Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Comparison of Communication Technologies for Industrial Middlewares and DDS-based Realization ⋆

Emanuel Trunzer, **Thomas Schilling**, **Micha Müller**, **Birgit Vogel-Heuser**

Institute of Automation and Information Systems, Technical University of Munich, Garching, Germany (e-mail: {emanuel.trunzer, thomas.schilling, micha.mueller, vogel-heuser}@tum.de).

Abstract: The automation industry is currently in the process of transforming towards Industrie 4.0. To fully leverage the new possibilities, e.g., predictive maintenance, systems from all levels of the AT-pyramid are required to be fully connected. For this purpose there exist abstract models as well as concrete implementations of middleware-based interconnection platforms.
This paper aims to close the gap between abstract concepts and concrete implementations by comparing technological concepts for a middleware implementation. At first, requirements are determined, which should be fulfilled by middleware to be comprehensible and competitive. Based on these requirements, six different middleware technologies are presented and evaluated. The evaluation yields that both OPC UA Pub/Sub with AMQP/MQTT as transport and DDS suit the requirements best. However, there are no complete implementations of OPC UA Pub/Sub with AMQP/MQTT as transport available yet. Therefore DDS is chosen for a prototypical middleware implementation. The prototype is tested and shows to fulfill all except one requirement, namely real-time constraints.
In conclusion, it is recorded that for future middleware implementations, OPC UA Pub/Sub (as soon as complete implementations are available) and DDS are the most appropriate technologies.

Keywords: Horizontal and Vertical Integration, Industrial communication protocols, Middleware Integration and Communication, Systems Interoperability

1. INTRODUCTION

In today's industry, the term Industrie 4.0 gets more and more tangible as an ever-increasing amount of sensors are recording our physical world. With this rise comes a surge of raw data. The industry has a significant interest in collecting and analyzing this data for further refinement. Nevertheless, transparent access and availability of data is still an issue as it is distributed over various systems. Further on, many of these systems bear additional challenges which prevent operators from effectively collecting data, e.g.

- outdated software (Bisbal et al. (1999); Crotty and Horrocks (2017)),
- little to no documentation (Crotty and Horrocks (2017)),
- proprietary (fieldbus-)systems (Sauter (2010)), and
- system constructed according to the classical automation pyramid (IEC (2013)), which does not allow out-of-band communication (Sauter (2010)).

Implementing a data collection service that makes all data available on top of such heterogeneous legacy environment requires many point to point connections. Each of these connections has to be implemented with a dedicated pair of adapters. This results in a complex connection mesh. Maintenance of such a mesh as well as integrating new systems is a cost-intensive task.

Trunzer et al. (2019) proposed a generic system architecture for Industrie 4.0 applications. The central part of the concept design is a middleware-based interconnection platform. In this setup, only one connection to the middleware has to be set up and maintained for each system instead of a communication mesh. New systems can be integrated natively into the middleware communication, while legacy systems can be attached via adapters. Finally, the concept also allows for easy integration of additional services like data storage to collect all and provide historical data. The architecture concept is technology independent and can be implemented using various sets of technology (see Trunzer et al. (2018) for an example). For instance, different middleware solutions can be used to mediate the connected systems. State-of-the-art approaches (see Section 3) focus on the presentation of abstract reference architectures or concrete realizations but do not consider the intermediate step of choosing a suitable middleware solution.

⋆ This research is part of the project "M@OK" (machine@onlineknowledge), which has received funding by the Bavarian Ministry of Economic Affairs, Energy and Technology (StMWi) under grant number IUK566/001.

The contribution of this paper is, therefore, the comparison of different technological concepts for the realization of industrial middleware concepts as proposed by Trunzer et al. (2019). Based on the technology comparison, a prototype is subsequently realized and evaluated against industrial requirements. In comparison to the state-of-the-art, the authors investigate the step of surveying available technological concepts and evaluating the suitability for an industrial middleware realization.

The remainder of the paper is as follows: Section 2 presents requirements an industrial middleware should fulfill. Section 3 gives an overview of the current state of the Art concepts. In Section 4 different technological concepts are presented and evaluated for their fulfillment of the requirements. The implementation of a prototype is described in Section 5. Section 6 gives results of the prototype evaluation. Finally, Section 7 concludes the paper.

2. MIDDLEWARE REQUIREMENTS

For an objective comparison and evaluation of middleware technology concepts, a set of preferably standardized requirements has to be determined beforehand. With VDI (2013), the German VDI and VDE associations presented a guideline defining the requirements for middleware in industrial automation. The VDI and VDE are both associations of leading German engineers, scientists, and enterprises, which are also promoting the German Industrie 4.0 initiative. At the same time, they are the primary target group for the proposed solution. Hence their stated requirements can be considered as a reference. Still, a comparison based on all requirements would be too exhaustive. Therefore, five characteristic requirements (R1-R5) are selected, which the authors consider to make up the core of a middleware, which are explained in the following. At the end of this chapter, an additional, non-functional requirement is derived.

The Industrie 4.0 initiative triggered a steady rise of *smart* communicating systems. Therefore, an industrial middleware technology should be capable of handling a variable number of participating systems without the need to adapt to the underlying architecture. This ability, called *horizontal scalability* (R1), is essential for middleware to be future-proof.

It is of great importance that a concept does not have to be implemented from scratch, if possible. *Adaption of available* (R2) solutions lowers development effort and shortens time-to-market. Therefore, available technologies are preferred over theoretical designs.

The generic middleware concept, as introduced by Trunzer et al. (2019), includes a real-time communication bus. Therefore, the real-time capability is a metric to consider as a requirement. Real-time can be divided into three subcategories by considering the classification of deadlines, hard, firm, and soft (Shin and Ramanathan (1994)). The first indicates severe consequences if a deadline is missed. Missing firm deadlines does not cause severe consequences, but the result produced does lose its value. Frequent misses of a soft deadline will degrade the overall utility of a result produced. For an industrial middleware, soft real-time is sufficient if no control interactions should be carried out over the middleware. However, missing data for a longer time frame may lead to false and/or delayed outcomes. Consequently, *soft real-time capability* (R3) is considered a requirement for middleware technology.

Another property the communication should fulfill, aside from latencies, is that no messages are lost. Furthermore, no messages should be received twice and perhaps be interpreted as two individual messages. Each message should be guaranteed to be received only once by each querying receiver. This property is referred to in the following by the term *Quality-of-Service (QoS)* (R4).

When deploying an implemented design, the *configurability* (R5) has to be considered. Even a well-functioning middleware can be considered suboptimal if a large amount of man-hours is required to configure the system. Also, adding new participants should be viable with a reasonable small amount of effort. Especially in a continually changing environment and to accommodate possible failures, fast and easy (re-)configurability is desirable.

For optimal middleware implementation, the selection of a technological concept should be as transparent and comprehensible as possible. Hence, the selection of a technological concept for the middleware should not be an arbitrary choice. A middleware based on a random technological concept may suffer from disadvantages which origin lies within the technology and cannot be resolved by implementation means. The selected design should thus be based on a comparison of multiple state of the art technology concepts (R0).

3. EXISTING MIDDLEWARE REALIZATIONS

There are already several approaches that investigate the same or similar problem statements.

The PERFoRM project is dedicated to solving the ever-increasing demand for flexibility and reconfigurability in the industry. The approach taken by the PERFoRM project members is similar to the one used in this contribution. They compare an extensive list of Enterprise Service Bus (ESB)-based implementations (Gosewehr et al. (2016)), but no other middleware concepts besides ESBs. Furthermore, a system architecture and an exemplary implementation are presented (Angione et al. (2017)).

The Line Information System Architecture (LISA), as described by Theorin et al. (2015), is designed to enable utilization of data while maintaining easy factory integration. The main goal of LISA is to be industrially applicable. Therefore, LISA is based on international standards and established off-the-shelf technologies. However, the authors present the implemented design without a preliminary comparison of other concepts or implementation possibilities.

Trunzer et al. (2018) describe a technological concept for a Unified Data Transfer Architecture in automated production systems based on their generic architecture presented in Trunzer et al. (2019). Their developed architecture aims to enable the handling of big data from heterogeneous sources. The authors analyze the requirements for such an architecture and compare different concepts. As none of the considered concepts fulfill all of their requirements,

the authors develop their architecture. Nevertheless, the reasoning for the implementation technology for the architecture is not mentioned.

Based on OPC UA aggregate servers, Großmann et al. (2014) developed a solution for the problem of point to point connection meshes. The concept replaces point to point OPC UA connections with an OPC UA aggregate server, which acts as a central integration point. A working prototype is implemented. However, there is no reason why an OPC UA aggregate server is chosen over other concepts.

Some of the presented works fulfill all five functional requirements (R1-R5). However, none of the approaches includes a comparison of suitable technological concepts for the implementation of the middleware (R0).

4. COMPARISON AND EVALUATION OF MIDDLEWARE TECHNOLOGIES

Addressing R0, relevant technology concepts for an industrial middleware are compared in the following. The presented technologies are based on well known (industrial) specifications and are already present in the literature. Therefore, they are expected to be highly relevant to the realization of industrial middleware. Overall, six different technology concept ideas are compared.

The first is based upon OPC UA (OPC Unified Architecture, OPC (2017)). Data producers run an OPC UA server to which the data consumers, OPC UA clients, can connect to (Schleipen et al. (2016), see Figure 1 a). There are multiple free and commercial OPA UA implementations available [1, 2, 3], which fulfills (R2). These provide communication over a TCP Transport Layer. TCP only ensures messages are received at least once, which partially satisfies (R4). Although OPC UA implementations cannot satisfy hard real-time constraints, they are still capable of fulfilling soft real-time constraints (R3). OPC UA itself is horizontally scalable (R1), however the point-to-point nature of server/client communication results in large connection meshes. A client may not only gather information from one data provider instance but may want to query multiple data producers. This will result in a high configuration complexity as setting up connections to the servers is left to the client (R5), and the provided discovery mechanisms are only informational.

The configuration complexity is a relative disadvantage of the previous technological concept. Adding an centralized aggregation instance can lower the configuration efforts (Großmann et al. (2014), see Figure 1 d). Data of several OPC UA servers get aggregated and is offered at the central aggregation server. Hence, each system needs only one connection to the aggregate server. Therefore, the requirement of low configuration effort (R5) is met. As this concept is based on OPC UA, requirements R2 and R4 are still fulfilled. The disadvantage of this solution is that by adding an aggregation layer, time constraints that were previously met may no longer be satisfied (R4). This problem intensifies if more aggregation layers are added. The major disadvantage of this concept is the inability of horizontal scaling (R1). One would have to partition the network into multiple parts to deal with an increasing number of aggregation servers.

At the beginning of 2018, the 14$^{\text{th}}$ part of the OPC UA specification (OPC (2018b)) was released, which extends the OPC UA standard by a Publish/Subscribe (Pub/Sub) communication model. The specification includes mappings to the two standardized protocols AMQP (AMQ (2014)) and MQTT (MQT (2016)), which allow a broker-based message distribution for OPC UA (see Figure 1 c). Part 14 also defines the customized, UDP-based UADP protocol (OPC (2017)). Pub/Sub enables a good horizontal scalability (R1). Moreover, the use of MQTT/AMQP broker technologies abstracts the need to manually identify and connect to endpoints, resulting in a low configuration effort (R5). R4 can be fulfilled by using AMQP or MQTT as the transport layer. Unfortunately, no feature-complete implementations are available yet (R2). Therefore, it can not be determined whether OPC UA Pub/Sub will be (soft) real-time capable or not (R3). Nevertheless, partial implementations are already available [4]. None of them supports AMQP/MQTT, but instead use UADP communication which is depicted in Figure 1 b). This provides almost all of the advantages of OPC UA Pub/Sub with AMQP/MQTT mentioned before, but with the drawback that UADP offers no QoS support (R4).

Another approach to realize the Pub/Sub communication model similar to Figure 1 c) is a custom queue-based message broker. Here, protocols like AMQP or MQTT are used. Participants are managed by a custom administration system for communication and service registry. Available message-brokers like RabbitMQ [5], Apache ActiveMQ [6] or Apache Kafka [7] can serve as a basis here. As the custom queue-based message broker realizes the Pub/Sub model, horizontal scalability is given (R1) as well as QoS when using an appropriate protocol (R4). Assessing fulfillment of (R2) depends on the selection of technologies for implementation. For example, the use of RabbitMQ as a message broker allows for easy deployment without much implementation effort. In comparison, the utilization of Kafka requires a moderate to high amount of implementation work. The same reasoning can be made for soft real-time capability (R3) and configurability (R5).

A further technology concept to consider is the usage of DDS (Data Distribution Service, DDS (2015); García-Valls et al. (2018)). Establishing connections between different participants is handled by the virtual DDS Domain and the DDS Data Space. Both systems are implemented in the participants themselves. Hence, the message exchange is decentralized and distributed using the Pub/Sub paradigm (see Figure 1 e). This allows for reasonable horizontal scalability (R1). The Object Management Group (OMG) specification for DDS dictates a minimum of QoS (R4) and can satisfy soft real-time deadlines (R3). There are already various, mostly commercial but also free, implementations available (R2), which include all features according to the latest specification revision. A minor disadvantage is the

[1] `https://open62541.org`, accessed 29.08.2019
[2] `https://github.com/opcfoundation`, accessed 29.08.2019
[3] `https://www.beckhoff.de/english/twincat/tf6100.htm`, accessed 29.08.2019
[4] See footnotes 1 and 3
[5] `https://www.rabbitmq.com/`, accessed 18.07.2019
[6] `http://activemq.apache.org/`, accessed 18.07.2019
[7] `https://kafka.apache.org/`, accessed 18.07.2019

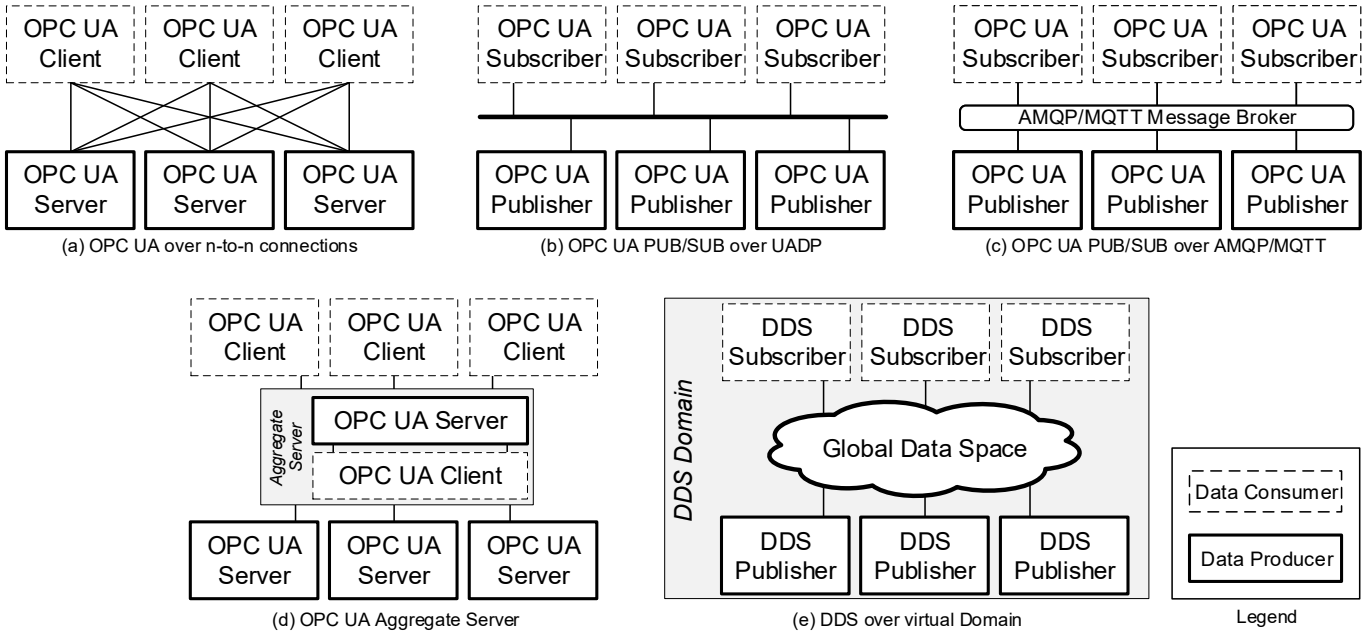

Fig. 1. Overview of five different technology concepts highlighting their communication interconnection

Table 1. Comparison of middleware concepts

Concept	R1	R2	R3	R4	R5
OPC UA	o	+	+	o	-
OPC UA with Aggregate Server	-	+	o	o	+
OPC UA Pub/Sub over MQTT/AMQP	+	-	x	+	+
OPC UA Pub/Sub over UDP	+	o	+	-	+
Queue-based Message Broker with own construct	+	x	x	+	x
Data Distribution Service	+	+	+	+	o

Legend: + *fulfilled,* - *not fulfilled,* o *partially fulfilled,* x *depends*

handling of messages and data types with DDS: every client requires information about message types it wants to process already at compile time. This contradicts easy configurability (R5).

Even though DDS has minor deficits in R5, it is still the most suitable middleware concept when considering the defined requirements, as can be seen in Table 1. The only alternative which could compete is OPC UA Pub/Sub over MQTT/AMQP, which fulfills every requirement except that it is not yet implemented. However, the need to commit oneself to a solution may render irrelevant in the future as the organizations behind DDS and OPC UA are currently working on an OPC UA/DDS gateway specification (OPC (2018a)).

5. OVERVIEW OF THE PROTOTYPE IMPLEMENTATION

Based on the previous results, a prototypical, DDS-based middleware is implemented. Primary development tasks for the prototype is leveraging all participants to communicate via DDS and integration of a data storage to store and provide historical data. Implementation of a central discovery service or a message broker is not required as DDS is a decentralized communication protocol.

The main goal is to make the prototype technology-independent, i.e., independent of the actual DDS implementation and data storage solution. Therefore, a communication library for participants is proposed. The library abstracts the DDS implementation-specific code to a general interface, which is used by participants to communicate via the middleware. The library allows to

- subscribe to live data,
- publish live data,
- get historic (archived) data, and
- discover all publishers.

OpenDDS [8] was chosen as a DDS system for the prototype since it is open source and well documented. To store large amounts of data and give easy access with efficient query mechanisms, PostgreSQL [9] is used as an additional data storage component connected to the architecture. A schematic overview of the prototype architecture can be seen in Figure 2.

The prototype, which is made up of the communication library and the database adapter, is written in C++. It was tested for x86 (Windows 10 & GNU/Linux Ubuntu) and ARM (Raspbian) platforms.

6. PROTOTYPE EVALUATION

To verify the concept and its implementation, the developed prototype is evaluated. For this purpose, two scenarios, a Proof of Concept (PoC) and a load test, are realized. The PoC test aims to show the general applicability of the prototype. The prototype is rolled out on the myJoghurt

[8] `http://opendds.org/`, accessed 29.08.2019
[9] `https://www.postgresql.org/`, accessed 29.08.2019

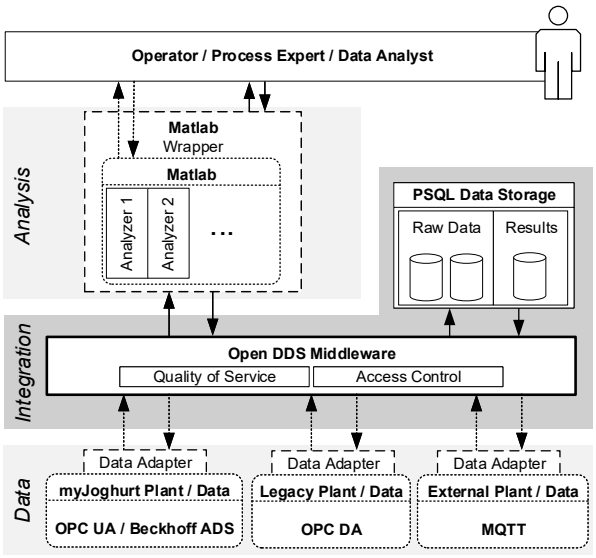

Fig. 2. Schematic overview of the prototype architecture.

Industrie 4.0 prototype plant [10] , which represents a realistic and heterogeneous semi-industrial environment. After the prototype proves to be working correctly, the subsequent load test is conducted. The load test is meant to measure relevant key figures of the implementation. It is conducted on an artificial setup, as described below.

All tests are carried out with DDS QoS set to *reliable* and infinite message lifespan. OpenDDS offers its discovery instance DCPSInfoRepo which is used to associate subscribers and publishers for this evaluation. Throughout all tests, each publisher and subscriber logged the total number of sent respectively received messages. Additionally, during the first PoC and load tests, all received and sent messages are logged individually. The exhaustive logging of all messages was dropped during later tests to reduce log size and overhead.

6.1 Proof of Concept Test

The applicability of the concept is verified by rolling out the implemented prototype on a lab-scale facility consisting of two independent plants. The plants consist of a variety of heterogeneous legacy systems and communicate via different protocols, namely *OPC UA*, *OPC DA*, *MQTT* and *Beckhoff ADS* [11] . This heterogeneity of systems and protocols is a common and representational situation in industrial engineering. The success of this test was proven by a working interconnection of all connected systems and protocols through the DDS middleware. Newly joining participants were discovered correctly and able to communicate. All published messages are received by the intended subscribers and are recorded correctly by the database.

6.2 Load Test

The PoC test does not indicate the reliability and stability of the middleware under high loads. Therefore, a load test determines the key figures of the prototype. As key figures, the maximum possible message rate, as well as the latency of messages, are identified. The first one sets the upper limit for participating publishers in the middleware or their publishing frequency. It indirectly reflects the scalability of the solution. The second one indicates the real-time communication capabilities of the middleware. To determine the maximum message rate on the specific hardware, as well as the latency of the messages at each rate, a latency load test was carried out.

[10] `http://i40d.ais.mw.tum.de/`, accessed 29.08.2019

[11] `https://www.beckhoff.de/default.asp?twincat/tc1000.htm?id=1890306418903071`, accessed 29.08.2019

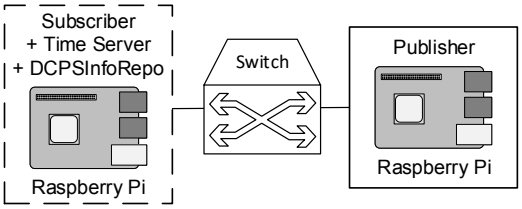

Fig. 3. Setup for latency tests.

To simplify the setup and execution of the tests, dummy publishers and subscribers are used instead of real participants. These publishers do not poll real data from a machine. Instead, they generate arbitrary data as payload (one integer of 4 bytes), which is published with a related timestamp (8 bytes). Upon receiving a message, the subscribers discard the payload and take the current timestamp. Both publishing and receiving timestamps are used to calculate the message latency. The clocks of all participants are synchronized at the beginning of each measurement, and calculated drift rates are used to correct the latency measurements.

The setup consists of one publisher and one subscriber, each running on a Raspberry Pi (RPi) 3 Model B with Raspbian 9. Both are connected via Ethernet to the same network switch (see Figure 3). The subscribing RPi also runs the DCPSInfoRepository and the time server for clock synchronization over NTP. The whole test is replicated on two more RPis connected via a different switch to increase confidence about the correctness of the results. The latency tests are carried out for message rates of 10, 100, 500, 1,000, 2,000 and 10,000 msg/s. Tests were repeated at least 35 times for each rate and lasted seven minutes, respectively.

In the following, the results of the load test for a selected message rate of 2,000 msg/s are shown. A histogram of all recorded values is depicted in Figure 4. The histogram states the average latency $\bar{t}_L$, the standard deviation σ of the latency (σ_L), as well as of the drift measured at the end of each test run (σ_D), and the jitter J. The jitter is calculated as $J = (\sigma_L + \sigma_D)^2$. Figure 4 reveals a small but persistent number of significant latency outliers up to almost seven seconds. The zoom in Figure 4 furthermore highlights the primary peak of measured latencies at around 600 microseconds and indicates that the majority of measurements reside in the millisecond range. For a postulated message lifespan of 2 milliseconds, only 0.33% of the messages would be expired and discarded. Nonetheless, the specification of a concrete lifespan setting is a trade-off between acceptable latencies and allowed message drops.

Results for measurements at all message rates over all repetitions are summarized in Table 2. The table states

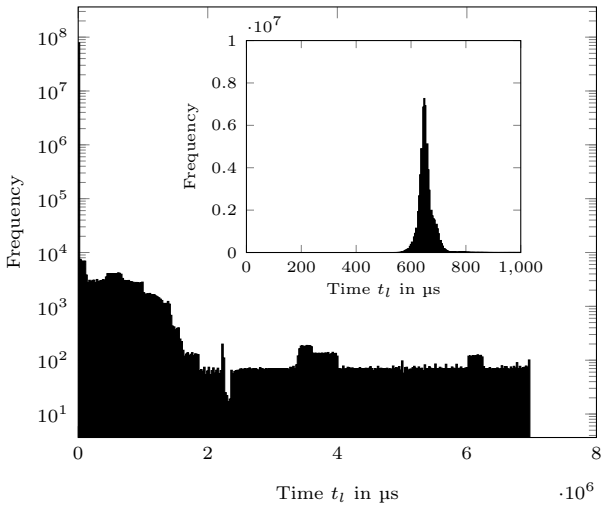

Fig. 4. Histogram of message latencies for the primary publisher/subscriber pair of RPis with a message rate of 2000 msg s^{-1} ($\bar{t}_L = 3066$ µs, $\sigma_L = 78{,}570$ µs, $\sigma_D = 41$ µs, $J = 6{,}179{,}842{,}220$ µs^2, samples = 78,641,375). All data (outer plot, bin size = 23,333 µs, logarithmic frequency axis) and zoom to primary peak (inner plot,bin size = 5 µs).

the minimum, average, and maximum measured latency for each conducted message rate as well as the jitter. Furthermore, it lists the overall number of measurement points, as well as the number and fractions of latencies above a threshold of two milliseconds. The results for the other message rates expose a similar trend as observed for 2000 msg/s: there is always a small number of outliers. Still, the percentage of latencies higher than two milliseconds never exceeds 0.5%.

Measurements with a message rate of 10,000 msg/s expose the upper limit of the possible message rate. 10,000 msg/s could not be achieved, but the rate instead capped at around 4,400 msg/s. One could observe that the publisher and subscriber RPi reached 100% CPU utilization. Therefore, it appears that the prototype reached its maximum achievable performance on the RPi hardware. Measurements on the secondary RPi pair confirmed the results.

Overall, the middleware is reliably capable of handling message rates up to 4,000 msg/s in the given use-case. Measurements for higher message rates are hindered by the hardware limitations of the used RPis, suggesting that the middleware implementation itself can handle higher loads on more powerful hardware. An overall average latency in magnitudes of milliseconds could be measured. However, outliers in magnitudes of around a few seconds diminish the quality of message receiving reliability.

6.3 Evaluation Results

The evaluation results are used to verify if the implementation of the concept meets the considered requirements.

During the PoC test, it was observable that participants, publisher as well as subscriber, could arbitrarily be added and removed from the DDS middleware without any adverse side effects. It is to be expected that the process of adding and removing participants to and from the communication layer at a larger scale of participating systems poses no problem as well. Further on, the load tests confirmed that the communication infrastructure and database could handle significant stress. For the middleware, it should be of no interest if few publishers cause a high load with unrealistic high message rates or many participants with significantly lower message rates. This does not directly prove the fulfillment of *horizontal scalability* (R1) for hundreds or more of participants. Still, it is assumed that the middleware achieves R1 even at larger scales for the two reasons mentioned above.

The demand for usage of *already implemented* (R2) solutions is obeyed as far as possible. Only abstracting layers like the communication interface and database adapter to make the underlying technologies, namely OpenDDS and PostgreSQL, exchangeable are implemented. The adapters make extensive use of existing libraries for their respective communication protocols.

Considering the outliers during the latency tests, the requirement *soft real-time capability* (R3) may not be fulfilled in every scenario. This is due to the magnitude of the outliers and their highly unpredictable manner of occurrence. With a postulated message lifetime of 2 milliseconds, only a maximum of 0.5% of the messages exceeds this limit. Depending on the specific use-case, adjustment of the lifetime setting is a trade-off between acceptable maximum latencies and message loss. Nevertheless, reliable communication with no loss of messages at very low latencies is not possible with the current implementation, which could limit the applicability of the middleware.

The analysis of the created log files was carried out to verify the correct transfer of messages between the systems. All subscribers have received the same number of messages their associated publisher has sent. A total of 25 samples of the exhaustive subscriber and associated publisher logs were randomly selected and examined in detail. It was found that each message is unique, and the correct number of total messages was not falsely achieved by receiving one message twice while losing another one entirely. The results strongly indicate that a *QoS* (R4) level where each sent message should be received precisely once is constantly met.

The prototype is implemented as *configurable* (R5) as possible. The DDS communication, as well as the database storage, are abstracted by interfaces and can, therefore, easily be replaced. The OpenDDS-specific DCPSInfoRepository allows for adding new communication participants without further actions required. Moreover, all implemented publishers and subscribers are configurable as far as possible through configuration files.

The evaluation proved the functioning of the prototype and that it is capable of integrating various protocols. Load tests have shown that the middleware can reliably handle message rates of up to 4000 msg/s on RPis. The chosen QoS settings for DDS prohibit a real-time critical usage but could be changed to most likely support real-time if a *best-effort* QoS is sufficient.

In total, four out of five functional requirements could be fulfilled (R1, R2, R4, and R5), as well as the non-functional requirement R0. R3 could only partially be

Table 2. Summary of latency results for primary RPi pair

Message rate	**Latency t_L**			**Jitter J**	**Total number of measured points**	**Measured latencies** $> 2\,\mu s$	**Fraction of latencies** $> 2\,\mu s$
	min	**avg**	**max**				
(msg/s)	(µs)	(µs)	(µs)	(μs^2)			
10	640	1,046	885,364	32,014,918	382,291	606	0.0016
100	624	1,132	1,676,759	134,688,067	3,778,740	8,227	0.0022
500	416	1,282	2,052,391	386,233,120	18,865,725	56,551	0.0030
1,000	471	1,902	1,287,547	777,661,633	15,078,611	71,247	0.0047
2,000	493	3,066	6,995,701	6,179,842,220	78,641,375	261,629	0.0033
4,400	510	37,677	2,692,251	48,610,062,311	64,387,106	2,597,623	0.0403

fulfilled, which means it may or may not be considered fulfilled depending on two factors. The first is the defined soft deadline for message transfers. The second is the rate of decline of the usefulness of message payloads because of missed deadlines.

7. SUMMARY AND OUTLOOK

With the trend towards Industrie 4.0, production plants offer more process data for further refinement. However, complex issues currently prevent the collection and further processing of this data. Primarily, the need for integrating a highly diverse set of systems and protocols sets up a huge barrier. To overcome this barrier, multiple abstract system architectures and concrete middleware realizations already exist. This work tries to overcome the gap between both categories by comparing multiple concept technologies.

Based on a requirements analysis, five functional requirements are selected, the middleware should fulfill. The technology concept comparison based on the requirements at hand yields that both, OPC UA Pub/Sub over AMQP/MQTT and DDS meet the desired requirements, except that currently, no complete implementations for OPC UA Pub/Sub over AMQP/MQTT are available.

A DDS-based prototype is realized and evaluated in detail. It is deployed to a lab-scale facility demonstrating its functionality. Load tests verified that the middleware is capable of message rates of 4,000 msg/s on RPis. Evaluation of the middleware shows that the presented prototype fulfills the requirements *horizontal scalability* (R1), *already implemented* (R2), *QoS* (R4) and *configurability* (R5). Nevertheless, *soft real-time* (R3) capability cannot be completely fulfilled at the same time as R4. Future works include additional measurements with larger hardware setups as well as a reevaluation as soon as OPC UA Pub/Sub over AMQP/MQTT is available.

Regarding OPC UA, it may be beneficial to check the development status of the mentioned OPC UA/DDS gateway. As of today, OPC UA does not fulfill all QoS requirements, while the implemented and evaluated DDS middleware in this work does not fulfill hard real-time demands. It may be of interest to explore a combination of both systems taking into account the gateway, as mentioned earlier, essentially combining two environments with different demands. OPC UA Pub/Sub over TSN (Pfrommer et al. (2018)) is hard real-time capable and can, therefore, be used on the field level. The business-level uses DDS to gain access to a variety of QoS settings, which may be more important than real-time properties. A gateway combines these otherwise incompatible networks and allows for intercommunication.

REFERENCES

(2013). IEC62264-1-2013: Enterprise-control system integration Part 1: Models and terminology.

(2013). VDI/VDE 2657 Part 1: Middleware in industrial automation: Fundamentals.

(2014). ISO/IEC 19464-2014: Information technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification.

(2015). DDS Specification. URL `https://www.omg.org/spec/DDS/1.4/PDF`. Last accessed 30.08.2018.

(2016). ISO/IEC 20922:2016: Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1.

(2017). OPC Unified Architecture Specification Part 1: Overview and Concepts. URL `https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/#downloadDialog`. Last accessed 30.08.2018.

(2018a). OPC UA/DDS Gateway Specification. URL `https://www.omg.org/spec/DDS-OPCUA/1.0/Beta1/PDF`. Last accessed 24.09.2018.

(2018b). OPC Unified Architecture Specification Part 14: PubSub. URL `https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub/#downloadDialog`. Last accessed 30.08.2018.

Angione, G., Barbosa, J., Gosewehr, F., Leito, P., Massa, D., Matos, J., Peres, R.S., Rocha, A.D., and Wermann, J. (2017). Integration and Deployment of a Distributed and Pluggable Industrial Architecture for the PERFoRM Project. *Procedia Manufacturing*, 11, 896 – 904.

Bisbal, J., Lawless, D., Wu, B., and Grimson, J. (1999). Legacy information systems: issues and directions. *IEEE Software*, 16(5), 103–111.

Crotty, J. and Horrocks, I. (2017). Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company. *Applied Computing and Informatics*, 13(2), 175–183.

García-Valls, M., Domínguez-Poblete, J., and Touahria, I.E. (2018). Using DDS Middleware in Distributed Partitioned Systems. *SIGBED Rev.*, 14(4), 14–20.

Gosewehr, F., Wermann, J., and Colombo, A.W. (2016). Assessment of industrial middleware technologies for the PERFoRM project. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 5699–5704.

Großmann, D., Bregulla, M., Banerjee, S., Schulz, D., and Braun, R. (2014). OPC UA server aggregation The

foundation for an internet of portals. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 1–6.

Pfrommer, J., Ebner, A., Ravikumar, S., and Karunakaran, B. (2018). Open source opc ua pubsub over tsn for realtime industrial communication. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1087–1090. IEEE.

Sauter, T. (2010). The three generations of field-level networks - evolution and compatibility issues. *IEEE Transactions on Industrial Electronics*, 57(11), 3585–3595.

Schleipen, M., Gilani, S.S., Bischoff, T., and Pfrommer, J. (2016). OPC UA & Industrie 4.0 - Enabling Technology with High Diversity and Variability. *Procedia CIRP*, 57, 315 – 320. Factories of the Future in the digital environment - Proceedings of the 49th CIRP Conference on Manufacturing Systems.

Shin, K.G. and Ramanathan, P. (1994). Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1), 6–24.

Theorin, A., Bengtsson, K., Provost, J., Lieder, M., Johnsson, C., Lundholm, T., and Lennartson, B. (2015). An Event-Driven Manufacturing Information System Architecture. *IFAC-PapersOnLine*, 48(3), 547 – 554. 15th IFAC Symposium on Information Control Problems in Manufacturing.

Trunzer, E., Calà, A., Leitão, P., Gepp, M., Kinghorst, J., Lüder, A., Schauerte, H., Reifferscheid, M., and Vogel-Heuser, B. (2019). System architectures for industrie 4.0 applications. *Production Engineering*, 13(3), 247–257.

Trunzer, E., Lötzerich, S., and Vogel-Heuser, B. (2018). *Concept and Implementation of a Software Architecture for Unifying Data Transfer in Automated Production Systems: Technologies for Intelligent Automation*, 1–17.