows.nodered.org/node/node-red-contrib-sparkplug

5 https://www.eclipse.org/org/workinggroups/eclipse_sparkplug_charter.php

EFPF

EFPF is a federated smart factory ecosystem and a digital platform that interlinks different stakeholders of the digital manufacturing domain. The EFPF platform enables users to utilize innovative functionalities, experiment with disruptive approaches, and develop custom solutions to maximize connectivity, interoperability, and efficiency across the supply chains.

An illustration of EFPF is provided in **Figure 1**, which shows the EFPF Data Spine component as the central entity. The Data Spine interconnects external platforms, as well as four different industrial platforms, which have been derived from four prior projects, including COMPOSITION[6], DIGICOR[7], NIMBLE[8], and vf-OS[9].

The EFPF ecosystem integrates smart tools provided by different partners. These tools and services aim to cover the complete lifecycle of production and logistic processes that occur in a modern industrial environment. Examples of the tools include the following: data gateways, distributed production planning and scheduling, distributed process design, monitoring, decision support, process optimization, risk management, and blockchain-based trust and message exchange.

Data Spine, which is a component that interconnects the different applications provided by various platforms, is the core of EFPF. Data Spine offers interoperable services at the level of protocols, message formats, data structures, data models, software services, and processes ranging from field-level control to business process enactment. The interoperable security features, which ensure transparent utilization of tools and services at the EFPF platform level, encompass the Data Spine.

The interoperable data exchange in EFPF is based on the development of connectors to key industrial protocols. For asynchronous communication, EFPF relies on MQTT and the *Advanced Message Queuing Protocol (AMQP)*[10].

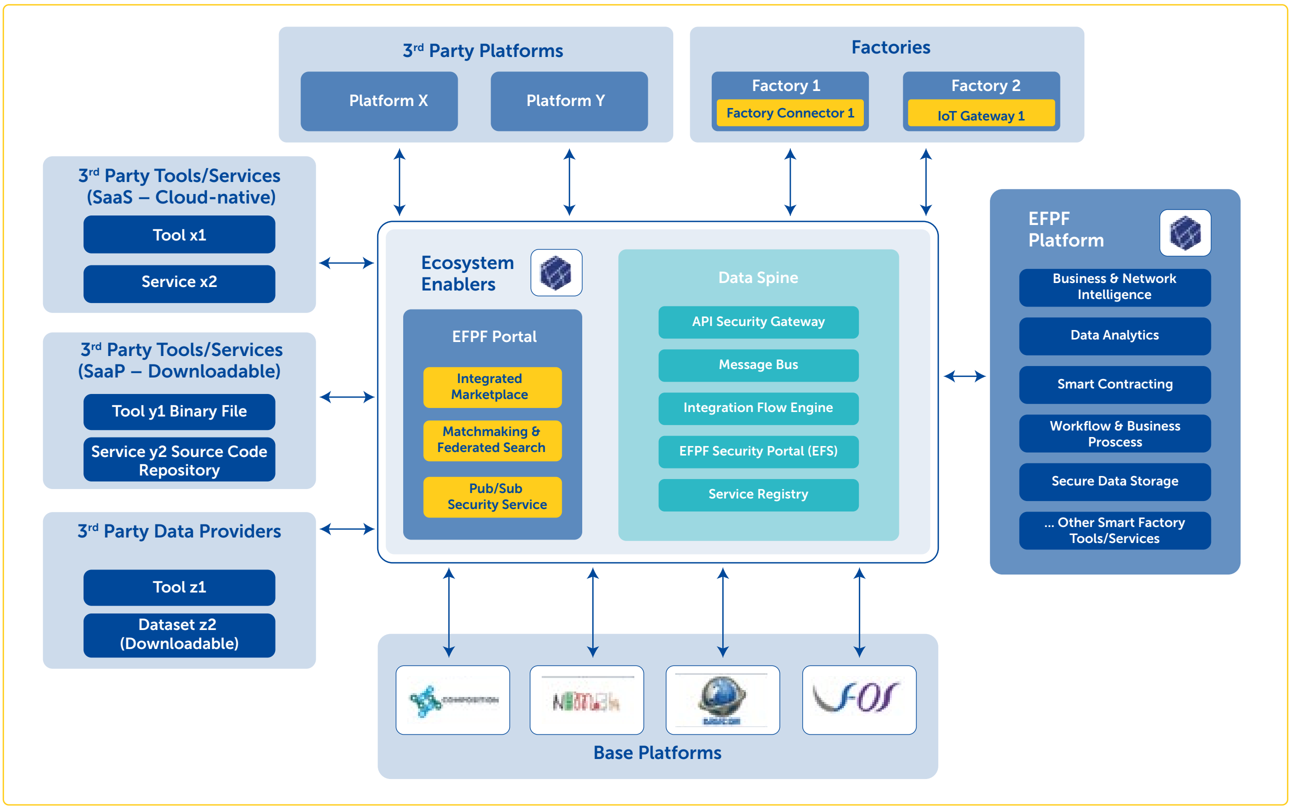


Figure 1: the EFPF architecture and its components.

The key components of EFPF[11] are as follows.

- **Data Spine.**
  This component corresponds to the core of EFPF, providing interconnection and interoperability. The Data Spine integrates services, such as single sign-on, service registration and discovery, message brokering, and dataflow management and service composition. As the interoperability backbone of the EFPF platform, one of the Data Spine focuses is to bridge the interoperability holes at three different levels between the tools that it interconnects. The Data Spine supports synchronous request–response and asynchronous publish/subscribe (PubSub) communication patterns.

- **Factory connectors/IoT gateways.**
  These components correspond to communication connectors (e.g., MQTT Sparkplug connector) and to middleware, which provides some form of data processing (e.g., matchmaking between IoT data sources and services).

- **EFPF platform.**
  This component provides a list of services that can be used by external partners[12], such as data analytics and predictive maintenance.

- **External platforms.**
  These components comprise the base and external partner platforms.

- **Integrated marketplace.**
  This component is the place where all services can be found.

- **Matchmaking.**
  The matchmaking service helps EFPF users find the best-suited suppliers from across different platforms and enables their efficient and effective transactions.

- **Message Bus.**
  This subcomponent of the Data Spine integrates the PubSub support via the integration of MQTT and AMQP.

- **Integration Flow Engine.**
  This subcomponent of the Data Spine integrates the tools to transform message contents and data models of different services connected to the Data Spine. Therefore, this subcomponent supports data interoperability aspects.

- **Security Interoperability.**
  The EFPF security portal component of the Data Spine provides a federated security layer and single sign-on capabilities to the ecosystem.

6 https://www.composition-project.eu/
7 https://www.digicor-project.eu/
8 https://www.nimble-project.org/
9 https://www.vf-os.eu/
10 https://amqp.org
11 A full perspective on all components is available via the EFPF User Guide 101, https://docs.efpf.linksmart.eu/projects/user-guide-101/
12 https://docs.efpf.linksmart.eu/projects/user-guide-101/list-of-services

MQTT Sparkplug in a NutShell

MQTT is an IP-based message-oriented protocol that relies on the notion of a mediating entity, a data broker, to support asynchronous communication between data sources (*Publishers*) and *Consumers* of the data. Therefore, MQTT provides a lightweight PubSub approach for IoT communications based on TCP. MQTT also introduces the advantage of supporting specific QoS levels. Therefore, the broker is a central (single point of failure) entity that manages the data delivery across the entire IoT infrastructure.

Within industrial environments, MQTT introduces the advantage of having the producers of data (robots, sensors) publish their data to a broker either based on periodic polling or when changes are detected. The consumers can be specific services, IoT platforms, software, or users. The consumers also register to specific topics on the broker and therefore asynchronous receive the published data, independently of the whereabouts of the producers.

Publishers and Consumers are aware of the broker's whereabouts (IP/Port, URL); however, *Publishers* and subscribers have no additional information (asynchronous communication pattern).

The data exchange is supported by *topic paths* (e.g., ***RoomA/Temperature*** (temperature in room A)). The *consumer* subscribes to a specific topic to obtain specific data. Therefore, a *consumer* could be an application monitoring data on a building; it would obtain the temperature from room A, subscribing to this topic.

The broker stores specific filters within the *consumer* session. It also establishes data routes that match updates from publishers to all consumers that are subscribed to a specific topic. MQTT allows for point-to-point and 1-to-many data exchange patterns. Therefore, MQTT can support interest- and event-driven communication within industrial environments. However, the communication is not bidirectional.

The key advantages of MQTT in the context of industrial environments can be summarized as follows [11].

- Lightweight, asynchronous communication. MQTT is data-driven and has a substantially small packet header, thus allowing data exchange across embedded devices with a remarkably small footprint.

- QoS and fault tolerance. MQTT integrates three QoS levels (0, 1, and 2) to ensure the reliability of message delivery. Data persistency (the registered data can be obtained even if a consumer disconnects and reconnects).

- Reduced bandwidth uses in comparison to request–response protocols, such as CoAP, claiming 80% improvement.

A. MQTT Sparkplug

A missing aspect in MQTT is a uniform namespace that can be applied in the context of manufacturing environments, considering the need to integrate brownfield devices into end-to-end, sophisticated IoT systems. Therefore, adapting the namespace to the specific use case is necessary for MQTT (which implies not only a description of the specific machines). It also relates to providing details on the data routes and topics that can be subscribed (how to handle the payload, how QoS levels are mapped).

These aspects are currently supported by Sparkplug B, which specifically addresses the particularity of industrial domains [11]. Therefore, the current specification of Sparkplug defines two specific node entities:

- **Edge of Network nodes (EoN).** These nodes correspond to gateways that support the interconnection of legacy devices. EoN also comprises smart devices and sensors that already published Sparkplug B data or metrics to a broker.

- **Application nodes.** These nodes correspond to software-based clients (consumers) or a legacy gateway.

In the context of the MQTT Sparkplug specification, all MQTT clients and brokers should be compliant with the latest MQTT V3.1.1 specification.

1) Sparkplug™ MQTT Topic Namespace

MQTT Sparkplug nodes rely on the Sparkplug MQTT Topic Namespace structure illustrated in Figure 2 and hold the following fields.

- Namespace refers to the Sparkplug version used. Therefore, it defines the remaining elements of the namespace as well as the payload. Two options are possible: either (i) "spAv1.0" referring to Sparkplug™ payload definition A or (ii) "spBv1.0" referring to the Sparkplug™ payload definition B.
- Group ID (group_id) is a logical identifier for a group of MQTT nodes, as defined by the user.
- Message Type (message_type) indicates how the payload is handled. The payload may contain the following aspects: polled data or a specific command; whether it belongs to a node or device; primary application.
- Edge Node ID (edge_node_id) identifies a specific MQTT EoN. The Group ID/Edge Node must be unique.
- Device ID (device_id) is an optional element that identifies the (logical or physical) device attached to the EoN.

2) Sparkplug B MQTT Payload

As one of the key strengths in industrial environments, Sparkplug B defines a specific structured and flexible payload format. The payload is based on key-value pairs and associated meta-data. Additional optional fields can be included (e.g., name, description). The payload can also integrate specific arrays with custom properties (e.g., units, published along with a process or a data object). The payload is encoded via Protobufs, thus providing an efficient binary representation, which becomes highly effective in industrial environments.

3) State Management

Sparkplug follows on the MQTT "last will" and "testament" messages to support data permanence, thus relying on "birth and death" certificate messages. These messages are used upon expiration of a consumer keepalive timer. A new aspect introduced in Sparkplug is the birth certificate, a message that a consumer can utilize to publish data for itself and each of its devices. This approach provides a natural method for data and topic discovery.

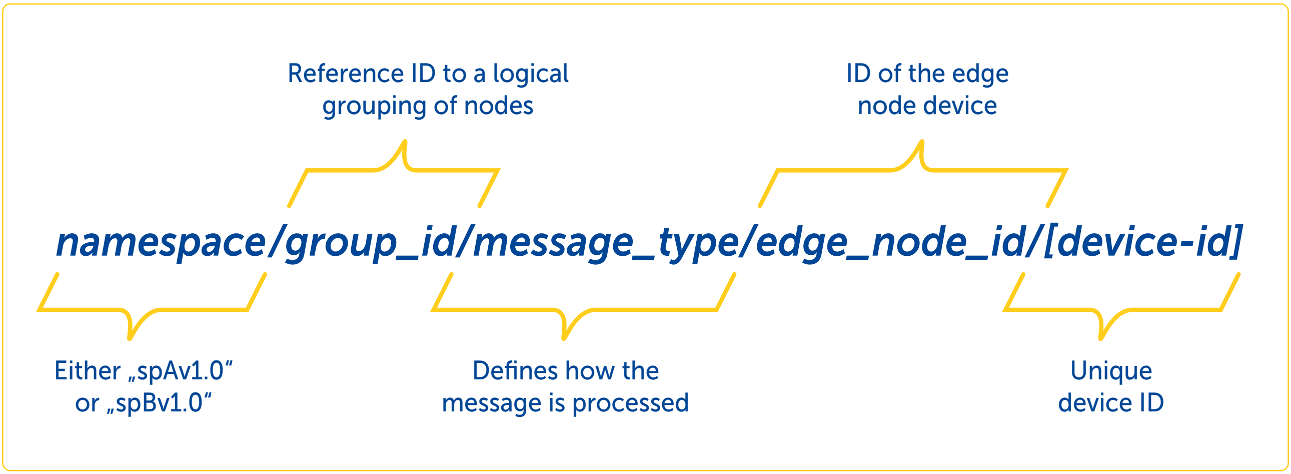

Reference ID to a logical grouping of nodes
ID of the edge node device
namespace/group_id/message_type/edge_node_id/[device-id]
Either „spAv1.0" or „spBv1.0"
Defines how the message is processed
Unique device ID

Figure 2: Sparkplug Topic Namespace structure.

MQTT Sparkplug in EFPF

The EFPF platform uses the MQTT Sparkplug to structure its topic namespace and provide some semantics to the MQTT topics. This condition is especially important in a federated platform where data are exchanged across platforms, thus utilizing a standard (such as MQTT Sparkplug) to effectively understand the type of data using the topic namespace and therefore improves interoperability.

In the context of EFPF, data are exchanged between shop floor and services from different platforms using EFPF IoT gateways and factory connectors, as illustrated in Figure 3. In this scenario, the shop floor data are published by the EFPF IoT gateways and factory connectors to the EFPF Message Bus, wherein the *edge_node_id* is the unique ID of the EFPF IoT gateway or factory connector, the *message_type* defines the type of message to be processed (e.g., "NDATA" (*Node data message*))," the *group_id* refers to the logical grouping of nodes (e.g., *plant1*), and the namespace is "spBv1.0" because EFPF use Sparkplug payload definition B. For a third-party service to consume the shop floor data in the required specific data model, a developer consumes the data via the data flow management, and service composition transforms these data to the required model and republishes them to the Message Bus. The topic namespace must be different in this step. However, understanding the origin of the data and its transformation is necessary.

Data exchange can be achieved in various ways. One approach is to adjust the *edge_node_id* to reflect the node where the data originates, namely data flow management and service composition. This approach carries two challenges: ensuring the uniqueness of the ID and losing the information regarding the specific EFPF IoT gateway or EFPF factory connector, which is initially published to the Data Spine.

Another approach would be to add a new *message_type*; however, such an approach would imply changes to the current specification.

The final approach, which is adopted by EFPF, is to adjust the *group_id* (for instance, by changing it from "*plant1*" to "*plant1_transformed*").

MQTT is also used in some EFPF scenarios to facilitate data exchange between services from different platforms, as illustrated in Figure 4. Such scenarios relate to exchanging industrial time series data (e.g., between a predictive maintenance service and a visualization or a data storage service). Therefore, reconsidering the use of a topic namespace in such scenarios is necessary.

The approach adopted by EFPF based on an example of a predictive maintenance application is presented as follows.

- If required, then use *device_id* to refer to a specific device (for instance, the id of the machine).
- Apply *edge_node_id* as the unique ID of a service instance that publishes data.
- Apply *message_type* to the type of message to be processed (e.g., "NDATA").
- Apply *group_id* to refer to the type of published data. For a specific example, the group_id could be set to "*predictive_maintenance*" if the published data relate to predictive parameters or could be set to "*Welding_quality*" if the data provide welding quality parameters of a specific machine.
- If data transformation occurs (if the data are transformed using the EFPF data flow management and service composition), then *group_id* is adjusted to reflect the transformation (for example, from "*predictive_maintenance*" to "*predictive_maintenance_transformed*").

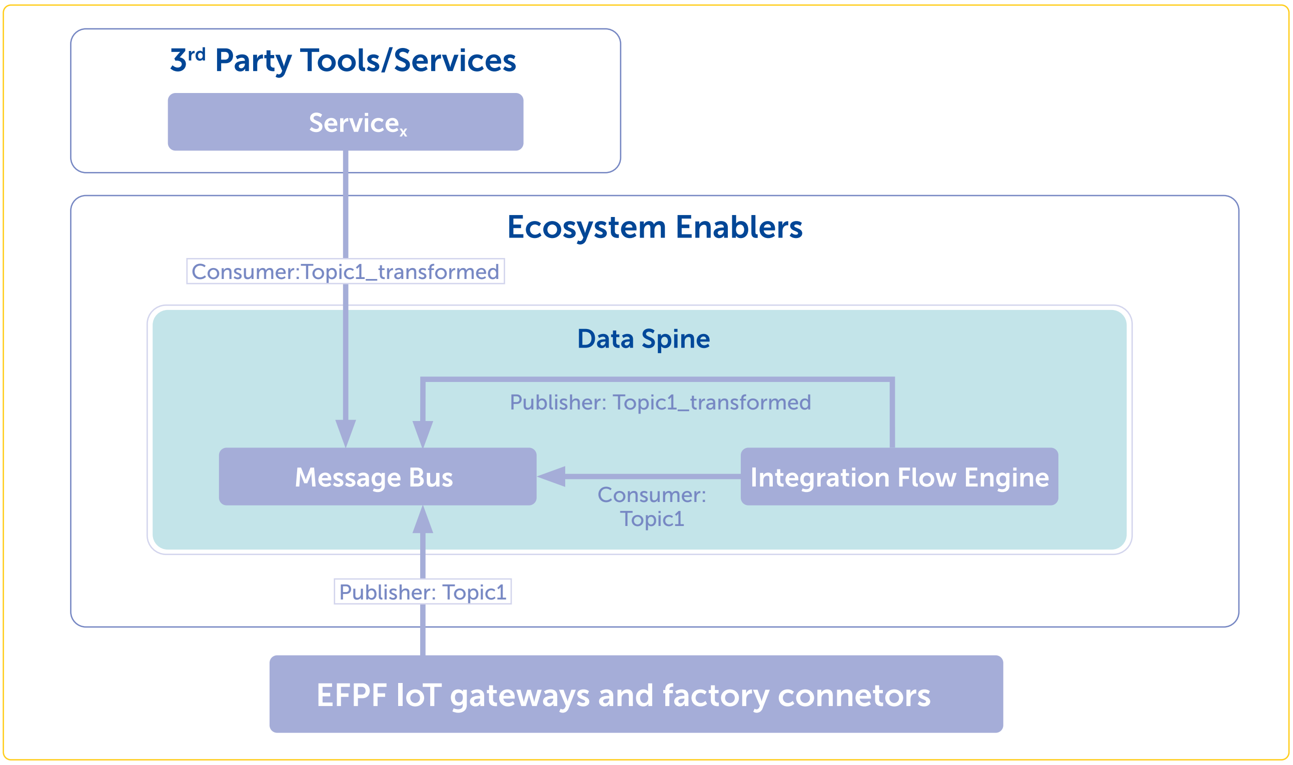


Figure 3: Data exchange between shop floor and services.

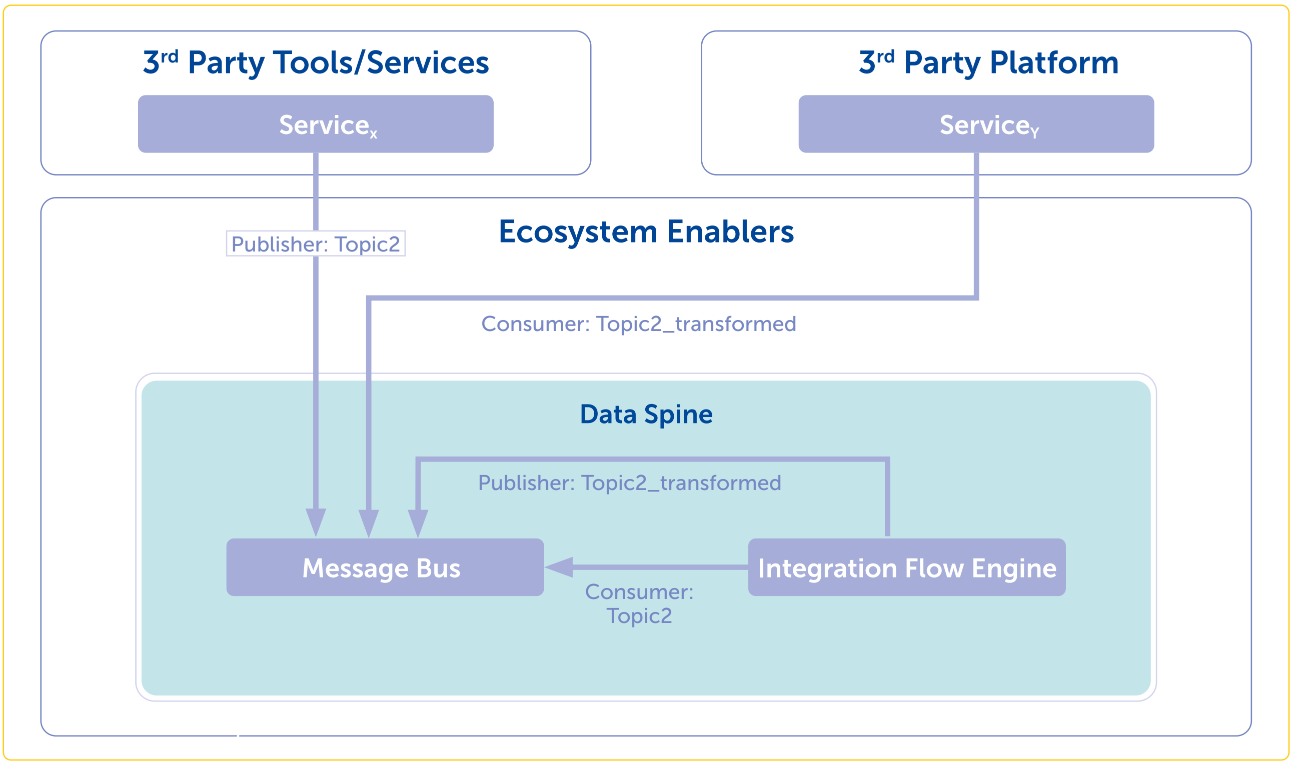


Figure 4: Data exchange between services across platforms.

EFPF MQTT Sparkplug Examples

This Section describes the application of MQTT Sparkplug into two different services provided in the EFPF platform: a data analytics tool and the use of MQTT Sparkplug to support data exchange across a shop floor by third parties.

A. Data Analytics Tooling Interconnection

MQTT Sparkplug has been used to allow third-parties, such as a systems integrator, to provide an interconnection between a shop-floor and the different EFPF data analytics tools[13].

The EFPF data analytics tools can be used to generate insights regarding the shop floor processes. The implemented data flow scheme is illustrated in **Error! Reference source not found.**. The data source in this example is an EFPF base component, namely a factory connector, which is connected to multiple sensors attached to an edge banding machine on the shop floor.

This use case has some challenges derived from the integration across an IIoT system. One challenge is the capability to push the data from a single *Publisher* (in this case, the factory connector) to different tools, which will perform various operations using the EFPF platform. Another challenge, created by the necessity to use different tools, is that each tool often has different requirements on the data model for the input data. Some tools may require the use of proprietary data models, while others will be based on standard data models. Other tools will still require some scaling operations on the input data for processing (e.g., converting temperature values from the Fahrenheit to the Celsius scale).

The EFPF platform contains all the resources required to enable the workflow illustrated in **Error! Reference source not found.**.

- The MQTT/AMQP broker integrated into the **Message Bus** component allows the interconnection of different components of the workflow asynchronously.
- The **PubSub Security Service** provides the user interface for creating the secured private MQTT/AMQP topics, on which the tools can exchange the data.
- The **Integration Flow Engine** component includes some useful integrated processors to interconnect the tools and to perform the required transformation operations on the exchanged data.

The first step for connecting a factory connector to the Data Spine is its registration as a resource on the PubSub Security Service. The topic on which it will publish the data can be created after its registration. Additional topics for a single resource can be registered when necessary. The resources and the topics are thus private to the company that created them; however, for privacy reasons, both can be shared only with users of the platform on demand. This step has been implemented using virtual hosts, and each registered company on the platform has its virtual host created on the Message Bus. One disadvantage of this implementation choice is the mitigated risk of creating equal topics by different users because different companies could register the same topic on various virtual hosts without data collision.

The factory connector can start to send data on the Message Bus continuously using the MQTT protocol once it is connected to the EFPF Message Bus on the Data Spine. Regarding the topic namespace, the following aspects in this example scenario have been integrated:

- Edge node ID (*edge_node_id*) has been defined as "IW2001_LAG."
- Message type (*message_type*) has been set as "NDATA."
- Group ID (*group_id*) has been set as "EDGE_BANDING_CONDITION" based on the assumption of the interconnection between the factory connector collecting sensor readings of the edge banding and the EFPF Ni-Fi integration flow engine.

Data must be transformed before their use by the EFPF data analytics tools once they reach the Data Spine. Therefore, multiple processors are used inside and outside the integration flow engine of EFPF (EFPF Ni-Fi)[14]. These transformations are necessary because different data analytics tools may rely on various data models.

Figure 5 illustrates two different data integration flows. From steps 2 to 5, data are collected from the Message Bus to facilitate scaling to standard units of measurement using a JOLT transformation processor and then published again for consumption by a data analytics tool.

The second integration flow covers steps 6 to 11, wherein data model transformation operation on the obtained data from the factory connector is performed by pushing data to an external data model translation microservice. This microservice processes the data and sends them back to the EFPF Data Spine, where they are pushed back to the Message Bus, thus readily becoming available for consumption by another tool.

The following aspects have been integrated to incorporate the described changes to the data performed on the integration flow engine.

- *group_id* is changed to "EDGE_BANDING_CONDITION_SCALED" for the topic used between the EFPF Ni-Fi integration flow engine and the data analytics tools, which can consume data encoded with the original data model. However, scaling operations must be performed on the data values.
- *group_id* is changed to "EDGE_BANDING_CONDITION_SCALED_TRANSFORMED" for the topic used between the EFPF Ni-Fi integration flow engine and the data analytics tool, which must consume data encoded with a different data model and requires scaling operations on the data values.

Each of the aforementioned publishing operations always requires registration of the resource, that is, publishing the data and the topic on which the data are published, as previously described. Keeping track of a complex workflow, such as the generic one described above, is easy for system integrators by using the proposed extension of the Sparkplug MQTT namespace. As described earlier, monitoring the transformations performed on the data and the data exchange on the topic is possible only by examining the available topic list on the PubSub Security Service.

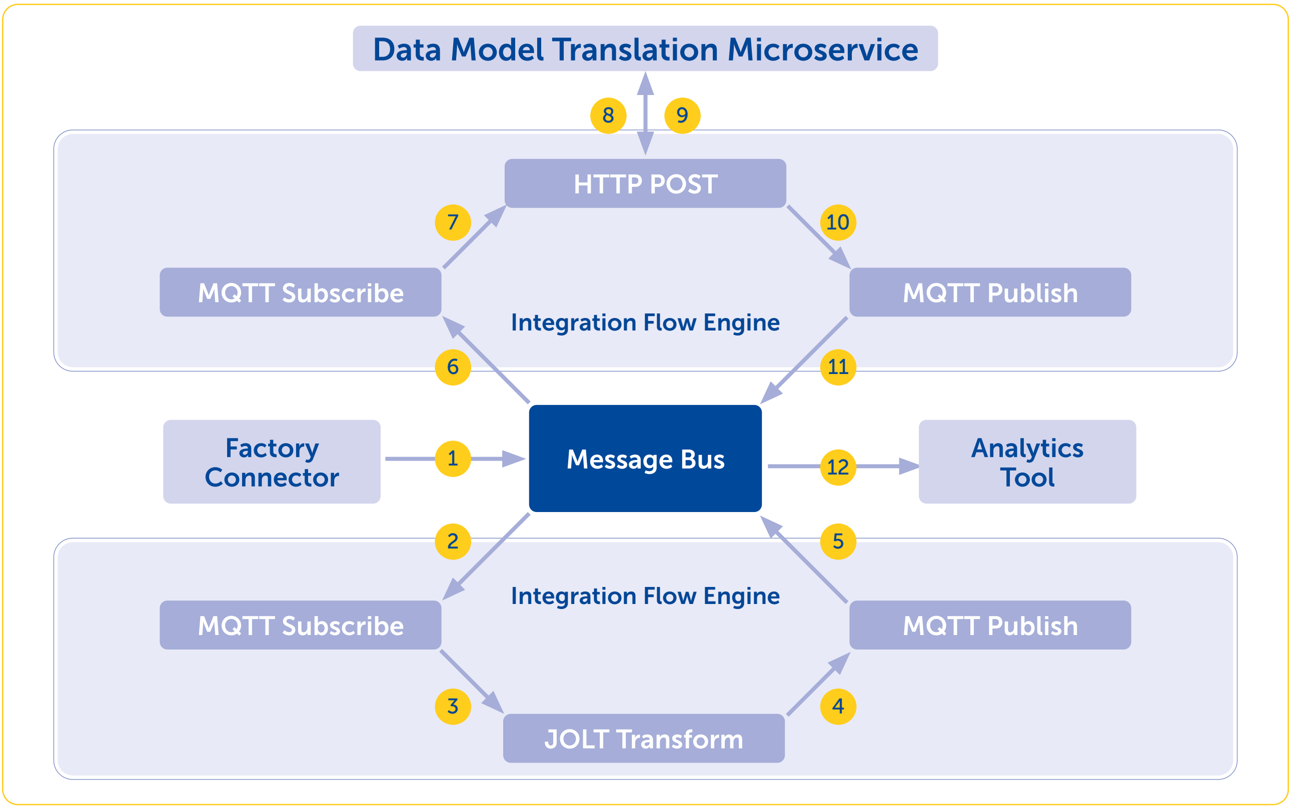


Figure 5: Data flow diagram.

13 https://www.efpf.org/data-analytics
14 https://docs.efpf.linksmart.eu/projects/data-spine-nifi/

B. Environment Monitoring Pilot

On a second pilot, MQTT Sparkplug has been used to support an IIoT monitoring use case; that is, monitoring environmental aspects on the shop floor, temperature, and humidity. The EFPF pilot involved two SMEs: Walter Otto Müller GmbH & Co.KG[15] and Innovint Aircraft Interior GmbH[16].

Figure 6 illustrates the data workflow in this pilot, focusing on the integration of MQTT Sparkplug to facilitate data exchange between two components of the EFPF platform: *the Thing to Service Matching (TSMatch)*[17] and the IoT Symphony platform[18].

TSMatch provides support for semantic matchmaking between sensor and machine descriptions and services [12]. The IoT Symphony platform is a complete BMS platform whose basic building blocks can be used to support the automation of different production sites.

Two topics were used to support the interconnection between TSMatch and the IoT Symphony platform. The first topic was used to send sensor observations from the TSMatch engine (server-side) to the EFPF Integration Flow Engine, wherein the data are transformed to the data model required by the IoT Symphony platform using JOLT transformation. The second topic was used to share the transformed sensor observations with the Symphony platform. A service registry is utilized to register the service descriptions using AsynAPI 2.0 specs to ease integration.

The following aspects considering the topic namespace have been integrated:

- *edge_node_id* has been defined as "TSMatch_WOM_1";
- *message_type* has been set as "NDATA";
- *group_id* has been set as "WOM_OBSERVATION," assuming the interconnection between TSMatch and the EFPF Ni-Fi Integration Flow Engine;
- *group_id* is changed to "WOM_OBSERVATION_TRANSFORMED" for the topic used between the EFPF Ni-Fi Integration Flow Engine and the Symphony platform.

Easily integrating different data sources (sensors) is feasible with the proposed data workflow and MQTT Sparkplug namespace configuration, thus allowing for simplified data processing on the shop floor.

15 https://www.wom.gmbh/}
16 https://www.innovint.de/}.
17 https://docs.efpf.linksmart.eu/projects/factory-connectivity-smart-factory-tools/ds-tsmatch-gateway/
18 https://docs.efpf.linksmart.eu/projects/factory-connectivity-smart-factory-tools/ds-symphony-platform/

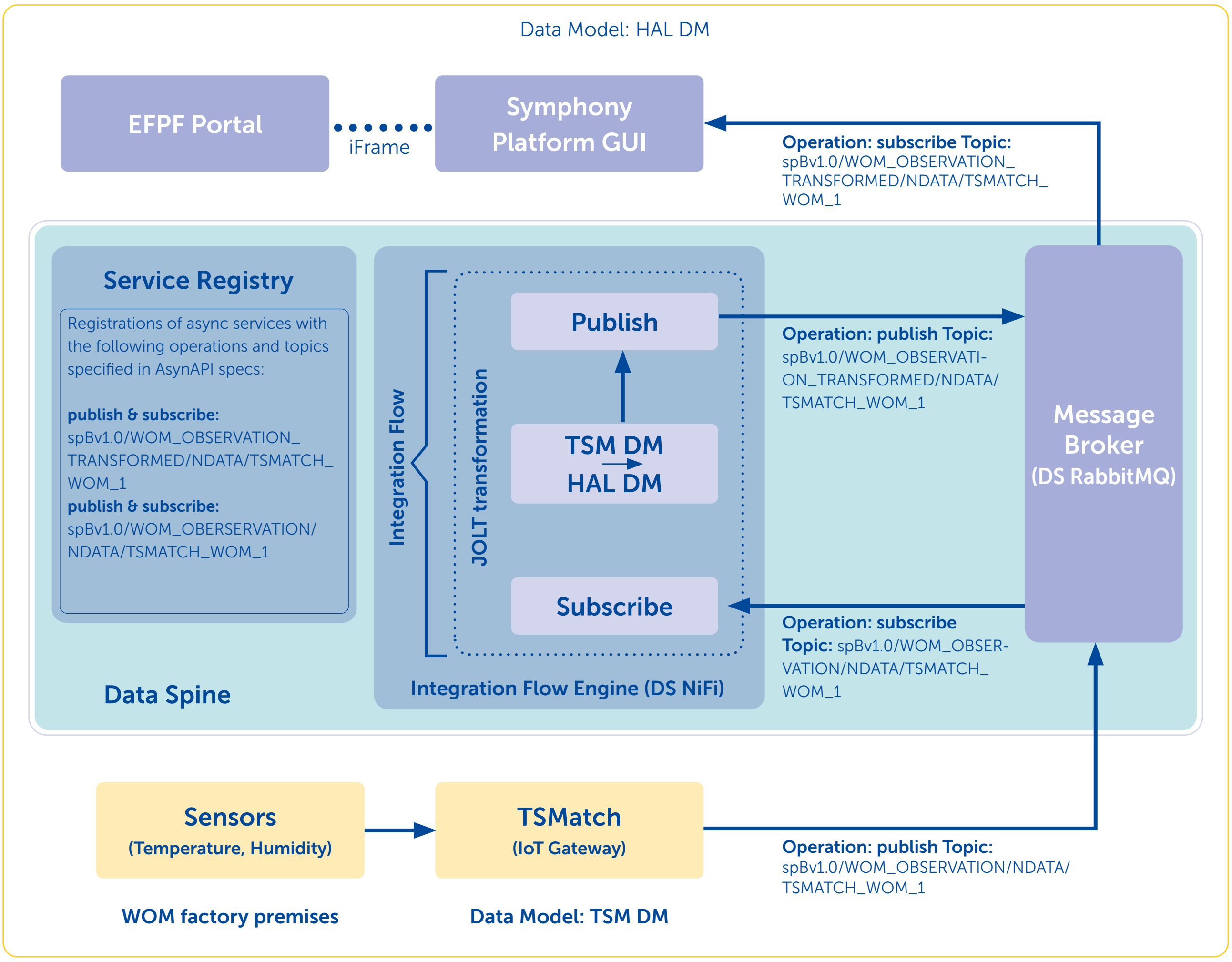

Data Model: HAL DM
EFPF Portal
iFrame
Symphony Platform GUI
Operation: subscribe Topic: spBv1.0/WOM_OBSERVATION_TRANSFORMED/NDATA/TSMATCH_WOM_1
Service Registry
Registrations of async services with the following operations and topics specified in AsynAPI specs:
publish & subscribe: spBv1.0/WOM_OBSERVATION_TRANSFORMED/NDATA/TSMATCH_WOM_1
publish & subscribe: spBv1.0/WOM_OBERSERVATION/NDATA/TSMATCH_WOM_1
Integration Flow
JOLT transformation
Publish
TSM DM
HAL DM
Subscribe
Operation: publish Topic: spBv1.0/WOM_OBSERVATION_TRANSFORMED/NDATA/TSMATCH_WOM_1
Operation: subscribe Topic: spBv1.0/WOM_OBSERVATION/NDATA/TSMATCH_WOM_1
Message Broker (DS RabbitMQ)
Data Spine
Integration Flow Engine (DS NiFi)
Sensors (Temperature, Humidity)
TSMatch (IoT Gateway)
Operation: publish Topic: spBv1.0/WOM_OBSERVATION/NDATA/TSMATCH_WOM_1
WOM factory premises
Data Model: TSM DM

Figure 6: Data flow diagram of the environment monitoring pilot scenario.

VII.

Summary and Lessons Learned

This white paper provides an overview of the application of MQTT Sparkplug in the context of different EFPF pilots to assist in reducing data workflow complexity and improve interoperability.

A key challenge detected during the development of the EFPF platform relates to the different data models applied in manufacturing. Another key challenge is the need to simplify the overall data process on the shop floor, minimizing the need for advanced expertise in IIoT and assisting manufacturers in a broad integration of services.

The EFPF PubSub security service, which is based on DS RabittMQ, facilitates the interconnection of current and future data processing services in EFPF via the creation of topics, on which EFPF services (e.g., Factory Connector and data analytics Tool) publish data. The use of MQTT Sparkplug in this context enables the interconnection of different EFPF services by providing third parties within a common interface.

Advantages derived from the adoption of the MQTT Sparkplug namespace lie in the possible implementation of advanced topic searching functions on the platform user interface, such as grouping by edge node or searching for all the transformations applied to a particular data source.

VIII.

Acknowledgements

The work detailed in this white paper has been supported by the H2020 EFPF project (2018–2022), grant agreement number 825075.

References

[1] S. H. Leitner and W. Mahnke, "OPC-UA Service-oriented architecture for industrial applications," 2006.

[2] S. Raff, "The MQTT community." [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki. [Accessed: 10-Jun-2018].

[3] M. S. Rocha, G. S. Sestito, A. L. Dias, A. C. Turcato, D. Brandão, and P. Ferrari, "On the performance of OPC UA and MQTT for data exchange between industrial plants and cloud servers," *ACTA IMEKO*, vol. 8, no. 2, pp. 80–87, 2019.

[4] D. Silva, L. I. Carvalho, J. Soares, and R. C. Sofia, "A Performance analysis of internet of things networking protocols: Evaluating MQTT, CoAP, OPC UA," *Applied Sciences*, vol. 11, no. 11, p. 4879, 2021.

[5] R. C. Sofia and J. Soldatos, "A vision on smart, decentralised edge computing research directions," *NGIoT White Pap.*, no. September, 2021.

[6] S. R. J. Ramson, S. Vishnu, and M. Shanmugam, "Applications of internet of things (iot)--An overview," in *2020 5th international conference on devices, Circuits and Systems (ICDCS)*, pp. 92–95, 2020.

[7] R. C. Sofia, M. Kovatsch, and P. Mendes, "Requirements for reliable wireless industrial services," Internet Engineering Task Force, December. 2021.

[8] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Systems Engineering Symposium* (ISSE), pp. 1–7, 2017.

[9] I. V. Nițulescu and A. Korodi, "Supervisory control and data acquisition approach in node-RED: Application and discussions," *IoT*, vol. 1, no. 1, pp. 76–91, 2020.

[10] A. Nipper, "How to build scalable data models with MQTT Sparkplug: A key to bridging the OT/IT gap is enabling successful data modeling, which is how organizations define and organize their business processes.," *Plant Engineering*, vol. 75, no. 4, pp. 21–23, 2021.

[11] Opto, "Industrial strength MQTT/Sparkplug B, building industrial MQTT networks at scale with edge computing," Opto vol. 22, p. 22, 2021.

[12] N. Bnouhanna, E. Karabulut, R. C. Sofia, E. E. Seder, G. Scivoletto, and G. Insolvibile, "An evaluation of a semantic thing to service matching approach in industrial IoT environments," *inProc IEEE Percom IoT-Producao 2022 Workshop*, 2022.

Imprint

Publisher
fortiss GmbH
Guerickestraße 25
80805 München

Editing
enago, München

Layout
fortiss Marketing

Print
viaprinto GmbH & Co. KG

ISSN Print
2699-1217

ISSN Online
2700-2977

1st issue:
December 2022

Photo credits:
Title: AdobeStock 248781604
Page 18: Fortiss GmbH @ Kathrin Kahle

Find here more
fortiss White Paper

fortiss is the Free State of Bavaria research institute for software-intensive systems based in Munich. The institute collaborates on research, development and transfer projects together with universities and technology companies in Bavaria and other parts of Germany, as well as across Europe. The research activities focus on state-of-the-art methods, tech-niques and tools used in Software & Systems-, AI- and IoT-Engineering and their application with cognitive cyber-physical systems.

fortiss is legally structured as a non-profit limited liability company (GmbH). The shareholders are the Free State of Bavaria (majority shareholder) and the Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.

fortiss GmbH
Guerickestraße 25
80805 München
Deutschland
www.fortiss.org
Tel: +49 89 3603522 0
E-Mail: info@fortiss.org

fortiss