Tunnelling and Mirroring Operational Technology Data with IP-based Middlewares

1st Patrick Denzler
Automation Systems Group
TU Wien
Vienna, Austria
patrick.denzler@tuwien.ac.at

2nd Daniel Ramsauer
Automation Systems Group
TU Wien
Vienna, Austria
daniel.ramsauer@tuwien.ac.at

3th Wolfgang Kastner
Automation Systems Group
TU Wien
Vienna, Austria
wolfgang.kastner@tuwien.ac.at

***Abstract*—Accessing runtime information originated from Operational Technology (OT) communication protocols by Information Technology (IT) applications remains a challenging task. Middlewares and gateways could provide a possible solution to bridge this OT/IT gap. In this context, this paper examines the feasibility of interconnecting legacy OT protocols via different middlewares and gateways by presenting two simplified scenarios. Each scenario exemplifies a possible use-case by either mimicking bidirectional tunnelling or propagating (mirroring) of automation data. Both scenarios use OPC Unified Architecture (OPC UA) and Message Queuing Telemetry Transport (MQTT) and a respective gateway prototype. The paper concludes by outlining future research challenges by discussing the limitations of the selected scenarios related to configuration, deployment and scalability.**

***Index Terms*—Industrial communication, Industrial Internet of Things, middleware, gateway**

I. Introduction

Accessing and exchanging runtime information is the central element in industrial automation [1]. However, a typical hierarchical automation system architecture is not promoting an ideal exchange of data. The reason lies in the diverse types of used technology and software, often exhibited in the automation pyramid [2] (cf. Figure 1). Referred to as operational technology (OT), the lower three levels represent sensors and actuators, followed by controllers (e.g., programmable logic controllers (PLC)) and Supervisory Control and Data Acquisition (SCADA) systems. The OT levels are responsible for the interaction with the underlying technical process and the processing of automation functions (e.g., open/closed-loop control), respectively [3]. Automation technology and industrial communication systems fulfil the stringent OT requirements, e.g., on real-time and safety. In contrast to OT, the top two information technology (IT) levels use off-the-shelf components and communication systems compatible with the Internet Protocol (IP) suite, part of any enterprise / IT environment. Manufacturing Execution System (MES) and Enterprise-Resource-Planning (ERP) tool suites are typical applications in IT for processing and abstracting the data from the lower levels. The differences in technology and requirements create the OT/IT gap and limit the exchange of data [4].

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation.

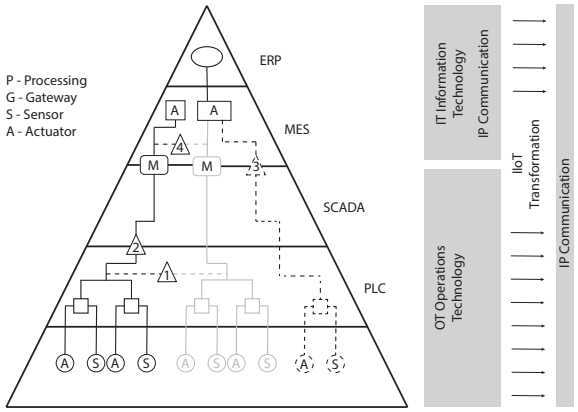

Fig. 1. Automation Pyramid

Initiatives such as the Industrial Internet of Things (IIoT) addresses the (OT/IT) gap by flattening the architecture and introducing all-IP communication at all levels of the pyramid. Such an architecture would allow a cloud application directly accessing a single sensor. Newer technologies such as fog computing [5], time-sensitive networking (TSN) [6] and middlewares (e.g., OPC Unified Architecture (OPC UA) [7], Data Distribution Service (DDS) [8] or Message Queuing Telemetry Transport (MQTT) [9]) could provide the means to implement this vison [10].

An often neglected fact is that the industrial sector looks back on a plethora of industrial communication technologies, which has grown over the last four decades [11]. Therefore, it is not uncommon to find a mixture of classical wired fieldbus systems, Ethernet-based approaches and wireless solutions [12]. Each protocol works ideally on its own, but their design does not foresee the exchange of information with other protocols since they all have their own application models deeply embedded. Realising IIoT however, would require full interoperability between all elements in the automation pyramid [13]. Exchanging the entire old technology is not feasible as it would require a considerable amount of time and money. Therefore, interim solutions are required to mitigate the transition [14].

Gateways could be such a solution. Widely used in automation, gateways connect protocols with software applications and provide the necessary abstraction. However, gateways are complex artefacts and not simple to develop and maintain. The considerable number of possible combinations between applications and protocols would further create an unmanageable mix of gateways. Middlewares could help to reduce the number of gateways by acting as a replacement. For example, the OPC UA middleware offers drivers and APIs to connect legacy protocols and provides IT applications with the required abstraction based on means to define common information models [7]. In the very best case, only a handful of middlewares and vertical gateways would be necessary to connect legacy protocols with IT.

In this context, the article reviews the feasibility of using gateways in combination with middlewares to improve interoperability in automation. For this purpose, an OPC UA / MQTT gateway prototype was built and applied in two scenarios that would require connecting two or more middlewares. Each of the scenarios revealed restrictions, e.g., in configuration, scalability or latency that require careful consideration when using middleware gateways. The results contribute to ongoing research in the areas of automation systems and IIoT.

Section II presents background information about gateways and connectivity in automation, followed by Section III, that introduces the middleware gateway concept and related work. The architecture of the gateway prototype in Section IV lays the ground for the two presented and evaluated scenarios in Section V and VI. Section VII discusses the findings, and Section VIII concludes the paper.

II. Gateways in automation

Gateways have a long history in automation, either bridging horizontally varying communication protocols or act vertically as an abstraction layer for information transfer to high-level IT applications [1] (cf. Figure 1).

In the lower levels of the automation pyramid, horizontal gateways remain an exception. There were suggestions for gateways since the early days of automation when fieldbus systems replaced parallel cabling [15]. However, those turned out to be challenging to implement as dedicated automation networks (e.g. Controller Area Network (CAN), PROFIBUS or INTERBUS) are solutions not designed for information exchange with other protocols [15].

With the advent of the Internet, uniform Ethernet-based networks also appeared in the automation environment and let gateways seem to be a relic of the past. However, due to the missing real-time capabilities of the Ethernet standard, the domain witnessed the development of several new dedicated real-time protocols and reintroduced the need for gateways [16], [17]. Protocols, e.g., PROFINET isochronous real-time (IRT) or EtherCAT, are incompatible, as each uniquely adjusts the OSI layers to be realtime capable. This peculiarity complicates the development of horizontal gateways. Additionally, research has shown that horizontally connecting OT protocols can pose a security risk [1].

There are two types of vertical gateways (cf. Figure 1) that transfer automation data to the IT domain. Either by directly attaching protocols to enterprise tools and provide the necessary data abstraction [18] or connect to a middleware. The first case creates a unique combination between a protocol and an IT application and is therefore not reusable for other pairings. Connecting to a middleware instead reduces this effect as middlewares provide standard interfaces for IT applications [14]. In some cases, middlewares (e.g. ,OPC UA) also provide APIs and drivers that allow connecting protocols directly [7].

The most complex gateways in automation are horizontal middleware gateways that allow the data exchange between middlewares. Each middleware might use a different communication pattern, i.e., client/server (OPC UA), broker-based (MQTT) or publish/subscribe (DDS) and different transportation protocols ranging from Transmission Control Protocol (TCP)/IP, User Datagram Protocol (UDP) to HTTP and Web Services [3]. Additionally, middlewares provide more advanced features, (e.g., definition of Quality of Service, or means for information modelling) that need to be adequately handled by a gateway. Despite their complexity, middleware gateways play an essential role in interconnecting OT and IT.

III. Middleware gateway concept

Considering the various possibilities to connect legacy protocols with the IT-level applications, a mix of middlewares and gateways might be a rational solution. Based on that idea, the Industrial Internet Consortium (IIC) investigated which middlewares are most suitable for such a task [19]. Their results indicate that OPC UA, DDS, oneM2M and Web Services are the best candidates, as they are widely used in various domains and provide advanced functionalities. Moreover, the concept, as shown in Figure 2, foresees adding middleware gateways and connecting legacy protocols or less common middlewares to the identified middlewares. This combination would improve the interoperability between OT/IT significantly. Despite the feasibility of the IIC concept, there is little research done about the possible challenges.

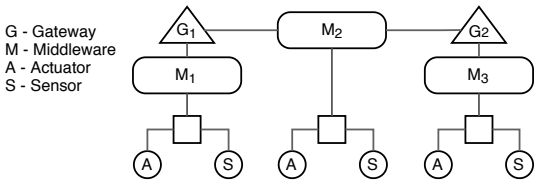

Fig. 2. Middleware gateways concept

The situation presents itself differently when focusing on the middleware gateways. Research proposed several prototypes to connect numerous types of middlewares in various use cases. For example, [20] uses a simplified OPC UA / MQTT gateway based on Node-RED in a ”loom machines” use case or [21] proposes a Phyton based OPC UA / Modbus gateway. Authors around [22] implemented a gateway between

OPC UA and VSOMEIP (Scalable service Oriented Middleware over IP) to cover the automotive context, and other solutions approach middleware gateways from a software-defined perspective [23]. Some designs aim to bridge protocols such as MQTT, DDS, and Websockets messages into HTTP to reduce overhead [24].

The Object Management Group (OMG) took a more general approach by publishing a detailed OPC UA / DDS gateway specification [25] in 2019. Two internal configurable bridges allow the gateway to connect the client/server (OPC UA) and publish/subscribe (DDS) middlewares with their advanced features. Based on the specification, authors presented implementations and evaluations of prototypes with reduced functionality [26]. In a recent use case study to consolidate SCADA functionality onto a fog computing platform, the usage of such a prototype gateway revealed open issues regarding deployment, configuration, security and latency [27].

In the following sections, the paper presents two scenarios that represent possible use cases when connecting legacy protocols with different middlewares that are interconnected by middleware gateways. The first scenario mimics the bidirectional tunnelling of automation data from one OT protocol through different IP middlewares to another OT protocol. In the second scenario, the automation data is propagated "mirrored" from one OT protocol to (an)other protocol(s) via different IP middlewares. By assessing the limitations of each scenario, the intent is to gain knowledge and understanding of the challenges involved and evaluate the feasibility of the proposed concept.

IV. Gateway architecture

In order to facilitate the scenarios, a bidirectional OPC UA/MQTT gateway, leaning on the OMG gateway specification [25] was implemented. One side connects to OPC UA servers and clients and processes their requests and responses, while the other side of the gateway connects to a MQTT broker. The internal OPC UA client and server as well as an MQTT client handle the internal bridging process (cf. Figure 3). A configuration file defines the necessary internal gateway mappings and routings.

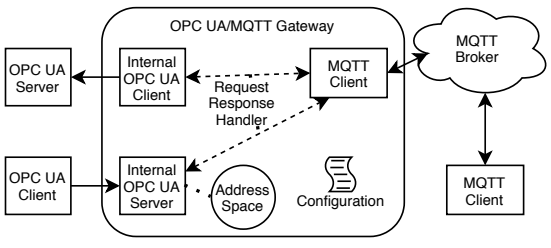

Fig. 3. Architecture overview

A. Middlewares (OPC UA and MQTT)

OPC UA is an object oriented client/server-based information exchange standard for industrial automation systems, that supports various IP-based networking technologies.. The request-response message pattern allows clients to access the server's services, for manipulating *Nodes* by addressing their unique *NodeId*. These services split into OPC UA's *Service Sets*, which provide access to the server's address space model, that represents *Nodes* to a client. Each *Node* can contain *Variables*, that describe the content type, and *Methods*, that provide lightweight and stateless functions [7].

MQTT is a TCP/IP based publish/subscribe protocol which enables communication for constrained devices in unreliable networks. MQTT provides so called *Topics* to which clients can publish data and subscribe for updates. A client publishes payload-agnostic messages to the MQTT-Server (*Broker*), which handles the distribution to every subscribed client.

B. Gateway component details

In order to bridge messages, the internal OPC UA server needs to represent MQTT topics and their corresponding data types within its address space. Changes made to these nodes will be reflected as published messages to the MQTT broker, so that subscribed MQTT applications and other gateways will receive these updates. Complementary, the internal OPC UA client provides access for MQTT applications to issue requests to OPC UA servers.

In order to connect OPC UA and MQTT, the gateway internally implements a request-response pattern. This pattern foresees one typed request topic and one corresponding typed-response topic. Therefore, each request and response pair is required to have a unique identifier (id). As every gateway issues requests over the same request-specific MQTT topic (e.g., write-request-topic), this unique id ensures, that the initiator of the request processes the generated response. The gateway's internal service ("RequestResponseHandler") handles the entire id mapping.

Another task of the "RequestResponseHandler" is to manage the requests issued from OPC UA to MQTT and vice versa. After forwarding such a request, the service waits for the response and propagates it back to the request initiator. As MQTT requires byte-encoded messages, the "RequestResponseHandler" uses so called "Wrappers" for serialization. Depending on the request's source, the service either wraps or unwraps a request/response as necessary.

C. Gateway configuration

Another essential part of the gateway is the configuration file that defines the internal and external mappings. The configuration specifies the typed MQTT topics represented by the gateway's internal OPC UA server. Moreover, for enabling MQTT to access *Nodes* of OPC UA servers connected to the gateway, the configuration defines the network addresses and *Nodes* of those servers. Furthermore, the configuration allows specifying scenario-specific nodes and their corresponding OPC UA servers.

D. Implementation details

The gateway prototype implementation uses the SDKs, Eclipse Paho v1.2.1 (MQTT) and Eclipse Milo v0.2.5 (OPC UA) and an XML-encoded configuration file [28], [29].

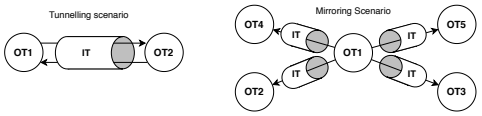

Fig. 4. Simplified scenarios

V. Middleware scenarios

The two scenarios (”Tunnelling” and ”Mirroring”), represent possible use cases when connecting legacy protocols with different middlewares (cf. Figure 4). The Tunnelling scenario connects two OT protocols by using an IT based middleware for transporting the issued requests and resulting responses. The Mirroring scenario aims to send the data from one to potentially multiple other OT protocols by using an IT based middleware for data delivery. To reduce the complexity, the scenarios are limited to the OPC UA and MQTT middlewares and the presented prototype gateway.

A. Tunnelling scenario

In this scenario, an OPC UA client issues requests to a remote OPC UA server and receives responses (cf. Figure 5). The client establishes a connection to the gateway and issues a request. This request is then forwarded/converted by the gateway to MQTT. The MQTT broker distributes the request to every subscribed gateway, which forwards it to the designated OPC UA server. After processing the request, the OPC UA server returns a response, which the gateway receives and publishes. The initiator gateway forwards this response to the initiating OPC UA client, which concludes this scenario. The scenario's message flow would apply to both directions.

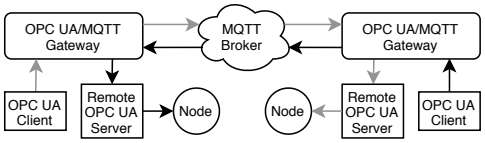

Fig. 5. Tunnelling scenario

Figure 6 depicts the respective message flow of the described Tunnelling scenario.

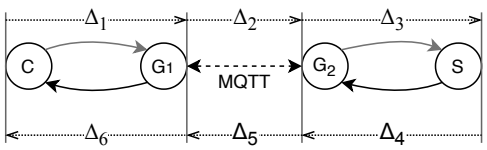

Fig. 6. Tunnelling scenario message flow

For this scenario, a prerequisite is the configuration of the two gateway's internal OPC UA servers holding a list of remote OPC UA server addresses and corresponding nodes.

B. Tunnelling scenario limitations

A significant limitation of this scenario is based on the fact that OPC UA is client/server-based. In detail, a request issued by an OPC UA client can only contain nodes located at the same remote OPC UA server. The reason lies in the fact that client/server communication is not intended to split messages or combine multiple responses. In OPC UA, for example, the ordering of responses has to correlate to the order of nodes within the request. Therefore, the OPC UA client would have to issue single requests for each node.

C. Mirroring scenario

The second scenario allows mirroring specific nodes of an OPC UA server to configured remote OPC UA servers (cf. Figure 7). In this case, an OPC UA client issues a write request to an OPC UA server, which processes the request, updates the node and returns a response. As the gateway maintains a subscription on this node, the OPC UA server will send this update to the gateway. The gateway creates a new write-request and publishes it to MQTT. The corresponding gateway receives this request and forwards it to the designated OPC UA server. After updating the node, the server returns a response, which the initiator gateway receives, and this concludes the scenario. If configured correctly, the Mirroring scenario allows distributing the node change to a multitude of remote OPC UA servers.

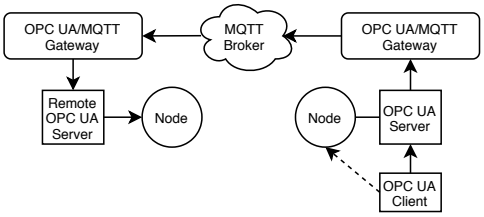

Fig. 7. Mirroring scenario

Figure 8 depicts the respective message flow of the described Mirroring scenario.

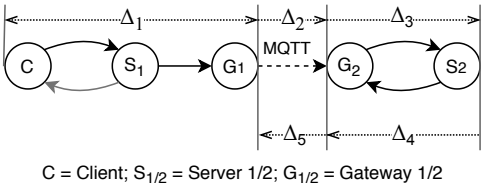

Fig. 8. Mirroring scenario measurements

Prerequisites for this scenario is the configuration of the gateway's internal OPC UA client holding a list of remote OPC UA server addresses and corresponding nodes. Additionally, this client has subscriptions to configured nodes, hosted by the connected OPC UA server. In this scenario, the OPC UA client is not aware that its changes of its nodes will be mirrored to other OPC UA servers.

D. Mirroring scenario limitations

As mentioned, the Mirroring scenario only distributes write-requests; hence it is not possible to manipulate the address space of any remote OPC UA server. The responses of write-requests are not available to the initiating OPC UA client. The reasons are twofold; firstly, the client does not expect a subsequent response other from the original request. Secondly, the responses from other gateways are not further propagated by initiator gateway because the client does not require it.

VI. Evaluation

Each message flow was measured on actual implementation to evaluate the described scenarios. The hardware setup included one desktop PC and one laptop, both running on Windows 10, and one Raspberry Pi 4B using Raspberry Pi OS. The PC's specifications include 16GB RAM and a quad-core CPU with 3,4GHz clock speed; the laptop also features 16GB RAM and a quad-core CPU with 2,7GHz clock speed. To accommodate the scenarios on both computers, one gateway prototype and one OPC UA server was installed and deployed. The Raspberry Pi utilised Docker for the MQTT broker functionality, which was provided by the Eclipse-Mosquitto image. A Gigabit switch provided the network connections.

For monitoring the message sequences (and measuring time), Wireshark was used, and the machines were time-synchronised by using the network time protocol (NTP). For both scenarios, a boolean value was changed, and this change was sent over MQTT to the remote OPC UA server based on the scenario. The request's size was 233 bytes from OPC UA and 1892 bytes forwarded to MQTT for the "Tunnelling" scenario and 215 bytes and 1887 bytes for the "Mirroring" scenario, respectively. Every wrapping and unwrapping of requests and responses included adding or parsing gateway specific information, like handling the server id, which influences the conversion times. This request was issued multiple times. The provided measurements show the average values gathered from this process.

A. Measuring results

The values in Table I represent the message flow as visualized in Figure 6. Δ_1 symbolizes the time difference between the OPC UA client issuing a request and the gateway publishing this request to the MQTT topic. Δ_3 represents the time needed for the gateway to receive a published request and forward it to the remote OPC UA server. Δ_4 shows the amount of time necessary to convert the remote OPC UA's server response to a MQTT publish request. In the last step, the client receives the answer from the gateway (Δ_6). Δ_2 and Δ_6 are strongly depended on the MQTT infrastructure and therefore not measured.

Similar to the measured times in the Tunnelling scenario, Table II shows the time values of the "Mirroring" message flow, as shown in Figure 8. Δ_1 includes issuing a write-request from the OPC UA client and updating this node in the address space of the OPC UA server. Additionally, Δ_1 covers the resulting subscription update of the gateway's internal OPC UA server and the publishing to the corresponding MQTT topic. Δ_3 represents the retrieving of this request and updating the specified node. The scenario concludes with Δ_4, which symbolizes the conversion of the OPC UA server's response and the publishing of this response to the corresponding MQTT topic. Δ_2 and Δ_5 are strongly depended on the MQTT infrastructure and are not covered.

TABLE I
Delta times in seconds, Tunnelling scenario

	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6
Request	0.0489s		0.0124s			
Response				0.06390s		0.0074s

TABLE II
Delta times in seconds, Mirroring scenario

	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5
Request	0.54097s		0.01241s		
Response				0.00697s	

VII. Discussion

The results obtained from the scenarios show that the use of a mixture of middleware and gateways is a possible solution to connect OT protocols with IT applications. However, the limitations of the individual scenarios allow some conclusions for the use of gateways that need further clarification. While the gateway implementation was not the main focus, it affected both scenarios—the required in-depth knowledge about each middleware to program a gateway directly transfers into the configuration issue. The intended node communication via a gateway requires the specification of e.g., addresses, topics or datatypes prior deployment in the configuration file. Both scenarios indicate that the complex and static configuration limits the use of the gateway. It appears necessary to automate the configuration file generation if gateway instances are deployed dynamically, e.g., on fog or edge devices. This finding would confirm the challenges described in [27], [30]. Moreover, the variations of available middleware program language implementations (e.g., OPC UA is available in C, JAVA, or Python) increases the complexity for implementation, configuration and deployment of gateways.

Another dependency exists between scalability and the intended node communication. While the second scenario allows the distribution to several severs and connected nodes, the first one is limited to the nodes of one server. Unquestionably, the source of the limitation lies in the classical way of a client/server architecture of OPC UA (excluding the newly published OPC UA Publish/Subscribe specification). However, from the broader perspective of connecting multiple middlewares with gateways, such limitations require careful consideration before implementation. A similar limitation applies related to message transfer and latency. The forwarding scenario delivers messages faster as the gateway simply

forwards them, while in the tunnelling scenario, the gateway establishes a client/server connection first. In combination with the quality of service of the transporting middleware, message timings might become unpredictable. Therefore, applications that involve safety aspects might not be realised with such a concept.

Besides, security was not a centre topic within explored scenarios; the experience indicates that security risks are related to the middlewares not the gateways. The architecture of the gateway prototype would require access by an intruder, e.g., internal message exchange. As the implemented gateway leans on the OMG specification, a similar conclusion could be drawn for the OPC UA / DDS gateway.

Summarised, the findings indicate that the middleware/gateway concept could be an interim solution to close the OT/IT gap for specific scenarios. However, the configuration issue needs to be addressed first, as the involved complexity exceeds the general knowledge of a typical middleware user.

VIII. Conclusion

The paper explores the feasibility of interconnecting OT protocols via different middlewares and gateways by presenting two simplified connection scenarios. Each scenario exemplifies a possible use case for connecting protocols by either mimicking bidirectional tunnelling or propagating (mirroring) of automation data. The found limitations in each scenario point out challenges related to configuration, deployment and scalability of the concept. Further studies should include research for solving the identified challenges (e.g., automatic model-transformations addressing semi-automatic configuration) and investigating possible further limitations of gateways in highly distributed and heterogeneous environments (e.g., fog or edge).

References

[1] T. Sauter and M. Lobashov, "How to Access Factory Floor Information Using Internet Technologies and Gateways," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 699–712, 2011.

[2] T. J. Williams, "The purdue enterprise reference architecture," in *Proceedings of the JSPE/IFIP TC5/WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing*, ser. DIISM '93. NLD: North-Holland Publishing Co., 1993, pp. 43–64.

[3] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, March 2017.

[4] Y. Zhang, C. Qian, J. Lv, and Y. Liu, "Agent and cyber-physical system based self-organizing and self-adaptive intelligent shopfloor," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 737–747, 2017.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 13–16.

[6] W. Steiner and S. Poledna, "Fog computing as enabler for the industrial internet of things," *e & i Elektrotechnik und Informationstechnik*, vol. 133, no. 7, pp. 310–314, Nov. 2016.

[7] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Springer Science & Business Media, 2009.

[8] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference*, pp. 200–206, 2003.

[9] T. Sauter, S. Soucek, W. Kastner, and D. Dietrich, "The Evolution of Factory and Building Automation," *IEEE Industrial Electronics Magazine*, vol. 5, no. 3, pp. 35–48, 9 2011.

[10] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-defined industrial internet of things in the context of industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, 2016.

[11] K. F. Tsang, M. Gidlund, and J. Åkerberg, "Guest editorial industrial wireless networks: Applications, challenges, and future directions," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 755–757, 2016.

[12] S. Vitturi, P. Pedreiras, J. Proenza, and T. Sauter, "Guest editorial special section on communication in automation," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1817–1821, 2016.

[13] Harp, Derek R and Gregory-Brown, Bengt, "IT / OT Convergence Bridging the Divide," *NexDefense*, p. 23, 2015.

[14] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 3 2015.

[15] G. Wood, "Survey of LANs and standards," *Computer Standards & Interfaces*, vol. 6, no. 1, pp. 27 – 36, 1987.

[16] P. Danielis, J. Skodzik, V. Altmann, E. B. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–8.

[17] J. Jasperneite, J. Imtiaz, M. Schumacher, and K. Weber, "A proposal for a generic real-time ethernet system," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 2, pp. 75–85, 2009.

[18] T. Sauter, "The continuing evolution of integration in manufacturing automation," *IEEE Industrial Electronics Magazine*, vol. 1, no. 1, pp. 10–19, 2007.

[19] Industrial Internet Consortium, "Industrial Internet Reference Architecture (IIRA)," *Industrial Internet Consortium*, 2015.

[20] R. Dionísio, S. Malhão, and P. Torres, "Development of a smart gateway for a label loom machine using industrial iot technologies," *Int. journal of online and biomedical engineering*, vol. 16, no. 4, pp. 6–14, 2020.

[21] R. R. Pena, R. D. Fernandez, M. Lorenc, and A. Cadiboni, "Gateway opc ua/modbus applied to an energy recovery system identification," in *2019 18th Workshop on Information Processing and Control, RPIC 2019*, 2019, pp. 235–240.

[22] A. Ioana and A. Korodi, "Vsomeip - opc ua gateway solution for the automotive industry," in *Proceedings - 2019 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC*, 2019.

[23] Z. Jiang, Y. Chang, and X. Liu, "Design of software-defined gateway for industrial interconnection," *Journal of Industrial Information Integration*, vol. 18, 2020.

[24] M. A. A. da Cruz, J. J. P. C. Rodrigues, P. Lorenz, P. Solic, J. Al-Muhtadi, and V. H. C. Albuquerque, "A proposal for bridging application layer protocols to http on iot solutions," *Future Generation Computer Systems*, vol. 97, pp. 145–152, 2019.

[25] Object Management Group. OPC UA/DDS Gateway Version 1.0. (2020, October 15). [Online]. Available: https://www.omg.org/spec/DDS-OPCUA/1.0

[26] R. Endeley, T. Fleming, N. Jin, G. Fehringer, and S. Cammish, "A smart gateway enabling opc ua and dds interoperability," in *Proceedings - 2019 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Internet of People and Smart City Innovation, SmartWorld/UIC/ATC/SCALCOM/IOP/SCI 2019*, 2019, pp. 88–93.

[27] P. Denzler, J. Ruh, M. Kadar, C. Avasalcai, and W. Kastner, "Towards Consolidating Industrial Use Cases on a Common Fog Computing Platform," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 172–179.

[28] Eclipse. Paho Github Repository. (2020, October 15). [Online]. Available: https://github.com/eclipse/paho.mqtt.java

[29] ——. Milo Github Repository. (2020, October 15). [Online]. Available: https://github.com/eclipse/milo

[30] H. M. Park and J. Wook Jeon, "Opc ua based universal edge gateway for legacy equipment," in *IEEE International Conference on Industrial Informatics (INDIN)*, vol. 2019-July, 2019, pp. 1002–1007.